Preference Learning - Part II

1. Principle Component Analysis (PCA)

2. K-mean clustering with/without normalization

3. Multiple regression

4. Groups' Price Responsiveness

In [2]:
```
%matplotlib inline
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import json
from pandas.io.json import json_normalize #package for flattening json in pandas df
import matplotlib.pyplot as plt
from datetime import date, timedelta, datetime
```

1. Principle Component Analysis

We first focus on the non-event data. For each hour,

(1) the first approach is we do principle component analysis and get several explanable top PCs and then use the top PCs to do K-mean clustering to find out groups;

(2) the second approach is forget about PCA, and start by drawing histogram of distribution on a temp, consumption, occurence 3D space, and using GMM to fit the data and find out the classes.

In [ ]:
```
# Import all preprocessed data necessary for the analysis
df_tou1h = pd.read_csv("C:\\Users\\Rockwell\\Desktop\\Paper4\\data_collection\\data_tables\\Consumption_tou2013_1h.csv")
df_Ntou1h = pd.read_csv("C:\\Users\\Rockwell\\Desktop\\Paper4\\data_collection\\data_tables\\Consumption_Ntou2013_1h.csv")
df_wea1h = pd.read_csv("C:\\Users\\Rockwell\\Desktop\\Paper4\\data_collection\\weather\\LondonWeather2013_interpolated.csv")
df_tariff_1h = pd.read_csv("C:\\Users\\Rockwell\\Desktop\\Paper4\\data_collection\\data_tables\\df_tariff_1h.csv")
```

In [3]:
```
import os
os.getcwd()
```

Out[3]:
```
'/Users/Rockwell/Documents/GitHub/Demand-Response'
```

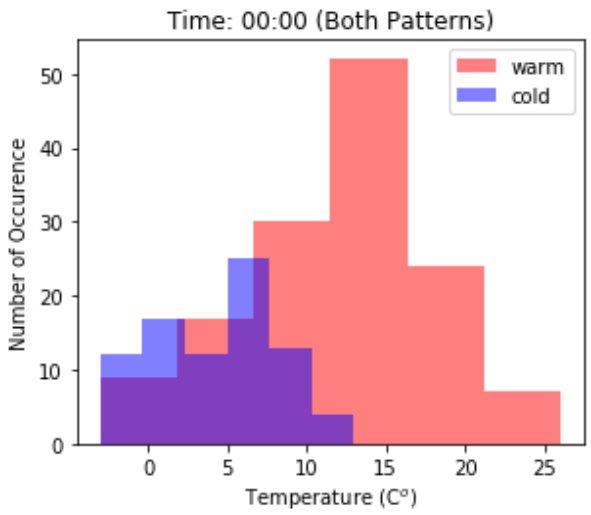
In [4]:
```
# for ios system, import all data necessary for the analysis
df_tou1h = pd.read_csv('/Users/Rockwell/Desktop/PhD/Paper4PreferenceLearning/data/Consumption_tou2013_1h.csv')
df_Ntou1h = pd.read_csv('/Users/Rockwell/Desktop/PhD/Paper4PreferenceLearning/data/Consumption_Ntou2013_1h.csv')
df_wea1h = pd.read_csv('/Users/Rockwell/Desktop/PhD/Paper4PreferenceLearning/data/LondonWeather2013_interpolated.csv')
df_tariff_1h = pd.read_csv('/Users/Rockwell/Desktop/PhD/Paper4PreferenceLearning/data/df_tariff_1h.csv')
```

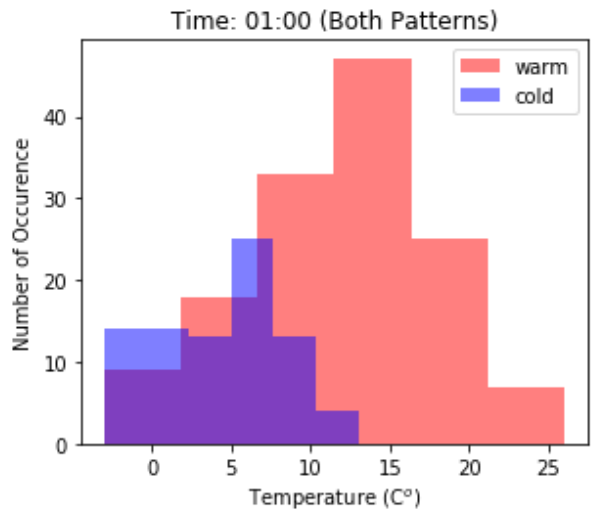
In [5]:
```
# first, create a list of days that belongs to event days
event_days = set()
event_series = df_tariff_1h[df_tariff_1h.Event_tags.notnull()].GMT
for i in event_series:
    event_days.add(datetime.strptime(i[:10], "%Y-%m-%d" ).date()) # add all event dates to the set
df_help = pd.DataFrame(pd.to_datetime(df_tariff_1h.GMT).dt.date) #str to datatime and extract date then make
 it to dataframe
# df_help[df_help['GMT'].isin(event_days)] #this shows the event days
# we can use ~df_help['GMT'].isin(event_days) to generate any non-flexible period items

# create TOU and non-TOU demand data in non-flexible hours
df_wea1h_nf = df_wea1h[~df_help['GMT'].isin(event_days)]
df_Ntou1h_nf = df_Ntou1h[~df_help['GMT'].isin(event_days)]
df_tou1h_nf = df_tou1h[~df_help['GMT'].isin(event_days)]
```

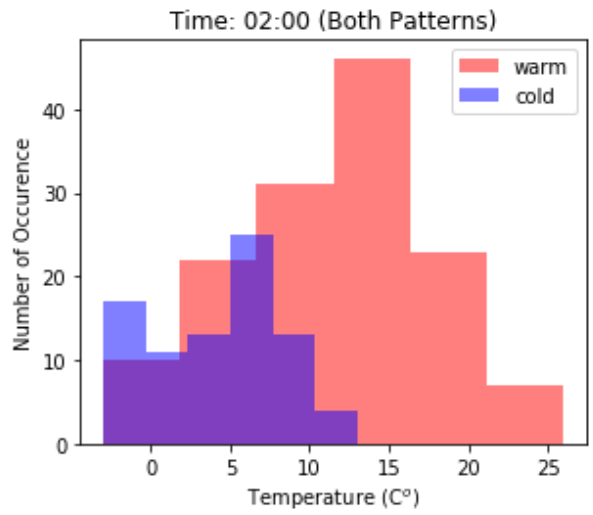
In [6]:
```
# seperate the above data set based on the seasonal effect
# i.e., months of 11, 12, 1, 2, 3 are in a group - cold season
# months of 4, 5, 6, 7, 8, 9, 10 are in another group - warm season
cold_season = [11, 12, 1, 2, 3]
warm_season = [4, 5, 6, 7, 8, 9, 10]
df_help_season = pd.DataFrame(pd.to_datetime(df_wea1h_nf.GMT).dt.month)
df_wea1h_nf_cold = df_wea1h_nf[df_help_season['GMT'].isin(cold_season)]
df_wea1h_nf_warm = df_wea1h_nf[df_help_season['GMT'].isin(warm_season)]
df_Ntou1h_nf_cold = df_Ntou1h_nf[df_help_season['GMT'].isin(cold_season)]
df_Ntou1h_nf_warm = df_Ntou1h_nf[df_help_season['GMT'].isin(warm_season)]
df_tou1h_nf_cold = df_tou1h_nf[df_help_season['GMT'].isin(cold_season)]
df_tou1h_nf_warm = df_tou1h_nf[df_help_season['GMT'].isin(warm_season)]
```

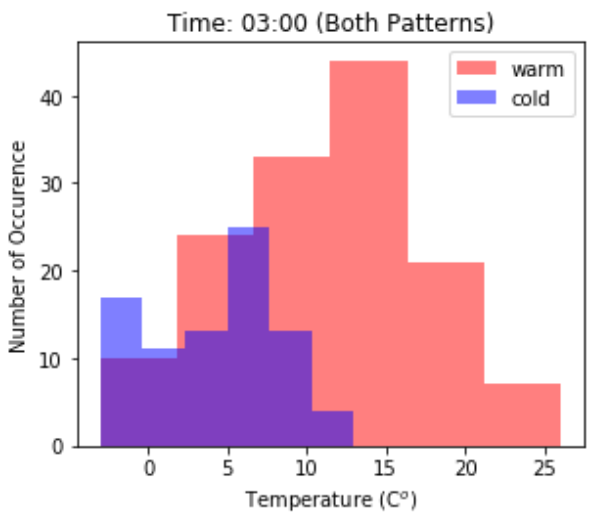
(0) Some basic temperature distribution of the two groups are given below

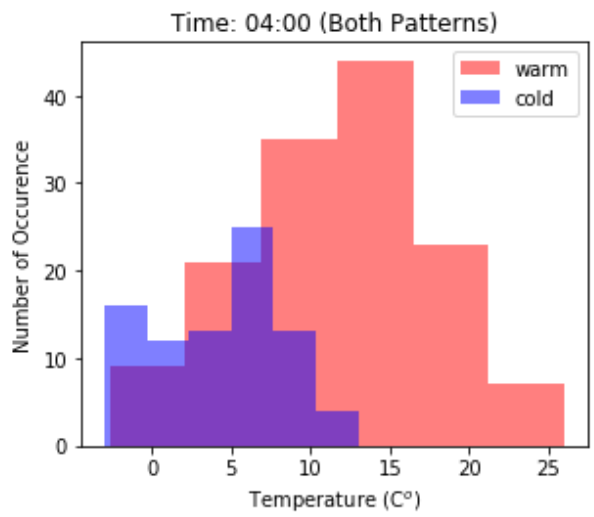
In [6]:
```
# temperature distribution of different hours of a day in a year
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        ax.hist(df_wea1h_nf_warm[df_tou1h_nf_warm.GMT.str.contains('0' + str(i) + ':00:00')]['TempC'], bins=6
, color = 'red', alpha = 0.5, label = 'warm')
        ax.hist(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]['TempC'], bins=6
, color = 'blue', alpha = 0.5, label = 'cold')
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Both Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Number of Occurence')
    else:
        ax.hist(df_wea1h_nf_warm[df_tou1h_nf_warm.GMT.str.contains(str(i) + ':00:00')]['TempC'], bins=6, colo
r = 'red', alpha = 0.5, label = 'warm')
        ax.hist(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['TempC'], bins=6, colo
r = 'blue', alpha = 0.5, label = 'cold')
        ax.set_title('Time: ' + str(i) + ':00' + ' (Both Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Number of Occurence')
plt.tight_layout()
```

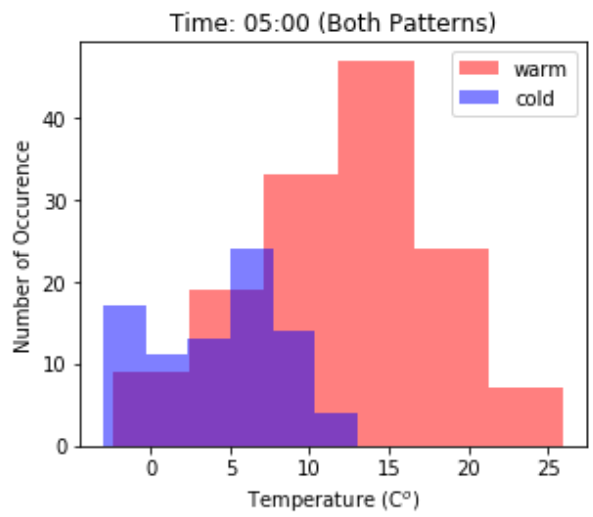

Time: 00:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

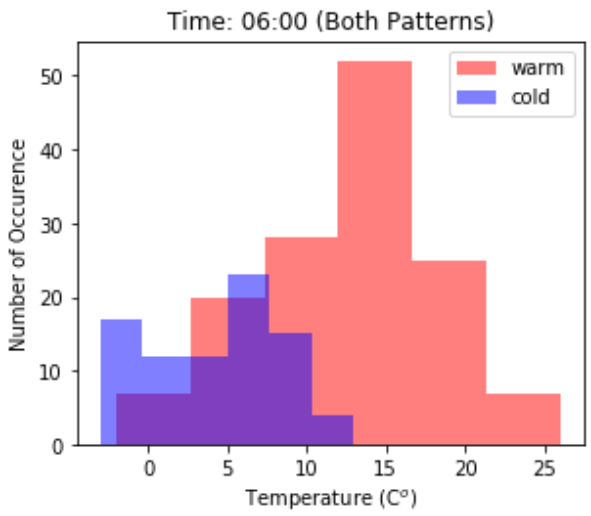

Time: 01:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

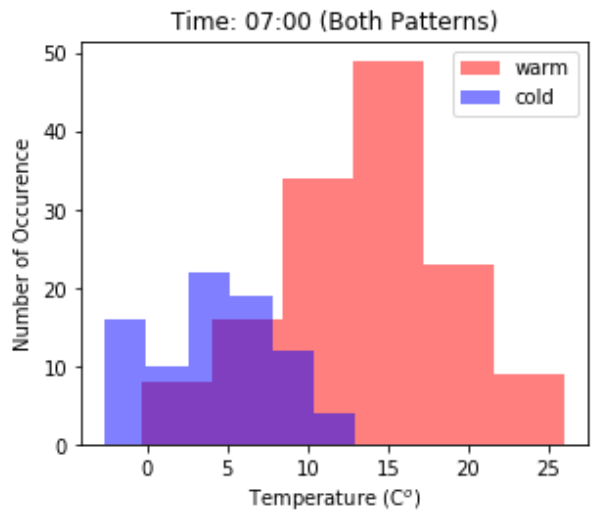

Time: 02:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

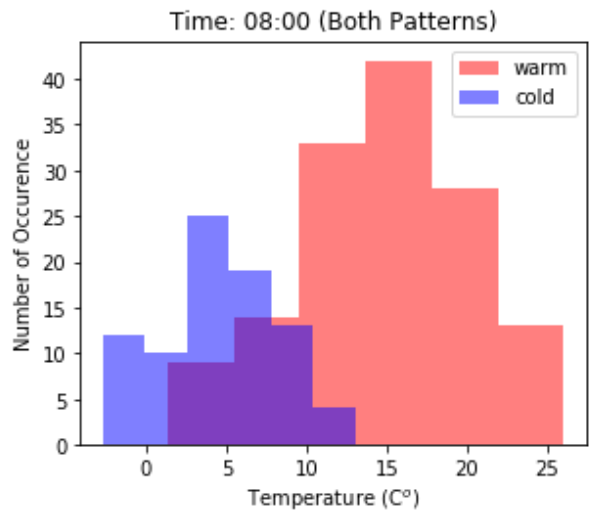

Time: 03:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

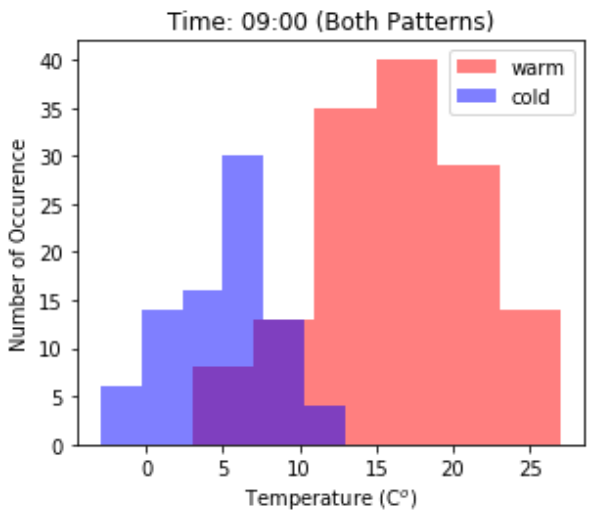

Time: 04:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

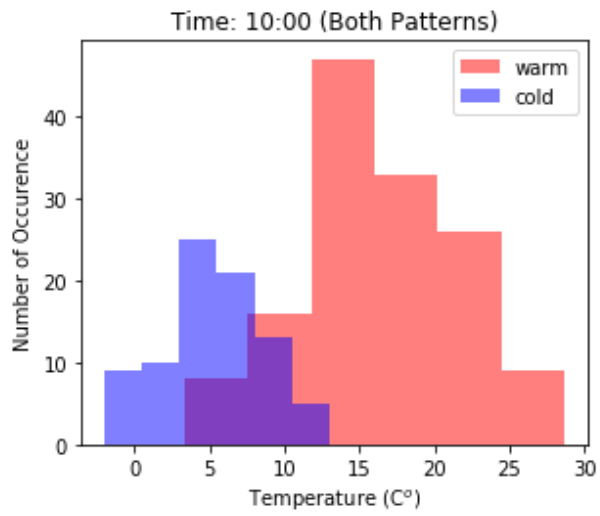

Time: 05:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

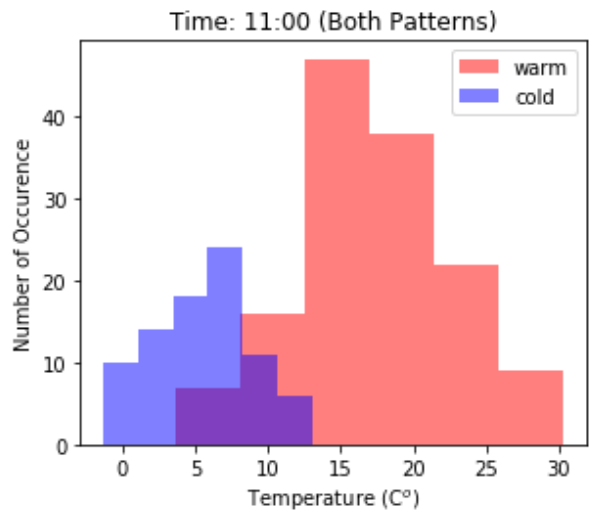

Time: 06:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

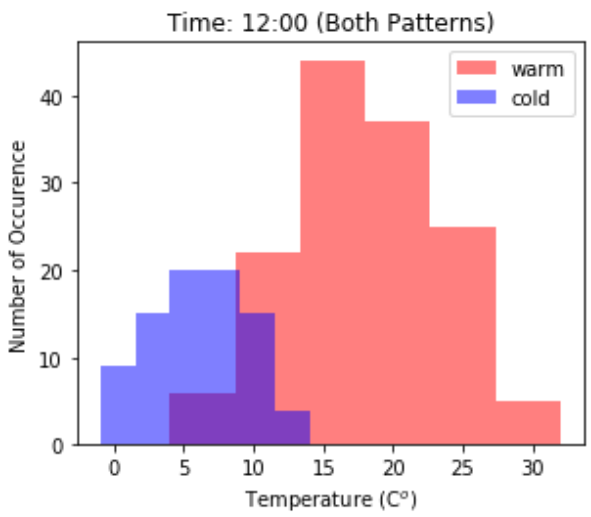

Time: 07:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

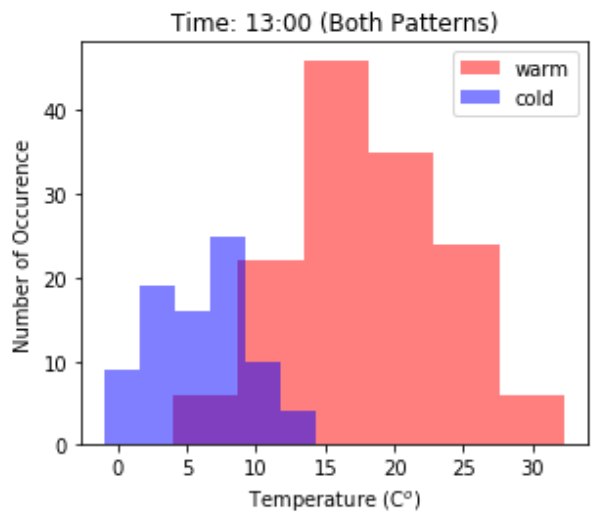

Time: 08:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

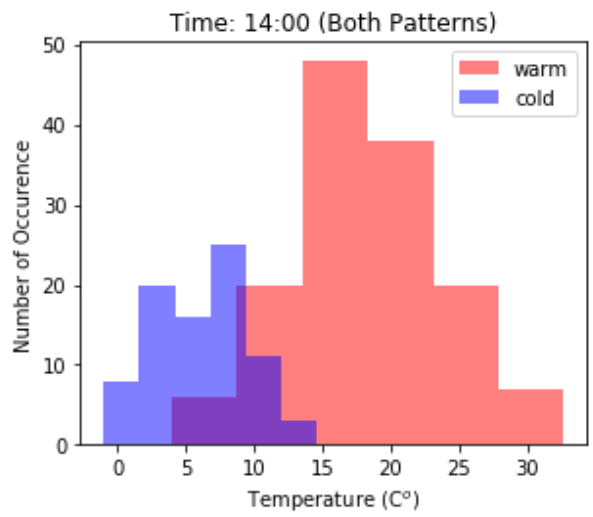

Time: 09:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

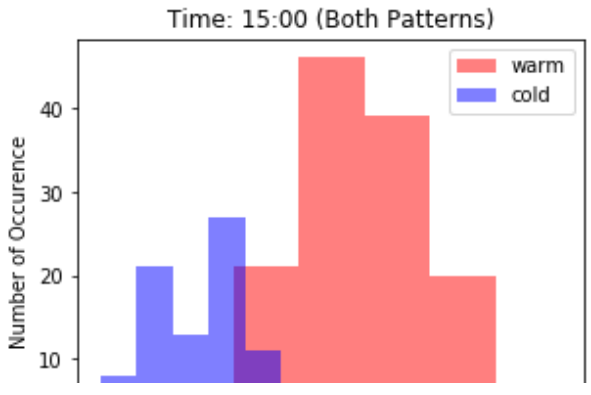

Time: 10:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

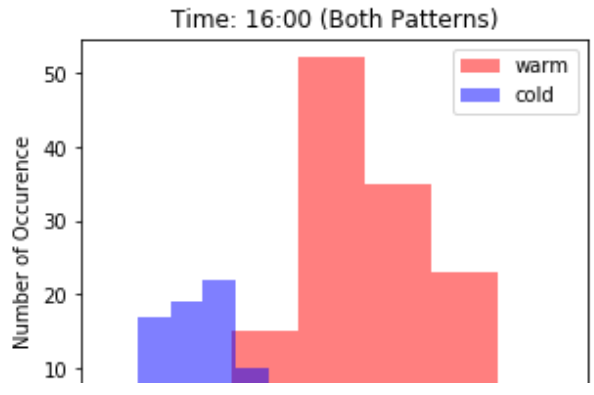

Time: 11:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

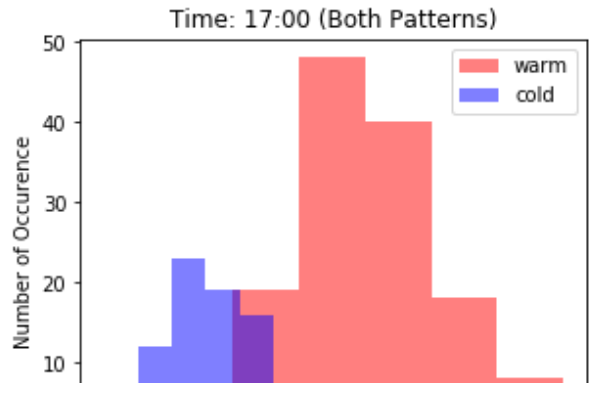

Time: 12:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)

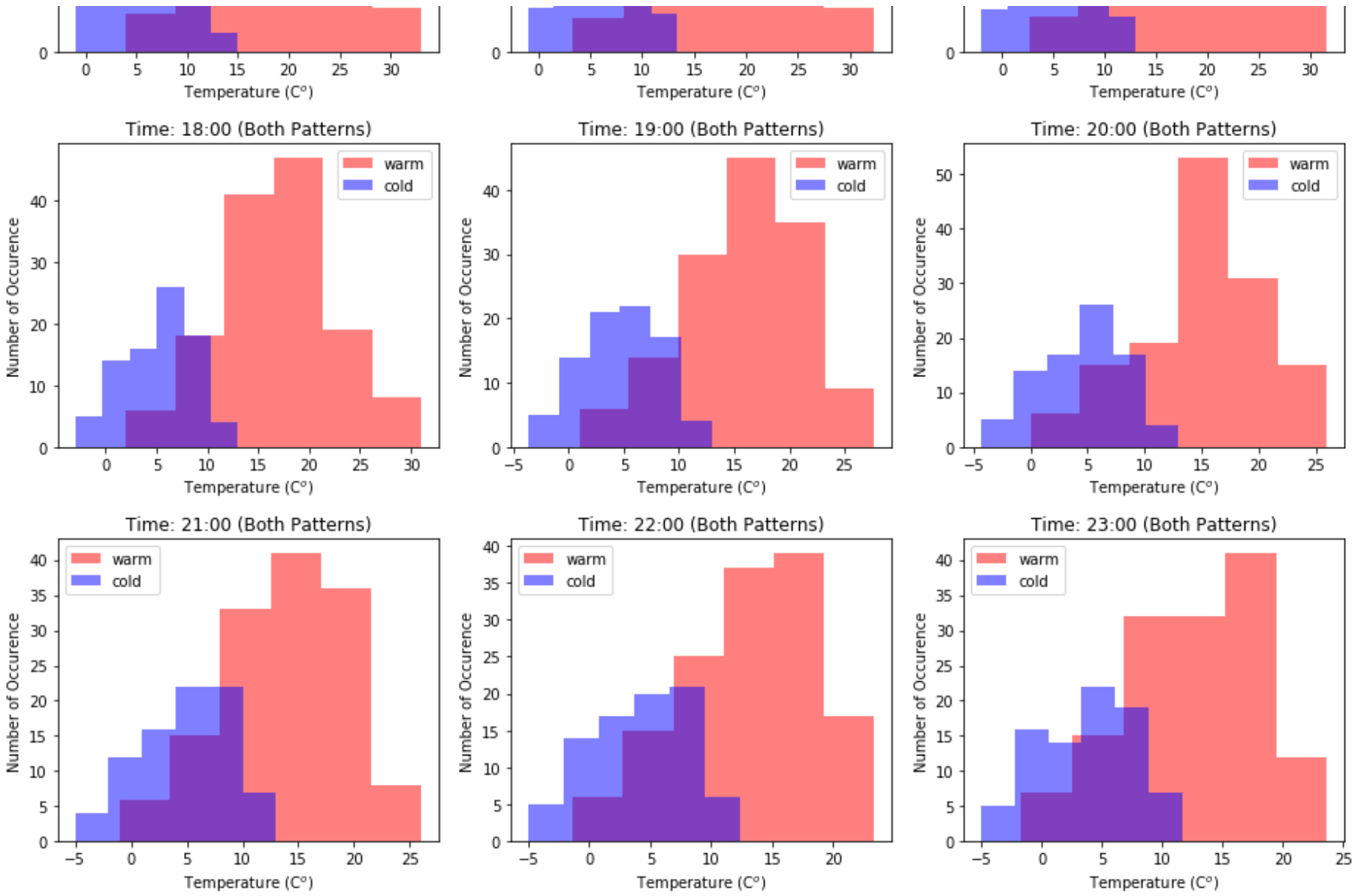

Time: 13:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)


Time: 14:00 (Both Patterns)
warm
cold
Number of Occurence
Temperature (C°)


Time: 15:00 (Both Patterns)
warm
cold
Number of Occurence


Time: 16:00 (Both Patterns)
warm
cold
Number of Occurence


Time: 17:00 (Both Patterns)
warm
cold
Number of Occurence



(1) In PCA, we don't consider normalization since all the consumptions have the same unit, we care about the absolute change rather than relative change since DR targets the customer who has largest potential shiftable load instead of largest price-responsiveness rate. Also we have seperate data by hours, so the temp-consumption relationship at different hours has been treated seprately and equally, so the time effect won't be a concern to let us normalize the data.

In [5]:
```
df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('09:00:00')].transpose()
```

Out[5]:

	9	33	57	105	129	201	273	321	345	417	...	8409	8433	8505	
GMT	2013-01-01 09:00:00	2013-01-02 09:00:00	2013-01-03 09:00:00	2013-01-05 09:00:00	2013-01-06 09:00:00	2013-01-09 09:00:00	2013-01-12 09:00:00	2013-01-14 09:00:00	2013-01-15 09:00:00	2013-01-18 09:00:00	...	2013-12-17 09:00:00	2013-12-18 09:00:00	2013-12-21 09:00:00	2 1 0
D0000	0.178	0.092	0.484	0.095	0.087	0.114	0.064	0.069	0.169	0.101	...	0.354	0.094	0.193	0
D0001	0.463	0.469	0.604	0.43	0.441	0.416	0.465	0.551	0.335	0.383	...	0.436	0.375	0.504	0
D0002	0.482	0.495	0.463	0.42	0.415	1.417	1.122	1.105	0.383	1.757	...	1.068	1.315	0.545	0
D0003	0.153	0.152	0.451	0.155	0.329	0.154	0.147	0.147	0.145	0.142	...	0.114	0.28	0.286	0
D0004	0.231	0.309	0.342	0.759	0.278	0.054	0.282	0.146	0.054	0.425	...	0.107	0.252	0.992	0
D0005	0.136	0.367	0.267	0.294	0.261	0.288	0.285	0.274	0.281	0.432	...	0.335	0.327	0.292	0
D0006	0.096	0.244	0.701	0.065	0.096	0.683	0.237	1.004	0.214	0.406	...	0.999	0.466	0.096	0
D0007	0.177	0.191	0.425	0.324	0.275	0.32	0.299	0.206	0.244	0.321	...	0.75	0.229	0.346	0
D0008	0.168	0.33	0.25	0.121	0.148	0.225	0.229	0.145	0.292	0.425	...	0.296	0.427	0.154	0
D0009	0.108	0.203	0.184	0.233	0.876	0.193	0.947	0.107	0.105	0.087	...	0.173	0.161	0.348	0
D0010	0.018	0.027	0.022	0.029	0.023	0.024	0.023	0.018	0.031	0.025	...	0.031	0.017	0.028	0
D0011	0.354	0.475	0.088	0.245	0.381	0.354	0.102	0.4	0.118	0.329	...	0.504	0.257	0.452	0
D0012	0.311	0.197	0.129	0.17	0.223	0.148	0.2	0.186	0.23	0.215	...	0.159	0.164	0.132	0
D0013	0.132	0.136	0.134	0.281	0.489	0.109	0.357	0.109	0.128	0.379	...	0.556	0.104	0.158	0
D0014	0.697	2.076	1.152	0.503	0.392	1.179	0.55	1.069	1.949	1.578	...	1.486	1.335	0.586	0
D0015	0.186	0.195	0.205	0.416	0.928	0.206	0.098	0.29	0.853	0.44	...	0.99	1.176	0.408	1
D0016	0.071	0.221	0.067	0.315	0.225	0.169	0.063	0.071	0.109	0.079	...	0.078	0.1	0.163	1
D0017	0.103	0.18	0.297	1.14	1.791	0.174	1.611	3.225	3.811	0.298	...	0.482	1.116	0.345	2
D0018	0.05	0.108	0.18	1.181	0.126	0.796	0.202	0.196	0.281	0.368	...	0.103	0.163	0.057	0
D0019	0.943	3.24	2.709	0.745	0.408	0.453	0.534	0.394	0.461	0.434	...	0.928	0.446	0.599	0
D0020	1.575	2.274	1.132	2.463	2.214	2.591	2.608	0.178	0.195	2.309	...	1.261	2.177	2.434	1
D0021	0.079	0	0.037	0.033	0.017	0.011	0.027	0	0.08	0.032	...	0	0	0	0
D0022	0.642	0.217	0.324	0.3	0.36	0.342	0.876	0.324	0.348	0.447	...	0.292	0.242	0.267	0
D0023	0.467	1.046	0.687	0.364	0.603	0.498	0.733	0.495	0.915	0.552	...	0.958	0.697	0.746	0
D0024	0.071	0.068	0.069	0.071	0.294	0.091	0.196	0.085	0.897	0.318	...	0.149	0.077	0.143	0
D0025	1.48	1.077	1.088	1.122	0.875	0.869	0.505	1.066	1.116	1.103	...	1.037	0.751	1.658	2
D0026	0.088	0.094	0.107	0.099	0.099	0.151	0.085	0.1	0.078	0.36	...	0.103	0.102	0.1	0
D0027	0.198	0.187	0.174	0.238	0.381	0.179	0.421	0.293	0.182	0.439	...	0.259	0.291	0.369	0
D0028	0.513	0.665	0.463	0.54	1.406	0.369	0.796	0.432	0.531	0.449	...	0.397	0.375	0.756	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
D0995	0.159	0.056	0.056	0.057	0.476	0.186	0.347	0.2	0.774	0.844	...	0.386	0.266	0.291	0
D0996	0.813	2.727	2.918	0.939	3.22	3.333	1.616	2.101	0.831	0.783	...	0.591	0.84	2.519	1
D0997	0.384	0.826	0.46	0.537	0.489	0.85	0.529	0.522	0.665	0.882	...	1.176	0.967	1.046	1
D0998	0.679	0.561	0.556	0.544	0.727	0.492	0.661	0.35	0.381	0.365	...	0.345	0.451	0.786	1
D0999	0.169	0.451	0.3	0.268	0.418	0.74	0.352	0.342	0.455	0.314	...	0.309	0.354	0.285	0
D1000	0.733	0.368	0.375	0.197	0.349	0.853	0.306	0.312	0.809	0.364	...	0.445	0.351	0.383	0
D1001	1.004	2.631	0.652	0.691	0.581	0.904	1.212	0.303	1.088	1.556	...	0.812	1.062	0.458	1
D1002	1.871	0.269	0.346	1.676	1.253	2.009	1.579	1.559	1.54	1.398	...	1.583	1.449	1.183	2
D1003	0.203	0.235	0.275	0.218	0.477	0.19	0.34	0.152	0.203	0.219	...	0.215	0.242	0.186	0
D1004	0.048	0.054	0.32	0.086	0.145	0.203	0.044	0.037	0.183	0.043	...	0.352	0.063	0.48	0
D1005	3.933	2.882	1.57	1.367	3.866	0.622	1.484	1.015	3.029	1.156	...	0.523	0.397	0.408	0
D1006	0.21	0.117	0.133	0.125	0.124	0.115	0.11	0.135	0.252	0.2	...	0.142	0.115	0.255	0
D1007	0.4	0.142	0.167	0.248	0.433	0.167	0.154	0.19	0.278	0.13	...	0.111	0.112	0.12	0
D1008	0.311	0.118	0.494	0.111	0.114	0.526	0.274	0.46	0.403	0.603	...	0.472	0.434	0.561	0
D1009	0.713	0.517	0.515	0.385	0.68	0.69	0.528	0.775	0.822	0.681	...	0.582	1.084	0.349	0
D1010	0.268	0.275	0.268	0.258	0.206	0.326	0.263	0.273	0.424	0.259	...	0.649	0.301	0.281	0
D1011	0.347	0.294	0.316	0.338	0.323	0.368	0.284	0.359	0.369	0.339	...	0.996	0.383	0.323	1
D1012	0.114	0.077	0.164	0.361	0.15	0.112	1.52	0.059	0.108	0.111	...	0.251	0.167	0.153	0

	9	33	57	105	129	201	273	321	345	417	...	8409	8433	8505	
D1013	1.217	1.255	0.941	1.212	0.499	0.869	1.974	1.719	2.522	4.279	...	1.545	1.661	2.47	1
D1014	0.34	0.629	0.835	0.53	0.299	1.411	0.729	0.902	1.408	1.403	...	1.634	1.672	0.81	0
D1015	0.062	0.043	0.07	0.078	0.081	0.062	0.207	0.132	0.137	1.331	...	0.083	0.088	0.08	0
D1016	0.106	0.255	0.252	0.197	0.231	0.01	0.808	0.01	0.911	0.351	...	0.035	0.034	0.136	0
D1017	0.245	0.207	0.207	0.449	0.557	0.192	0.5	0.236	0.245	0.381	...	0.971	0.326	0.33	0
D1018	0.251	0.397	0.086	0.087	0.093	0.102	0.08	0.089	0.09	0.098	...	0.132	0.126	0.201	0
D1019	0.165	0.032	0.036	0.376	0.393	0.029	0.55	0.19	0.322	0.18	...	0.249	0.1	0.352	0
D1020	0.214	0.11	0.259	0.238	0.243	0.243	0.256	0.153	0.238	0.226	...	0.114	0.296	0.213	0
D1021	0.348	0.118	0.09	0.369	0.566	0.088	0.234	0.085	0.088	0.069	...	0.109	0.109	0.674	0
D1022	0.065	0.06	0.068	0.06	0.06	0.058	0.057	0.143	0.143	0.06	...	0.068	0.066	0.068	0
D1023	0.017	0.016	0.046	0.046	0.026	0.013	0.016	0.052	0.037	0.023	...	0.024	0.045	0.047	0
D1024	0.172	0.441	0.121	0.12	0.178	0.074	0.173	0.266	0.321	0.537	...	0.179	0.178	0.122	0

1026 rows × 83 columns

In [7]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Test with 9:00 am TOU data
x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('09:00:00')]
x.set_index('GMT', inplace = True)
x = x.transpose().dropna() # data for the specific hour, and filtering out null value
x = x.values
x = StandardScaler().fit_transform(x)
pca = PCA()# keep all columns to cover 100% variance
principleComponents = pca.fit_transform(x)
principleDf = pd.DataFrame(data = principleComponents)
principleDf
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	73	
0	-4.534874	0.453149	-0.791187	0.190559	-0.019240	-0.216637	0.002832	0.033447	-0.200984	-0.116277	...	0.047344	-0.0354
1	-2.259145	-0.894439	0.065706	-1.158057	-0.169163	-0.384709	0.269293	0.295374	0.321222	0.080441	...	-0.084316	-0.1865
2	6.623904	0.566495	2.227523	0.283639	1.263913	0.823391	-1.208211	1.297698	-0.745383	-0.250109	...	0.207286	-0.3131
3	-2.740304	1.903443	0.820230	0.305847	0.466140	-0.485148	-0.582654	0.260836	-0.090145	-0.733082	...	0.106626	-0.2120
4	-3.662682	-0.357209	1.408946	-0.311664	0.480621	-0.457705	-0.179151	0.287924	0.007132	0.492172	...	0.301034	-0.2724
5	-2.355763	-0.460153	-0.311662	0.083363	0.184648	-0.127965	0.004612	-0.082631	0.190776	-0.060942	...	-0.029751	-0.0476
6	-1.970082	0.457461	-1.200723	-0.310279	0.513020	-0.830899	0.089446	0.092306	-0.365178	-0.462662	...	-0.015333	-0.5668
7	-0.681808	0.273903	-0.735341	-0.149064	0.061141	-0.226158	-0.949310	-0.030437	-0.046488	-0.241626	...	-0.056229	0.2180
8	-3.266754	0.000133	-0.386045	-0.399162	0.111960	0.136354	-0.249222	-0.018619	-0.546985	0.047331	...	-0.105117	0.1406
9	0.400567	0.209370	1.729419	2.746633	-0.107041	0.326880	-2.408428	0.175536	1.585223	1.549523	...	-0.598657	-0.9729
10	-7.042212	0.513386	-0.329802	-0.132723	0.213440	0.120629	-0.109313	-0.021479	0.020474	0.067812	...	0.007259	-0.0120
11	0.233345	0.855499	-0.241367	0.765003	-0.745088	1.305379	-0.492979	-0.151881	-1.100386	-0.359916	...	-0.391132	-0.2397
12	-4.007395	0.560723	-0.013456	0.084144	0.103276	0.401981	-0.276543	0.282432	-0.319985	0.222099	...	-0.161069	-0.1913
13	-2.853565	-1.664745	0.897125	0.519937	0.507544	0.861155	0.032013	-0.506876	-0.194053	0.031603	...	0.300615	0.1573
14	14.498900	0.299989	-4.448355	0.502973	-1.332026	-0.198652	0.147603	-0.396975	-0.126261	-2.330912	...	0.951352	-0.2989
15	2.167650	-4.180741	-1.749307	-0.316495	-0.322945	-0.869980	-1.531033	0.524028	-0.056853	-0.615293	...	-0.863072	0.1543
16	-3.728301	0.842762	1.856092	0.690502	0.491812	-0.314615	-1.004052	-0.231904	0.097897	0.450348	...	0.202260	0.0645
17	4.102987	-2.643518	2.886709	-2.894087	1.265600	1.237621	-1.494439	-1.511791	0.528030	-0.166749	...	0.218258	-0.0171
18	-3.537458	0.893724	-0.240811	0.052045	-0.686409	0.384680	0.823733	0.132967	0.360862	0.264065	...	-0.012396	-0.6452
19	2.104148	0.195389	0.866380	-1.255357	-2.038346	-2.907979	2.075851	0.933191	-2.241491	1.323905	...	-0.140413	-0.3340
20	18.303541	1.400164	1.849293	-1.700998	-3.301577	-1.747596	-2.234843	3.230908	1.833557	-0.673990	...	-0.811869	0.4601
21	-7.036772	0.820405	-0.274520	-0.138987	0.191607	0.098103	-0.153313	-0.003722	0.027258	0.036139	...	0.030300	0.0650
22	-1.505614	-0.739888	-0.101177	-0.872268	0.550156	-0.098609	0.257770	-0.034515	-0.602426	0.407121	...	0.162990	0.2572
23	3.064493	-2.394037	-0.082017	-0.240338	-0.637217	-0.119054	-1.175948	-0.298469	-0.541840	0.238749	...	-0.668478	0.5236
24	-4.936588	0.606600	0.143683	-0.319888	-0.021122	0.405841	-0.322089	-0.474045	0.344248	0.104470	...	-0.113667	0.0761
25	10.108856	-2.332781	1.813273	0.670830	-0.616196	0.537470	0.732115	0.480197	0.005433	0.757899	...	-0.507717	0.0082
26	-5.530083	0.445816	-0.194528	-0.177582	0.144911	0.115361	-0.177564	-0.083938	-0.089109	-0.184788	...	0.054479	0.1161
27	-3.317530	-0.038190	0.537351	0.077267	0.076375	0.247581	-0.191911	-0.206590	0.134168	-0.025325	...	-0.168241	0.0978
28	0.370119	-0.531714	1.452065	-0.382532	-0.532530	-0.521505	0.405743	-0.069571	0.265831	-0.094091	...	-0.167408	0.1987
29	25.094028	10.211609	1.506032	-2.422289	-0.762213	-2.541431	1.238837	0.605744	-3.365384	2.014202	...	-0.605630	-0.3447
...	...	...	...	...	...	...	...	...	...	...	...	...	...
928	13.153705	-6.122491	-0.936893	-0.067435	-0.633626	0.444703	-1.677814	0.161935	-1.490300	-0.386026	...	0.698630	-0.4010
929	-2.935874	0.929708	-0.035909	-0.426892	0.195084	0.755464	-0.311774	0.429462	0.713920	-0.006891	...	-0.253823	0.3901
930	14.075726	-3.518164	1.727959	-3.201588	0.034378	-3.397127	2.803022	-0.160513	-1.695230	2.844778	...	-0.289703	0.7024
931	4.000995	-1.277612	0.686759	-0.758089	0.612863	-0.422578	-0.604598	1.150344	-0.244890	-0.563993	...	0.265708	0.2771
932	0.990693	-0.609995	1.340742	0.791783	-0.574173	-0.493437	-0.166120	0.325129	0.104913	-0.119455	...	-0.373864	-0.0610
933	-2.485661	0.454496	-0.349028	-0.349391	0.310378	-0.243987	0.773327	-0.313985	-0.211420	0.168497	...	-0.300455	-0.0044
934	-0.658207	-0.871539	-0.448679	-0.330381	-0.316628	-0.433882	0.220438	0.200953	0.019014	0.080577	...	-0.228598	0.4271
935	8.663549	0.613845	0.162767	-0.009711	-0.905966	0.181044	-0.423802	0.474654	0.348862	0.945534	...	-0.764420	-0.2392
936	13.605658	-2.656613	0.092270	-0.501913	2.020198	-0.034162	0.872248	1.242708	1.222496	2.303653	...	-0.944851	-0.5468
937	-2.568863	-0.541209	1.502244	0.326071	0.042381	-0.291426	-0.925268	-0.655208	0.043763	-1.127296	...	0.013813	-0.1180
938	-4.380465	-0.369758	0.206026	-0.143376	0.603609	-0.009522	-0.220267	0.342438	-0.004381	0.055447	...	0.099183	-0.4478
939	9.738359	6.188866	1.601376	-2.638353	-5.047227	-2.608289	2.424325	-0.176007	0.858945	2.784427	...	-0.695127	0.7960
940	-4.344584	0.826550	-0.204983	0.202969	0.327891	-0.015496	-0.095289	0.058839	0.241027	0.348843	...	0.182468	0.2811
941	-3.352356	0.557025	0.554580	0.762879	-0.097865	-0.517873	0.526028	0.006917	0.223316	0.021214	...	-0.350777	0.0157
942	0.833003	-0.867532	0.201552	0.921370	1.403132	0.170312	-0.170834	-0.156300	0.415621	-1.297357	...	0.300371	-0.1072
943	3.459084	-0.223873	-0.722817	0.214997	0.664079	0.480794	0.778462	0.388023	-0.509747	0.085176	...	0.829120	0.5830
944	0.564098	-1.554118	-1.787215	1.933966	-0.155582	0.420222	1.638490	0.065721	1.061218	-2.927511	...	-0.051116	0.4370
945	0.468435	-1.730717	-0.236668	0.139935	0.304885	-0.202096	-0.610773	-0.343775	-0.064980	-0.125990	...	0.380727	-0.2244
946	-2.229655	0.545837	0.165733	0.765799	0.827582	0.113421	-0.567176	-0.342415	0.527412	1.024382	...	-0.066970	0.2037

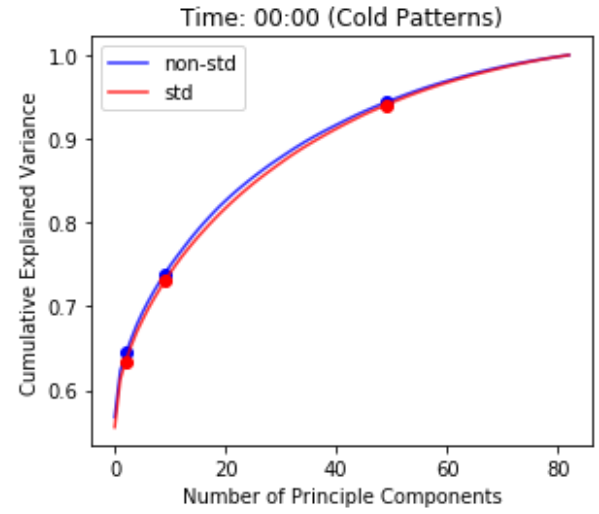
	0	1	2	3	4	5	6	7	8	9	...	73	
947	8.367828	-3.993003	-3.357941	-0.541480	0.151241	0.115484	1.259508	-1.022628	-0.588344	0.182489	...	0.285962	0.2793
948	-5.187998	0.328570	0.438370	0.411986	-0.010860	1.105498	0.210231	0.110966	0.216752	0.342440	...	0.178846	0.0316
949	-4.065603	1.689053	-0.055707	-0.549430	-0.296125	1.245819	0.154214	-0.434853	0.145266	-0.095215	...	-0.131266	0.0171
950	-2.582443	-0.177968	0.177919	0.140931	-0.058079	-0.344395	-0.031018	-0.151867	0.306144	-0.205229	...	0.175638	0.1414
951	-5.172672	0.539687	-0.109800	-0.283150	0.153736	-0.099180	-0.163905	0.187286	-0.138322	0.323977	...	-0.066971	0.2378
952	-3.050225	-0.178013	-0.037244	0.280033	-0.114016	0.596511	-0.522600	-0.117968	0.252858	0.176127	...	-0.050604	0.1538
953	-4.263098	0.100796	0.041619	-0.156598	0.207957	-0.242796	0.052498	-0.032927	-0.110964	-0.155199	...	-0.013113	0.0576
954	-3.871392	-0.095418	0.640463	0.023969	0.143116	0.136389	0.047174	-0.059563	0.465892	0.507855	...	-0.164085	-0.6852
955	-5.825854	0.555410	-0.352943	-0.095357	0.395843	0.064172	-0.107393	-0.112629	-0.060299	0.090714	...	-0.024258	-0.1257
956	-6.768653	0.552730	-0.344165	-0.041109	0.243888	0.128197	-0.105119	-0.015108	-0.046033	0.031498	...	0.020951	-0.0085
957	-4.194996	0.735968	-0.315896	-0.365311	0.012948	0.179958	-0.228105	-0.329538	-0.518435	0.228259	...	-0.295101	-0.1673

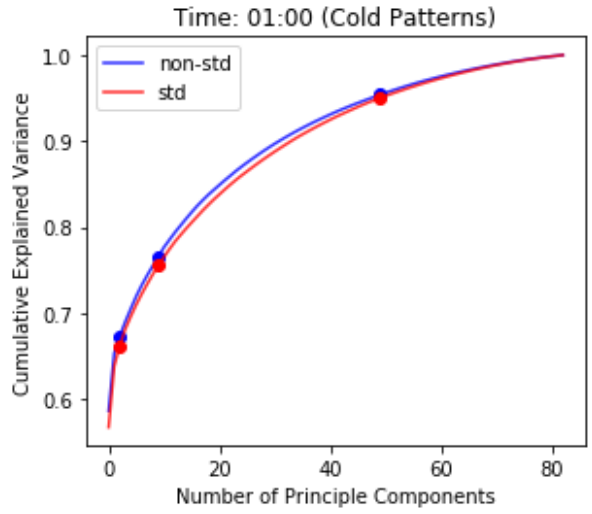
958 rows × 83 columns

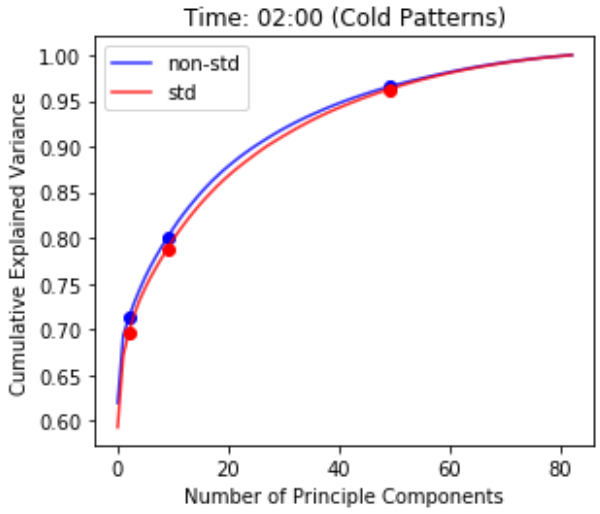
In [8]:
```
sum(x[:,0])
```

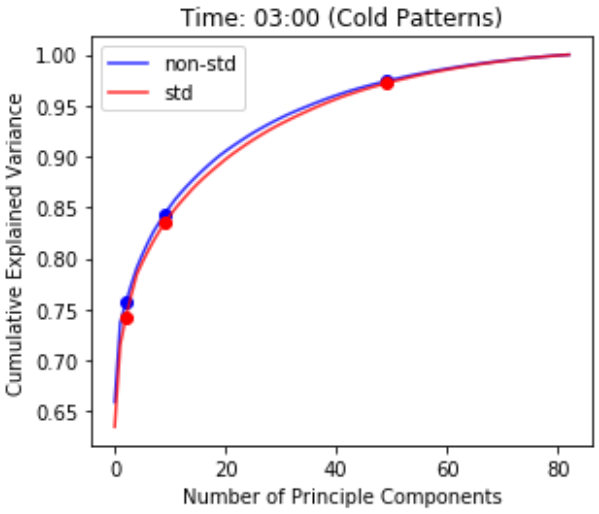
Out[8]:
```
4.796163466380676e-14
```

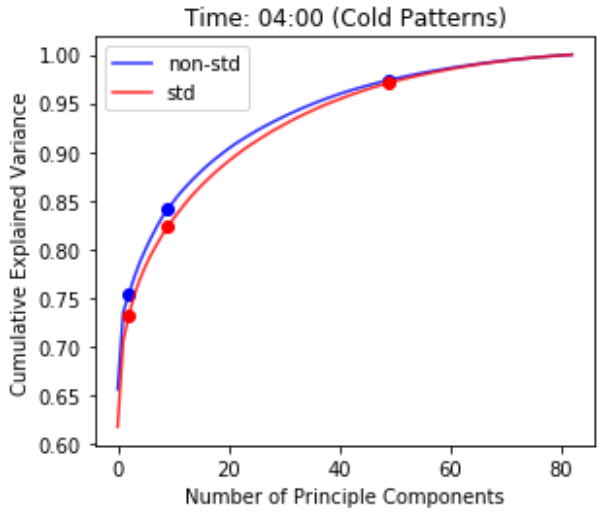
In [6]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.plot(np.cumsum(pca_nstd.explained_variance_ratio_), c = 'blue', label = 'non-std', alpha = 0.8)
        ax.plot([2, 9, 49], [np.cumsum(pca_nstd.explained_variance_ratio_)[2], np.cumsum(pca_nstd.explained_v
ariance_ratio_)[9], np.cumsum(pca_nstd.explained_variance_ratio_)[49]], 'bo')
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.plot(np.cumsum(pca.explained_variance_ratio_), c = 'red', label = 'std', alpha = 0.8)
        ax.plot([2, 9, 49], [np.cumsum(pca.explained_variance_ratio_)[2], np.cumsum(pca.explained_variance_ra
tio_)[9], np.cumsum(pca.explained_variance_ratio_)[49]], 'ro')
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Number of Principle Components')
        ax.set_ylabel('Cumulative Explained Variance')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.plot(np.cumsum(pca_nstd.explained_variance_ratio_), c = 'blue', label = 'non-std', alpha = 0.8)
        ax.plot([2, 9, 49], [np.cumsum(pca_nstd.explained_variance_ratio_)[2], np.cumsum(pca_nstd.explained_v
ariance_ratio_)[9], np.cumsum(pca_nstd.explained_variance_ratio_)[49]], 'bo')
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.plot(np.cumsum(pca.explained_variance_ratio_), c = 'red', label = 'std', alpha = 0.8)
        ax.plot([2, 9, 49], [np.cumsum(pca.explained_variance_ratio_)[2], np.cumsum(pca.explained_variance_ra
tio_)[9], np.cumsum(pca.explained_variance_ratio_)[49]], 'ro')
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Number of Principle Components')
        ax.set_ylabel('Cumulative Explained Variance')
plt.tight_layout()
```

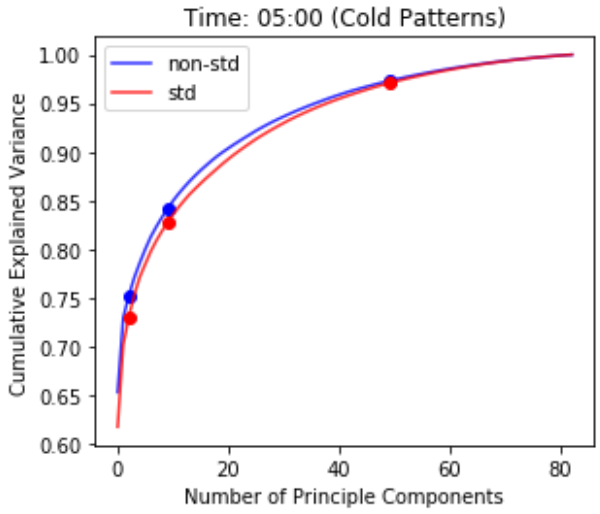

Time: 00:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

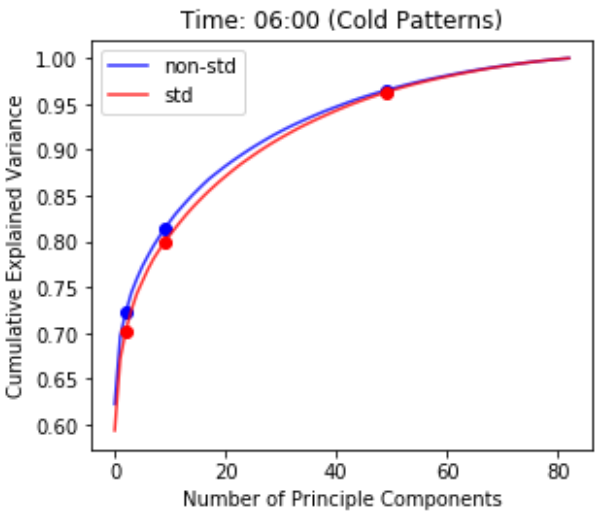

Time: 01:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

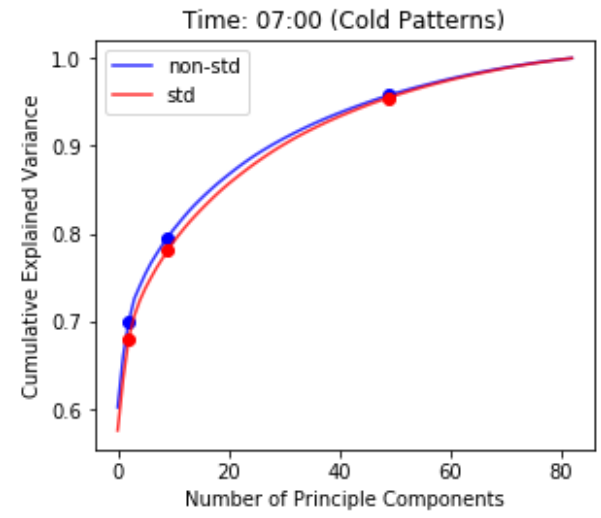

Time: 02:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

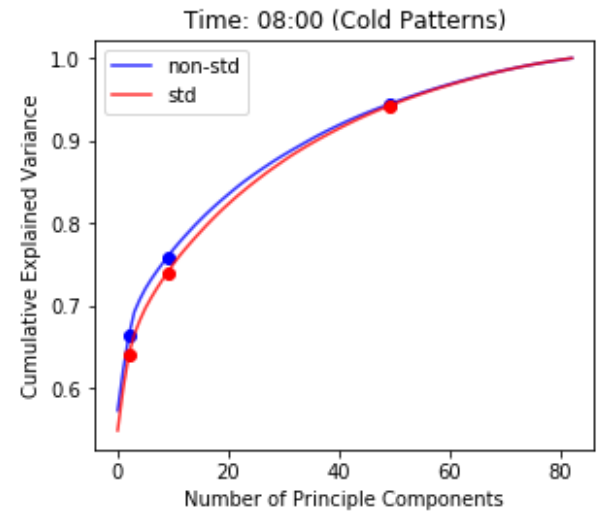

Time: 03:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

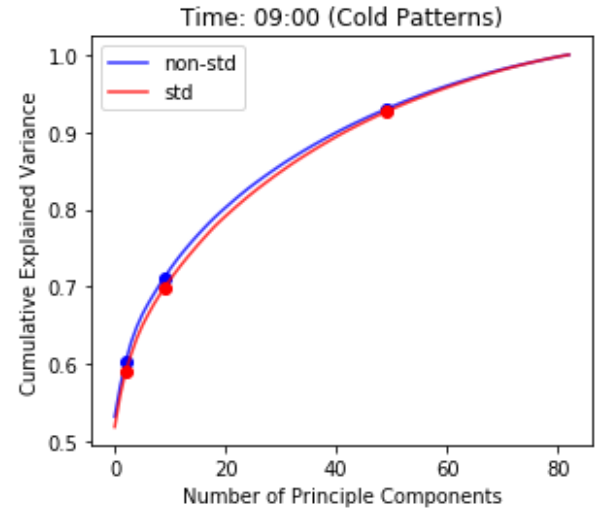

Time: 04:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

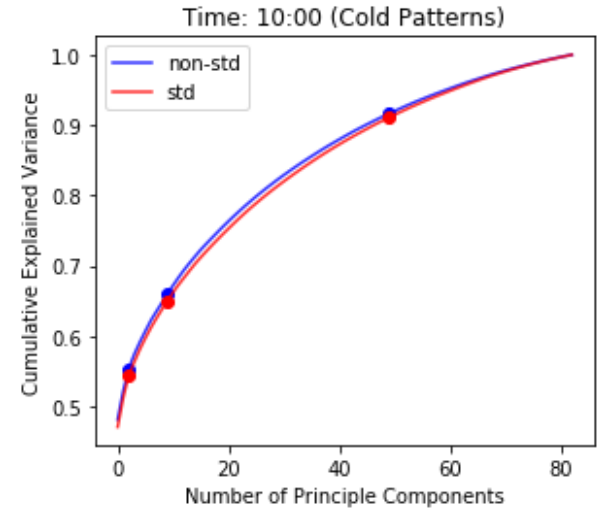

Time: 05:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

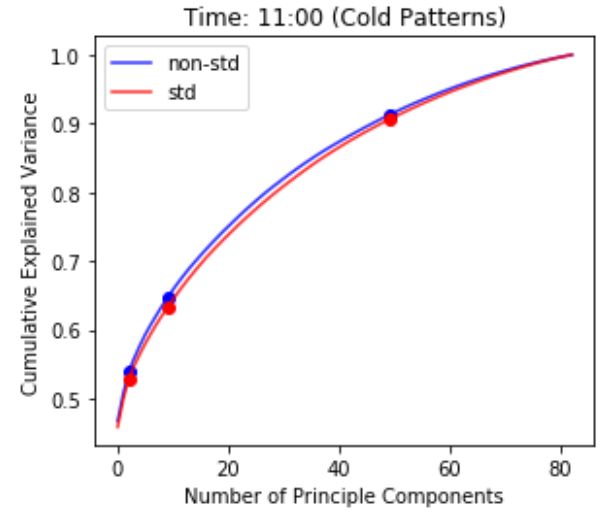

Time: 06:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

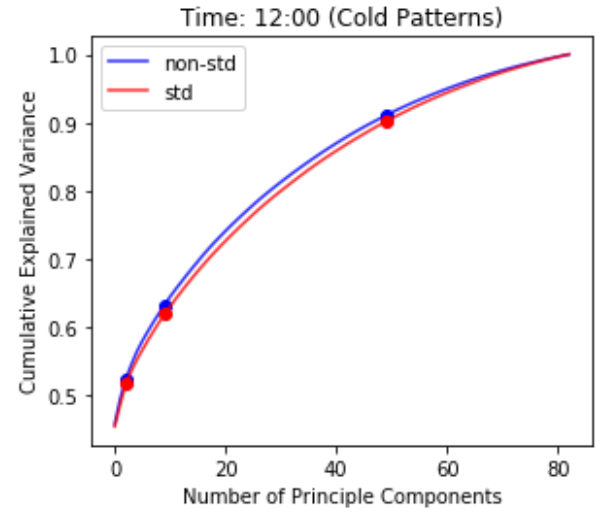

Time: 07:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

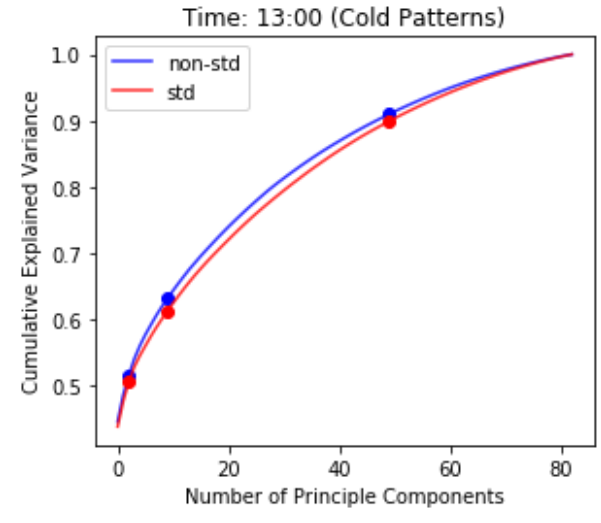

Time: 08:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

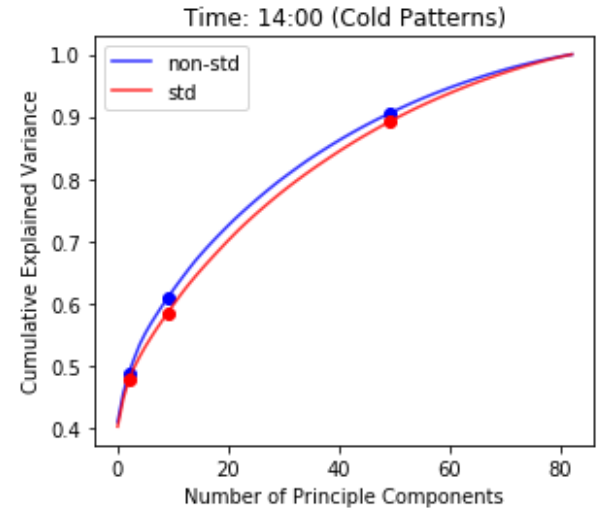

Time: 09:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components


Time: 10:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components


Time: 11:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components


Time: 12:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

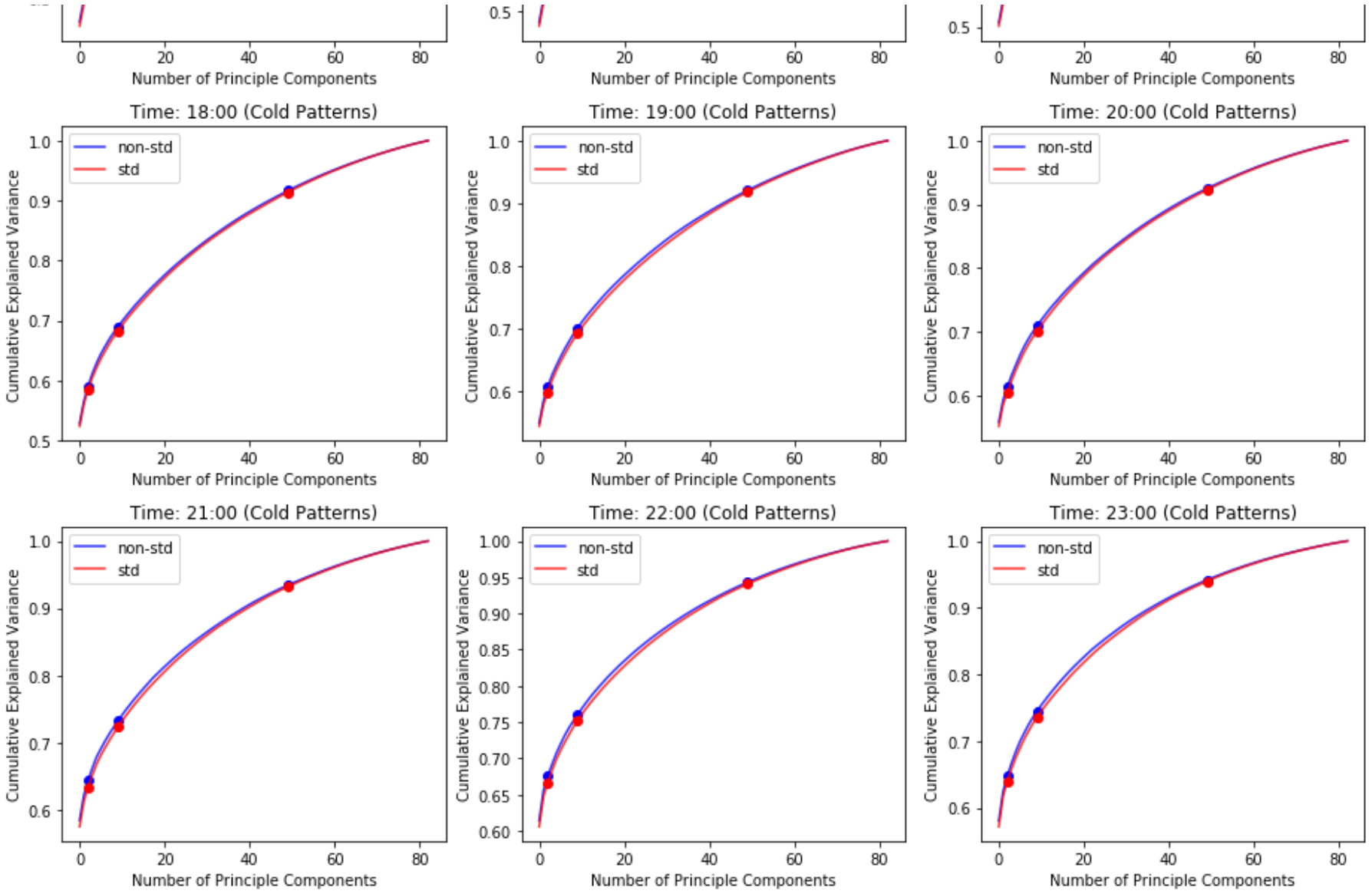

Time: 13:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components


Time: 14:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

Time: 15:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

Time: 16:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components

Time: 17:00 (Cold Patterns)
non-std
std
Cumulative Explained Variance
Number of Principle Components



The above pictures show that using the first 3 PCs can always explain from 48% to 72% amount of variance.

Next we will give the meaning of the first 3 PCs:

PC1 & PC2: temperature related (no matter of standarization, PC1 and PC2 are somehow correlated)

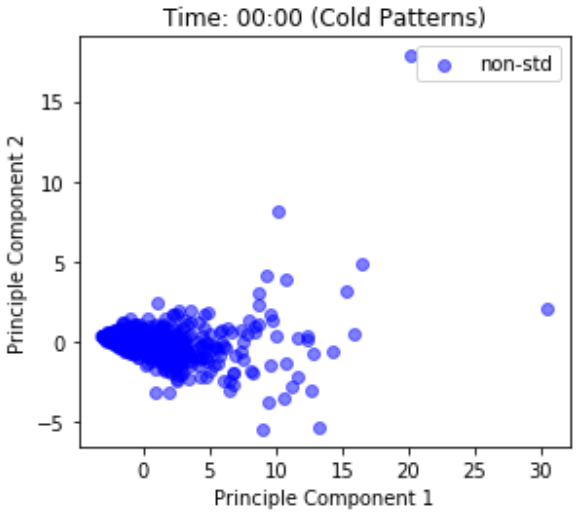
PC3: related to days of week, since the coeffients of Friday and Saturday are skewied to positive area, if there's not correlation, it should always be equally distributed around 0

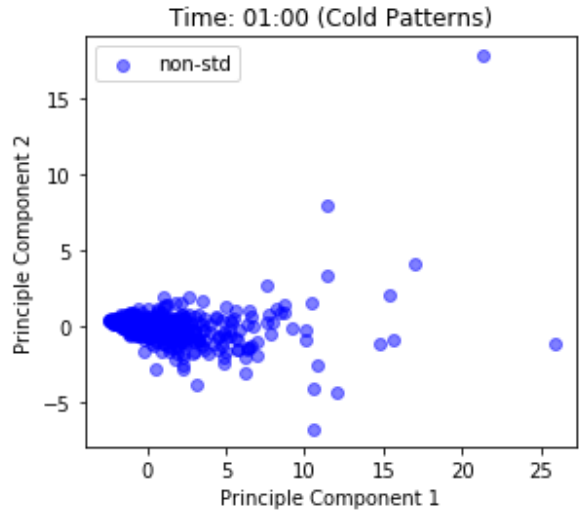
Using the 3 components to do k-mean clustering and we can get several groups

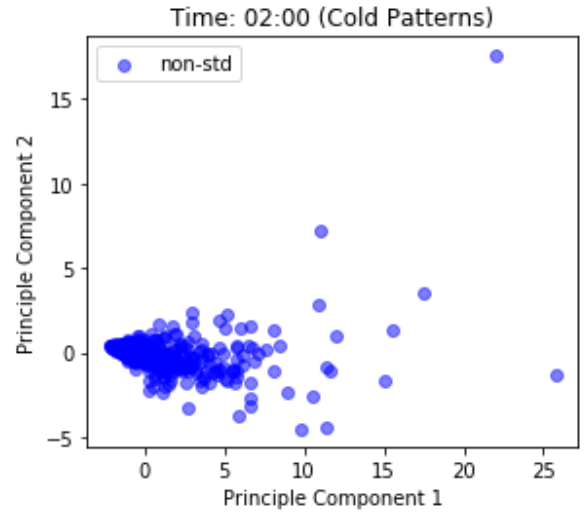
Coloring them in a 3D space (or if the first two PCs leads to some confustion, maybe do 2D with 3 differnt projections)

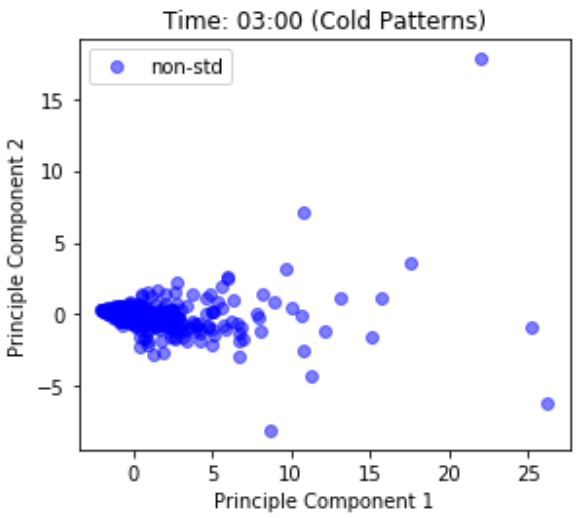
Then we could try to color them in a 2D slice surphase (temp vs day of week, temp)

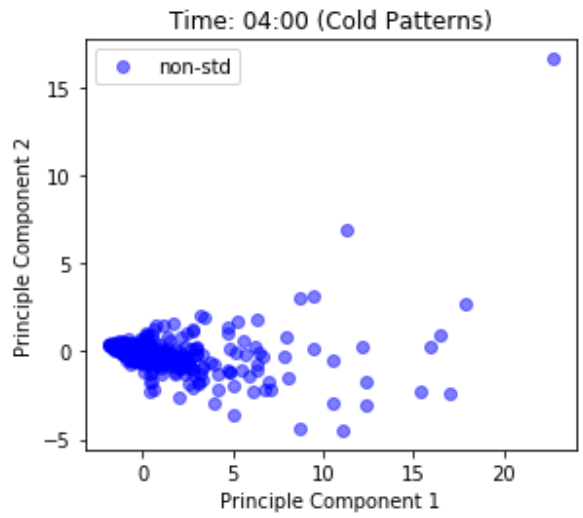
In [250]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[0], principleDf_nstd[1], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[0], principleDf[1], c = 'red', label = 'std', alpha = 0.1)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 2')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[0], principleDf_nstd[1], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[0], principleDf[1], c = 'red', label = 'std', alpha = 0.1)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 2')
plt.tight_layout()
```

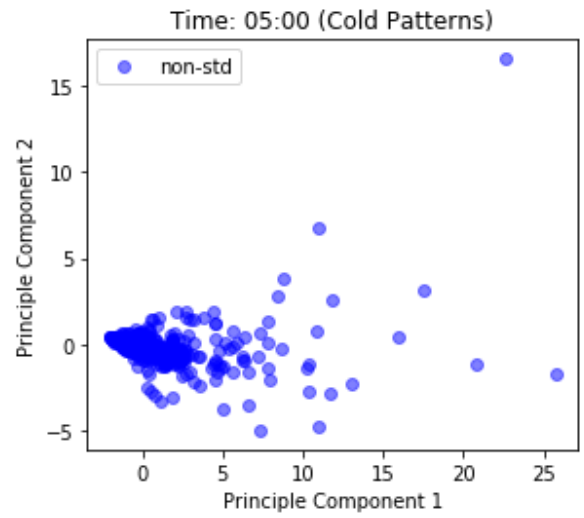

Time: 00:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

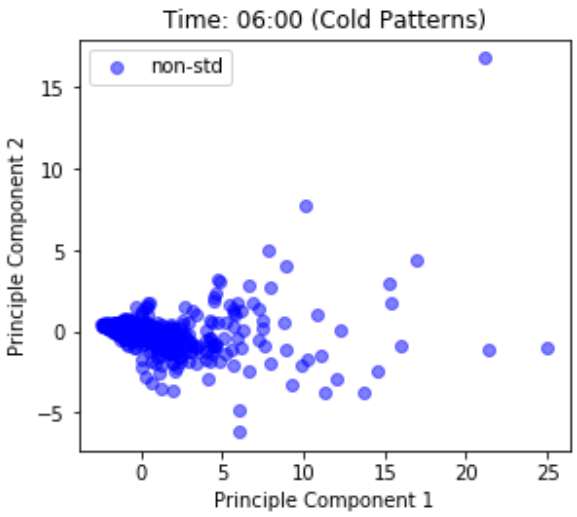

Time: 01:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

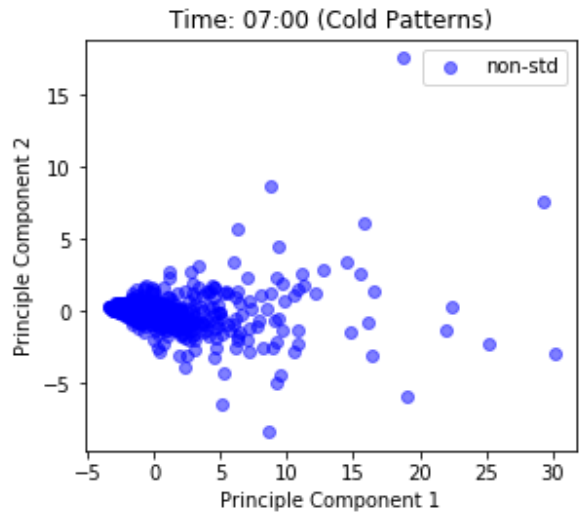

Time: 02:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

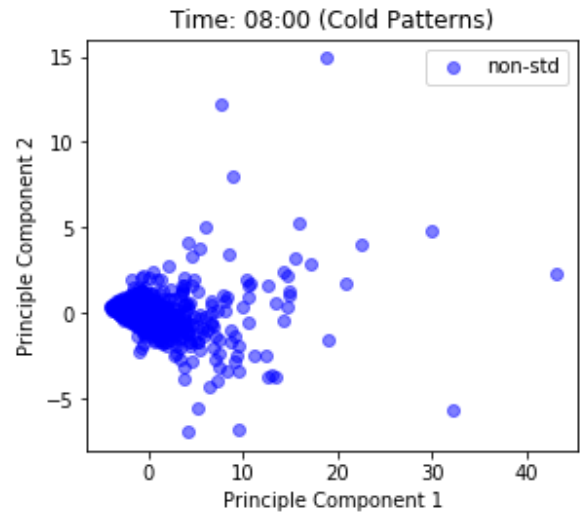

Time: 03:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

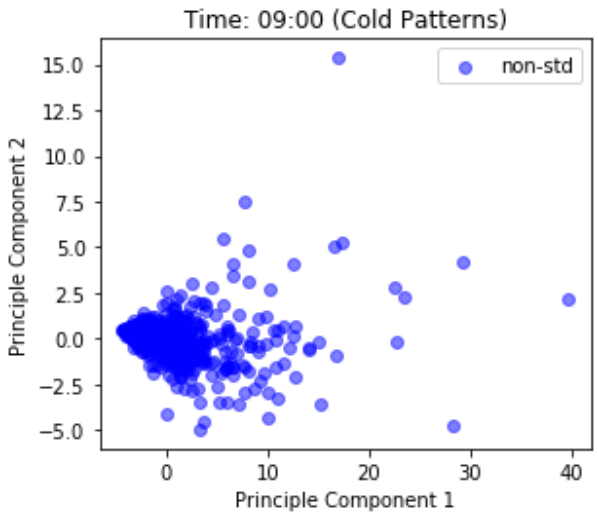

Time: 04:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

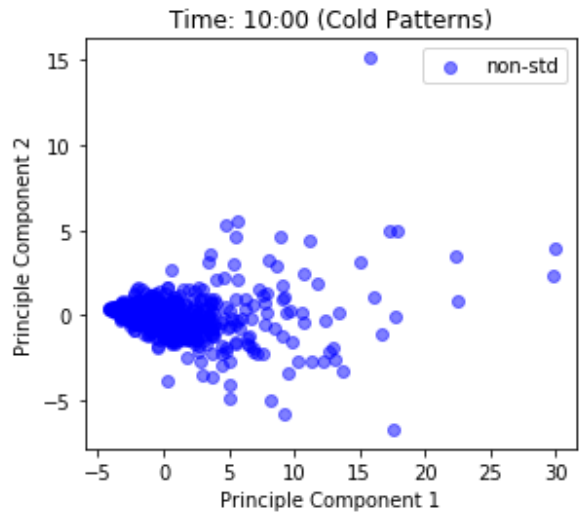

Time: 05:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

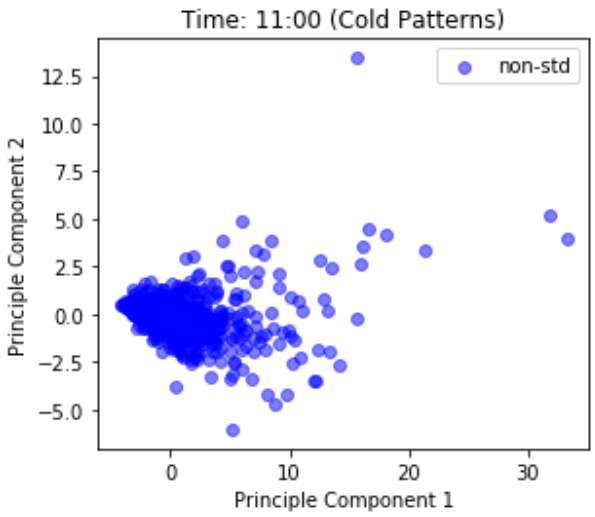

Time: 06:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

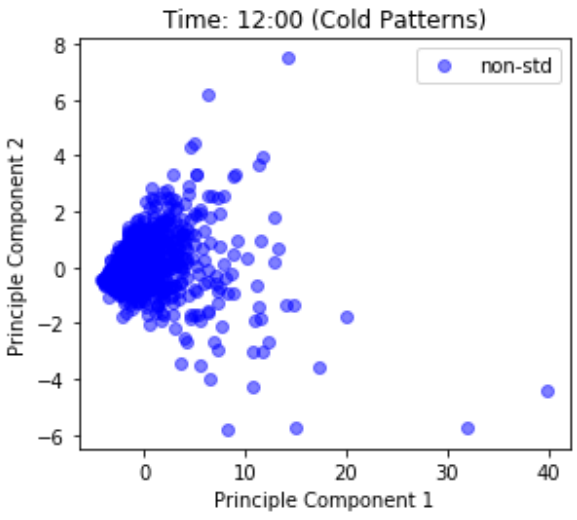

Time: 07:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

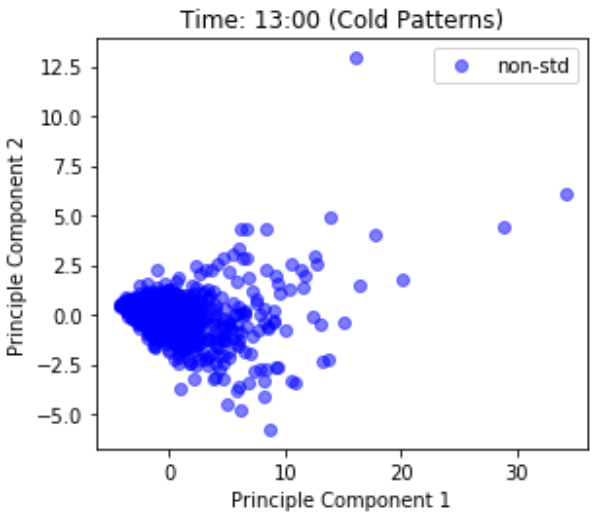

Time: 08:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

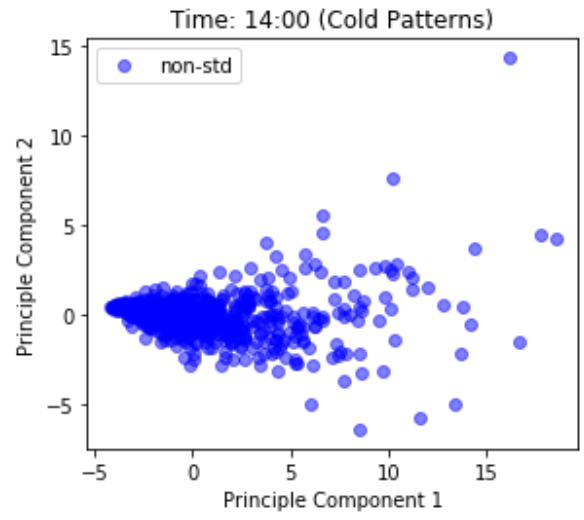

Time: 09:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

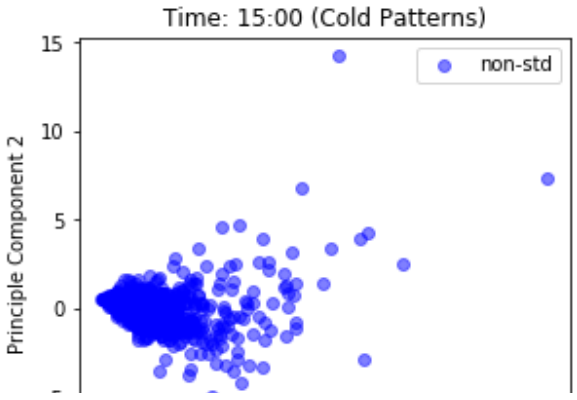

Time: 10:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

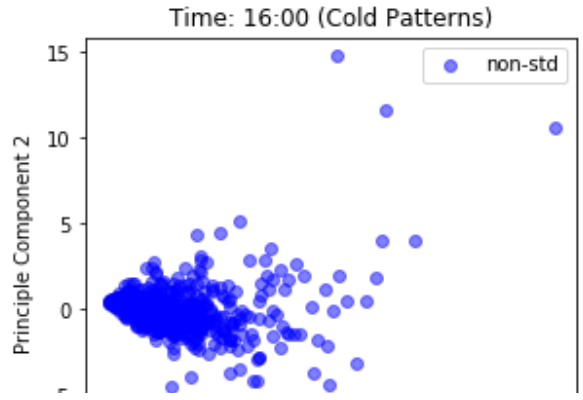

Time: 11:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

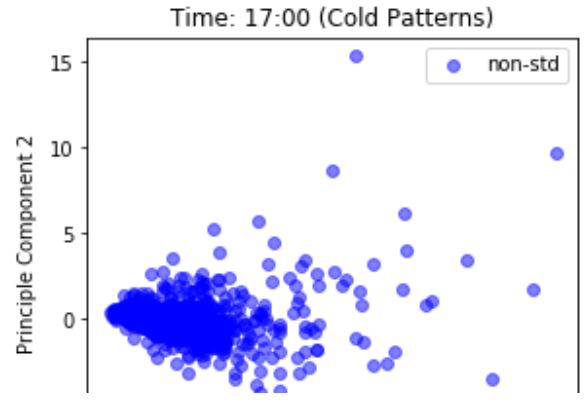

Time: 12:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

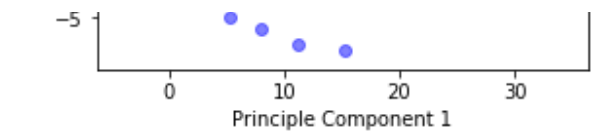

Time: 13:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

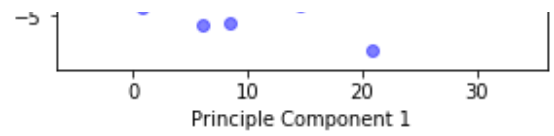

Time: 14:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

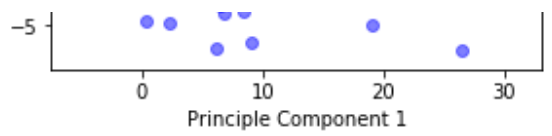

Time: 15:00 (Cold Patterns)
non-std
Principle Component 2

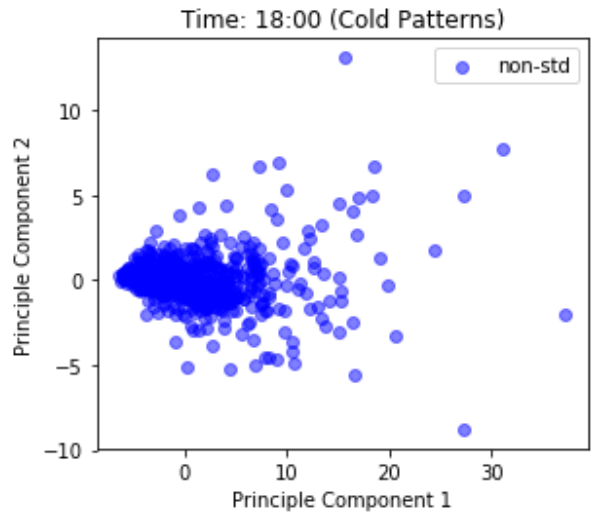

Time: 16:00 (Cold Patterns)
non-std
Principle Component 2

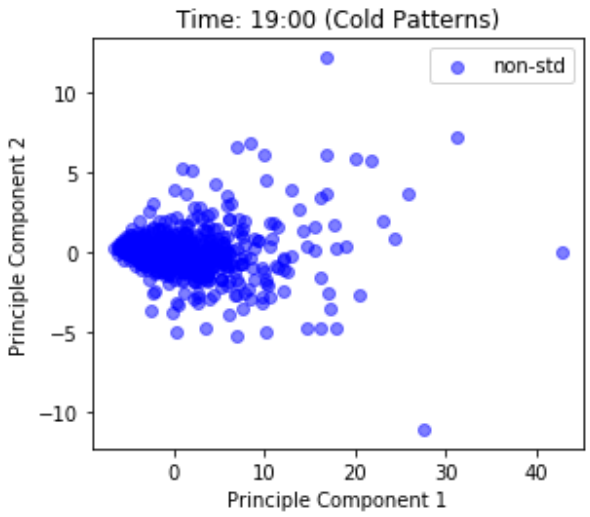

Time: 17:00 (Cold Patterns)
non-std
Principle Component 2

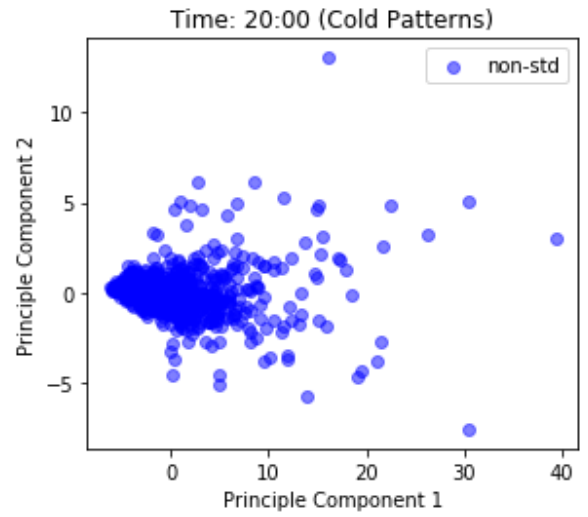

Principle Component 1

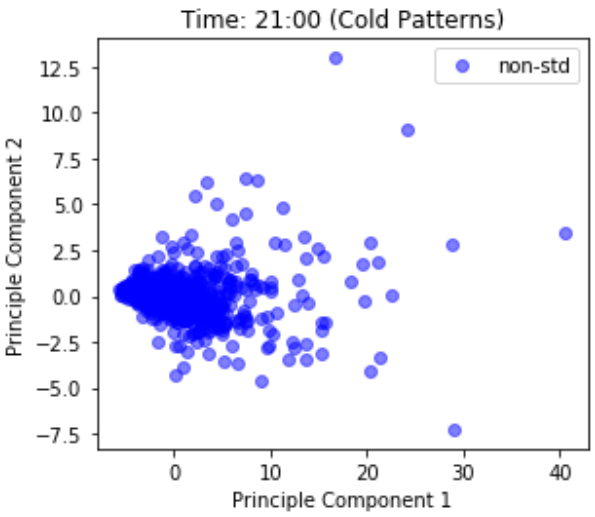

Principle Component 1

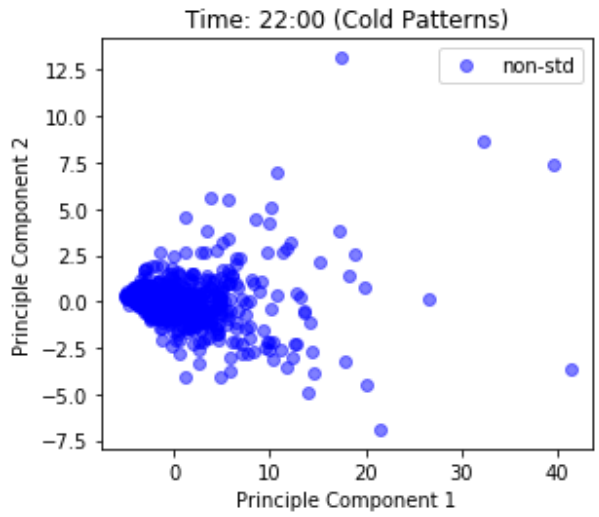

Principle Component 1

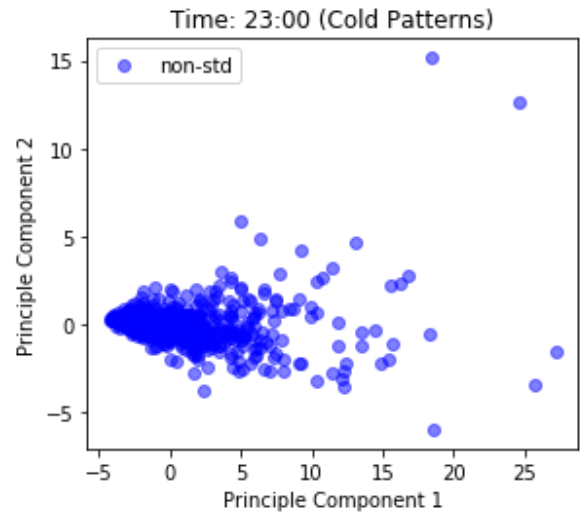

Time: 18:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1


Time: 19:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

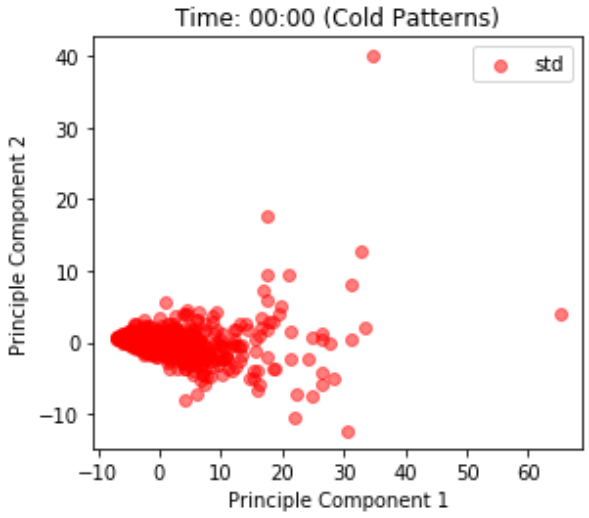

Time: 20:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

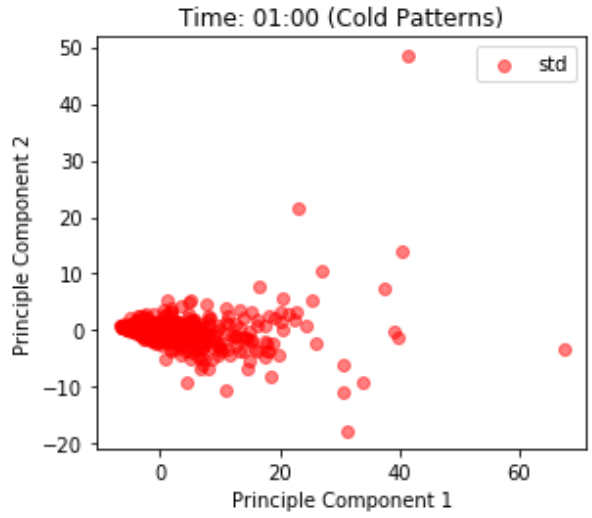

Time: 21:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

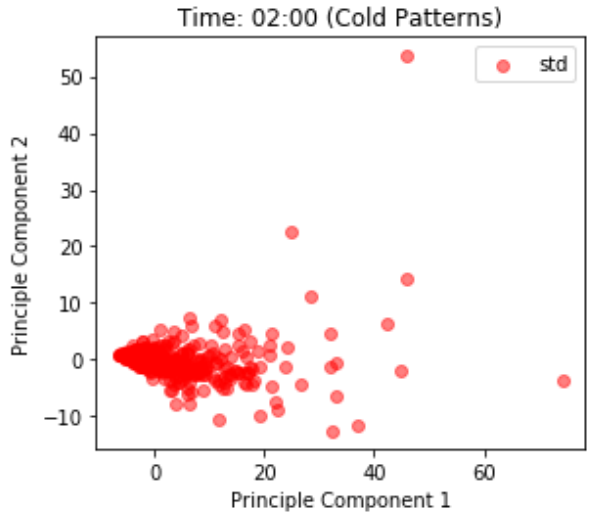

Time: 22:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

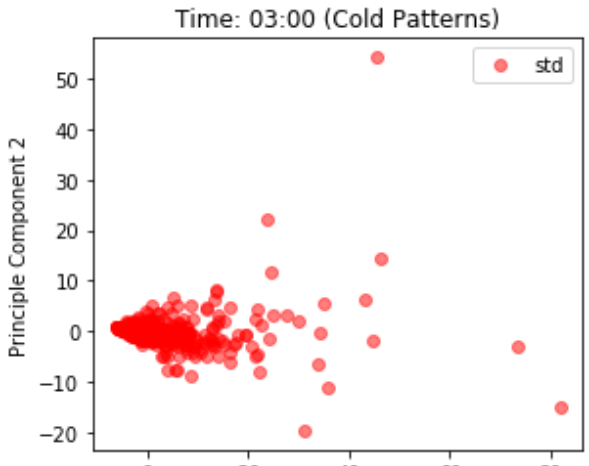

Time: 23:00 (Cold Patterns)
non-std
Principle Component 2
Principle Component 1

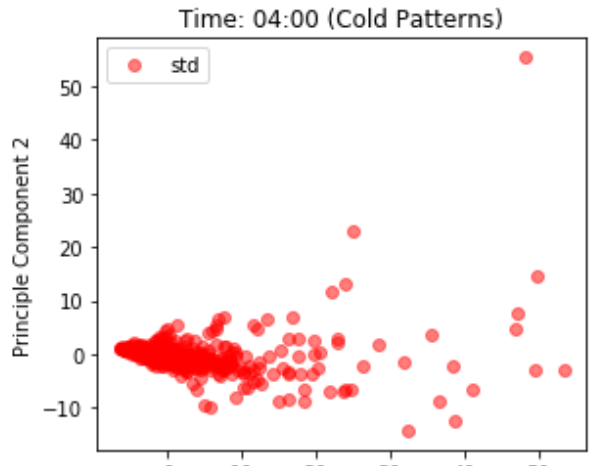
In [253]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[0], principleDf_nstd[1], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[0], principleDf[1], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 2')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[0], principleDf_nstd[1], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[0], principleDf[1], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 2')
plt.tight_layout()
```

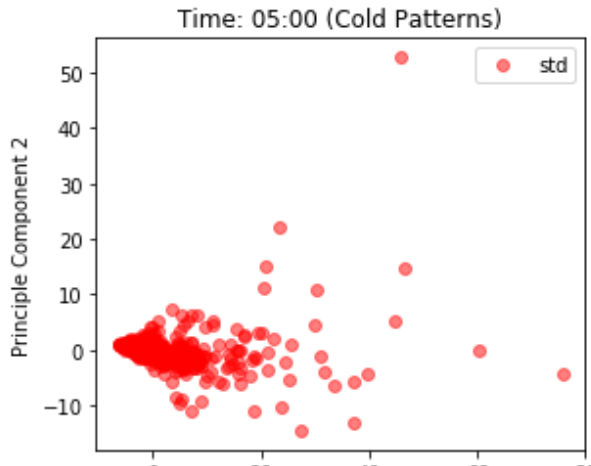

Time: 00:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

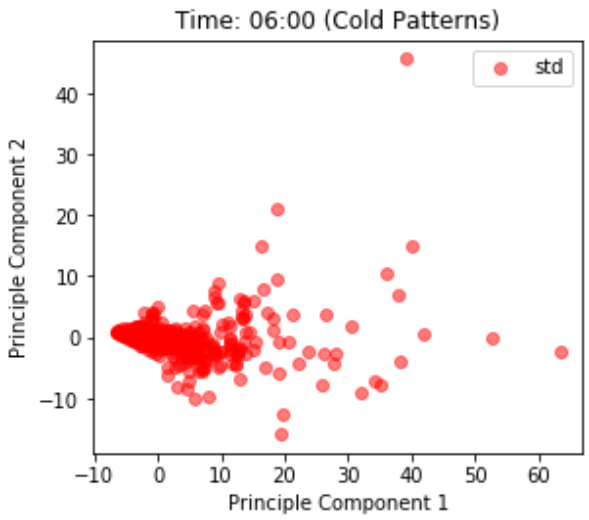

Time: 01:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

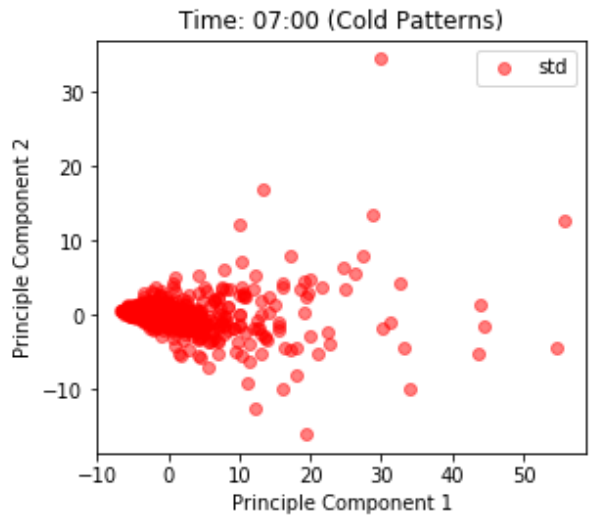

Time: 02:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

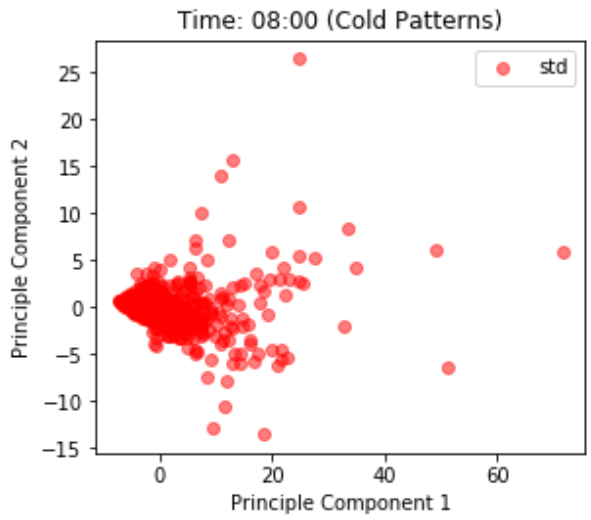

Time: 03:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

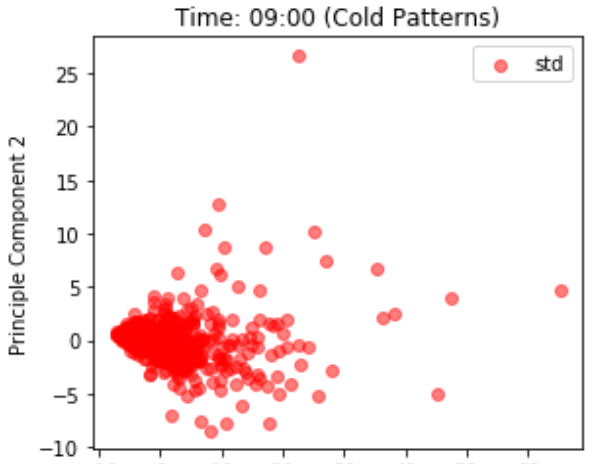

Time: 04:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

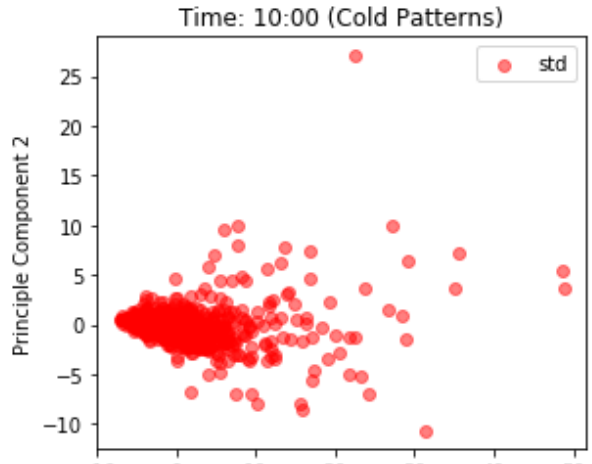

Time: 05:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

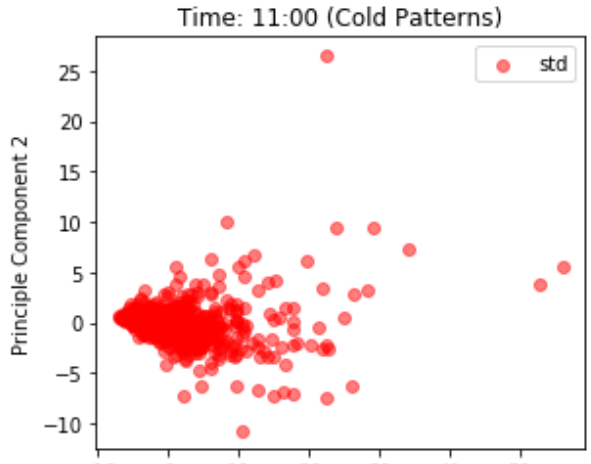

Time: 06:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

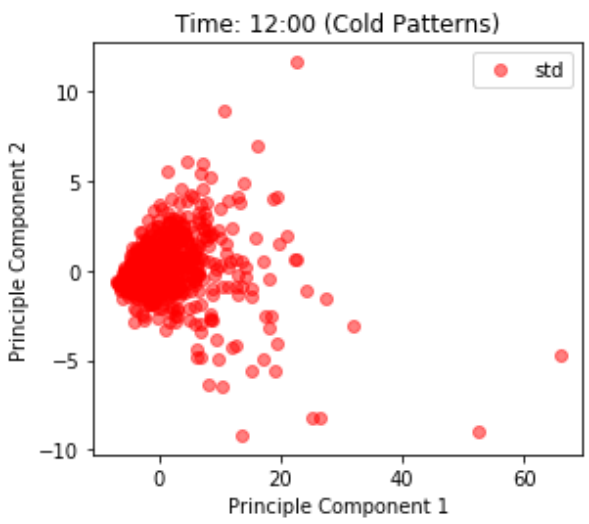

Time: 07:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

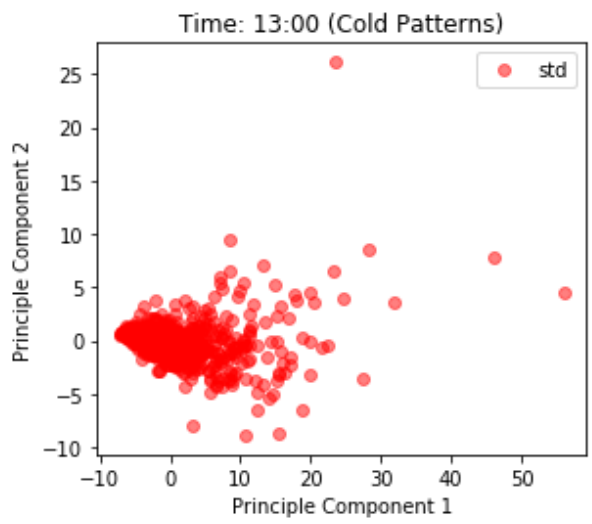

Time: 08:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

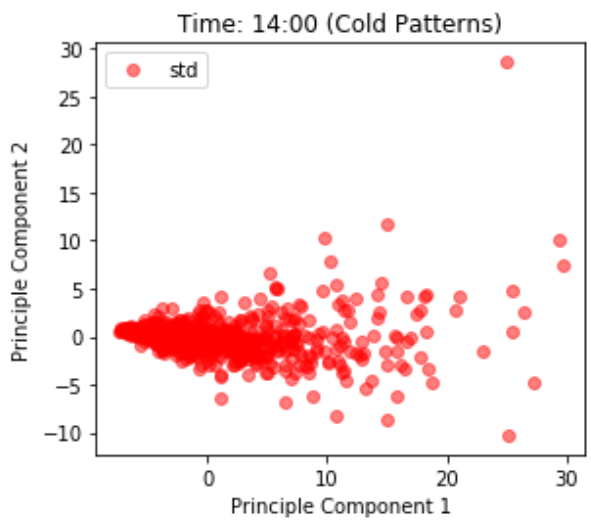

Time: 09:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

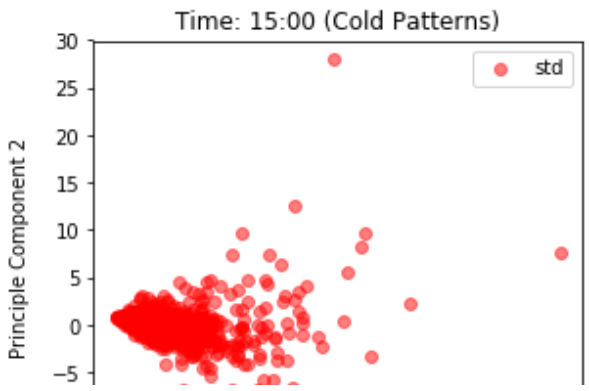

Time: 10:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

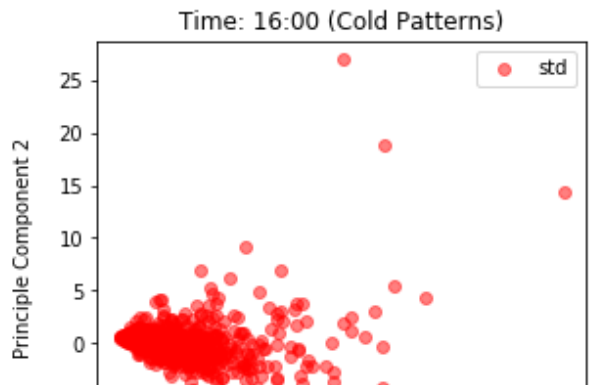

Time: 11:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

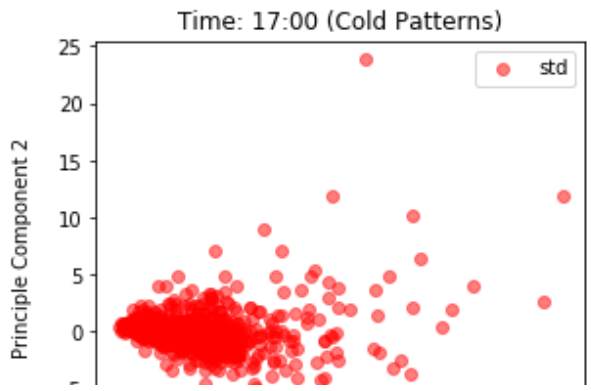

Time: 12:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1

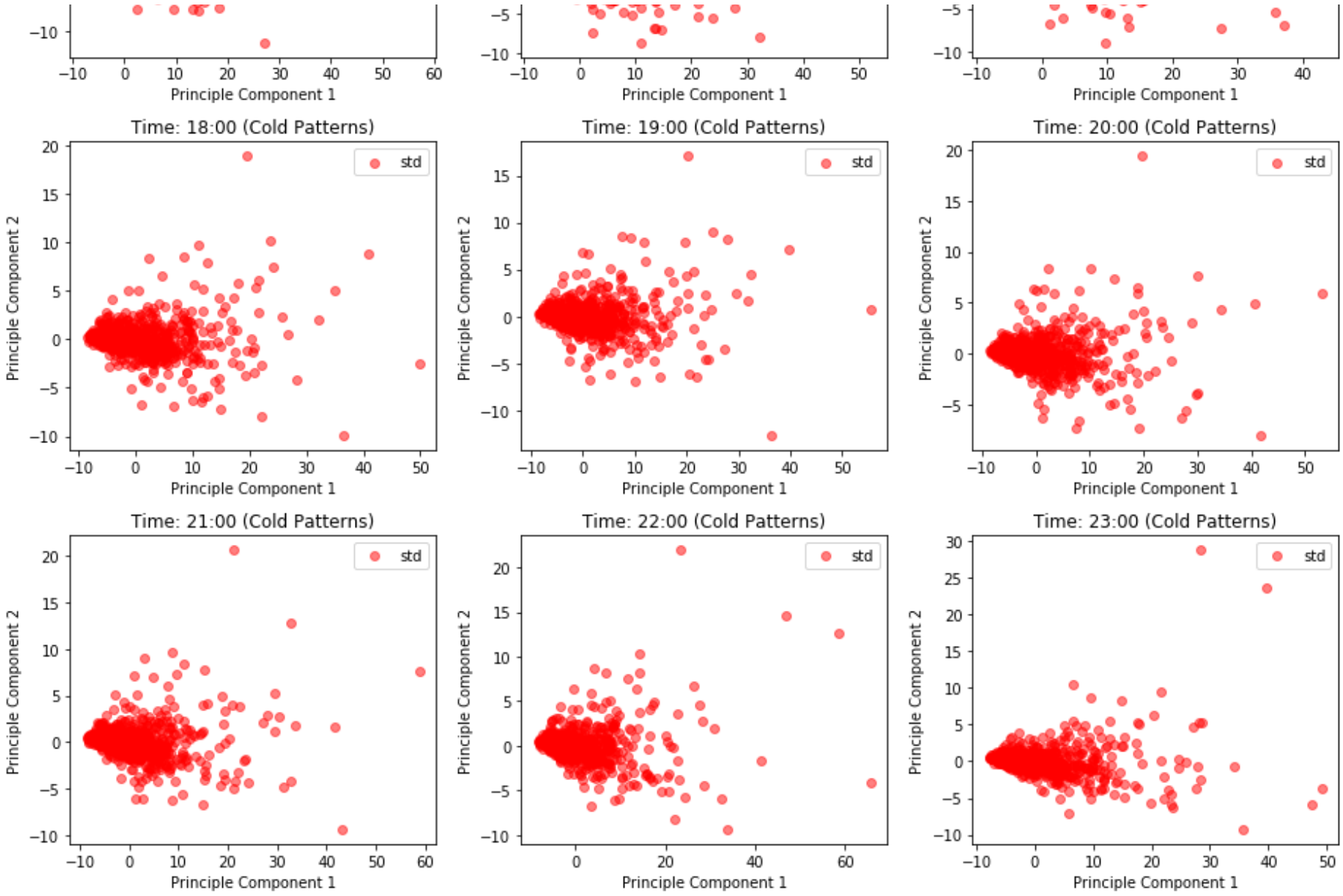

Time: 13:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1


Time: 14:00 (Cold Patterns)
std
Principle Component 2
Principle Component 1


Time: 15:00 (Cold Patterns)
std
Principle Component 2


Time: 16:00 (Cold Patterns)
std
Principle Component 2

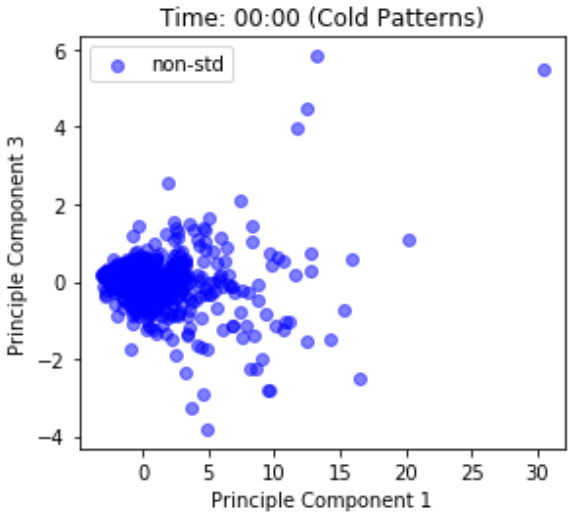

Time: 17:00 (Cold Patterns)
std
Principle Component 2

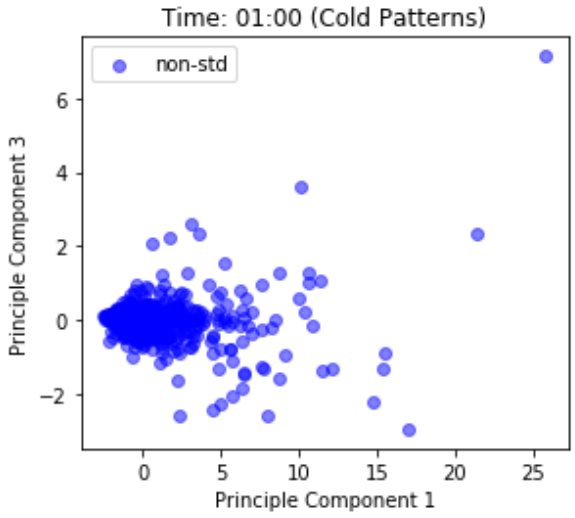


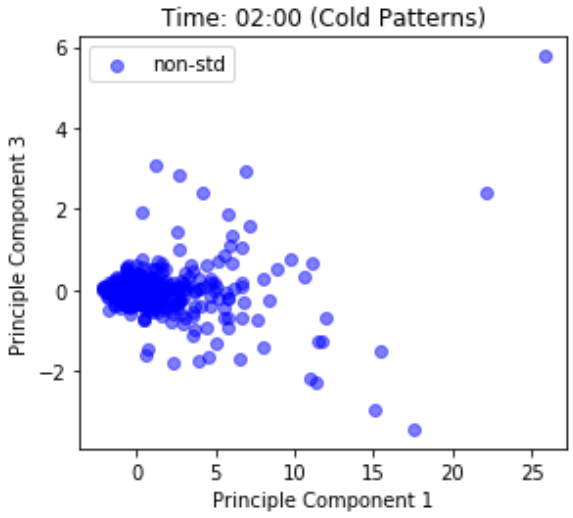
Different from the size-and-variable both standardization in paper 2, we care about the size effect. So we do not perform size standardization. Also in paper2, the author worried about the correlated problems, which wouldn't appear if they didn't use the no-size-scaled PCA to perform K-mean clustering. So the reason that problem seems to exist is they used the different PCA as the input of K-mean clustering and get the different labels, then put these labels to the previous PCA method, it seems that in Fig2 of their paper, each "category" seems has PC1 and PC2 linearly correlated, but this problem shouldn't have.

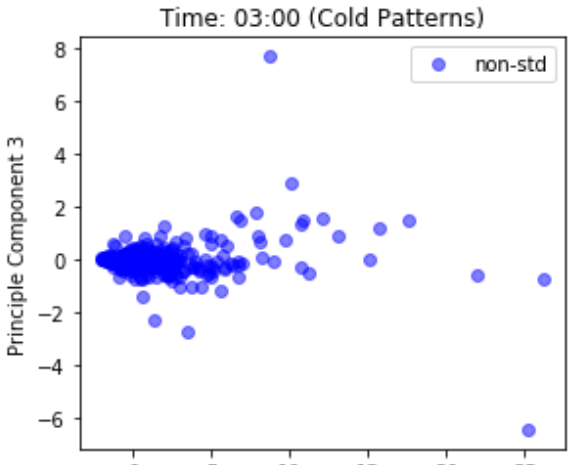
Component 1 and Component 3 relationship is shown below

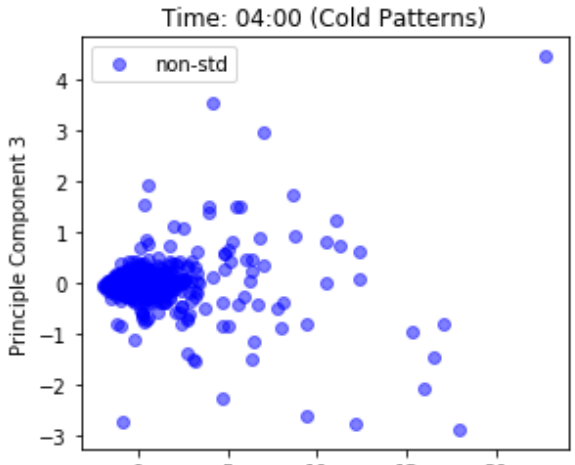
In [254]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[0], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[0], principleDf[2], c = 'red', label = 'std', alpha = 0.1)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 3')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[0], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[0], principleDf[2], c = 'red', label = 'std', alpha = 0.1)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 3')
plt.tight_layout()
```

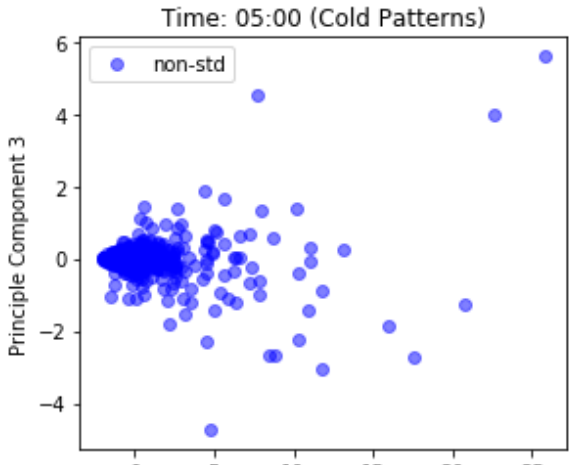

Time: 00:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

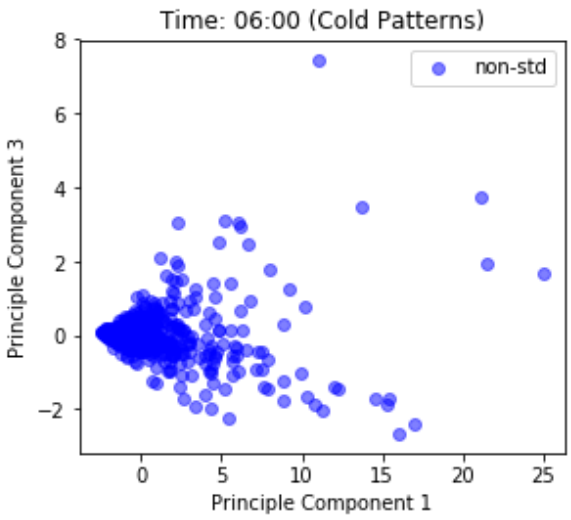

Time: 01:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

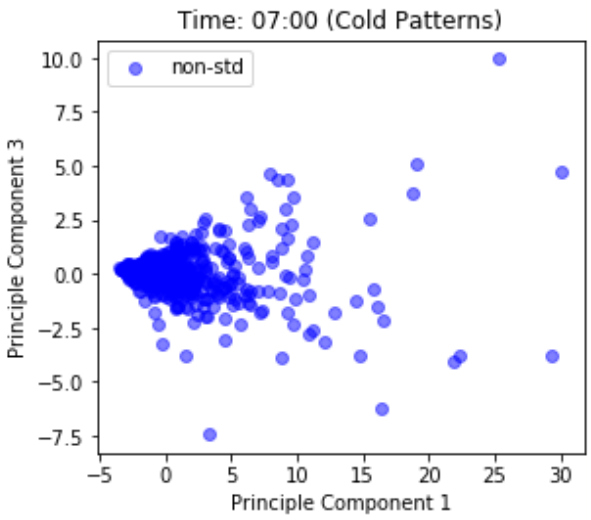

Time: 02:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

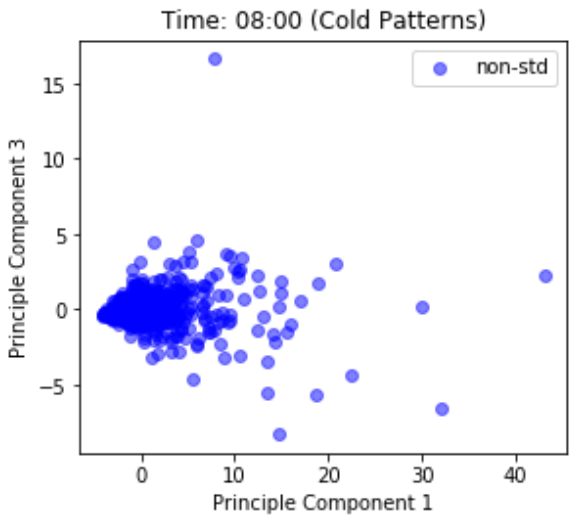

Time: 03:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

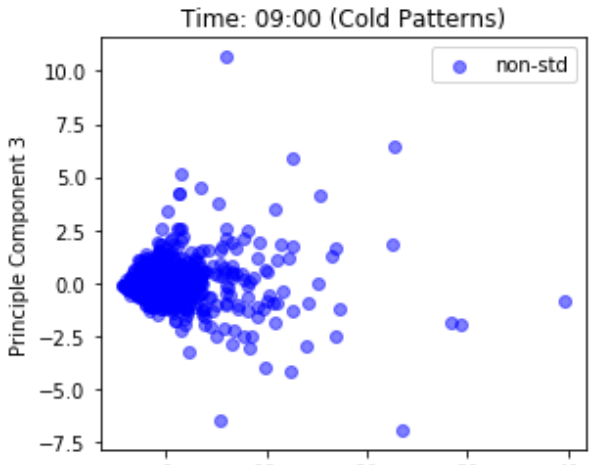

Time: 04:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

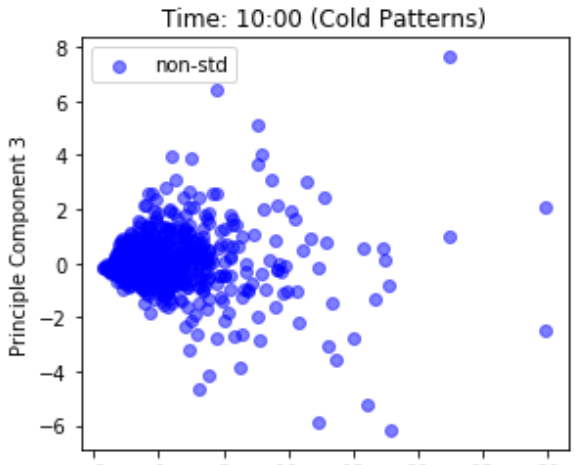

Time: 05:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

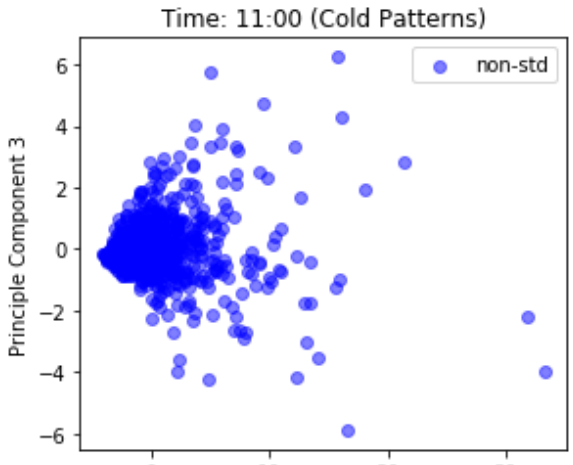

Time: 06:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

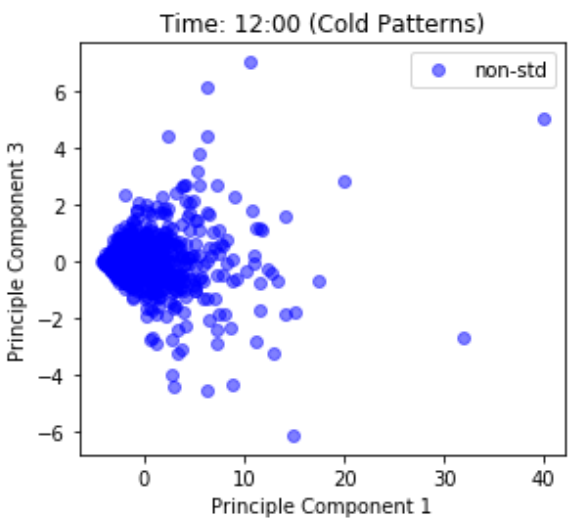

Time: 07:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

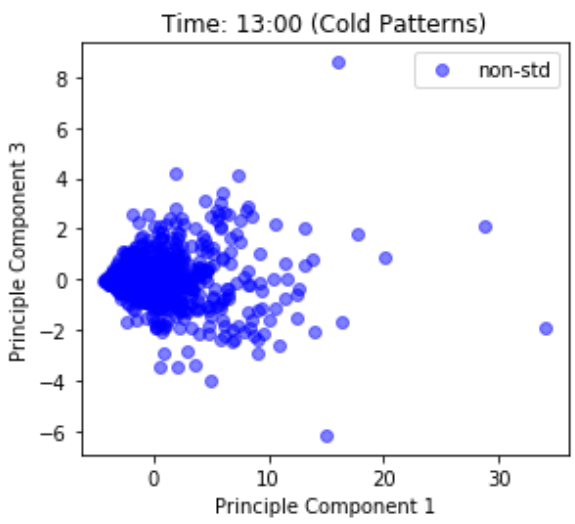

Time: 08:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

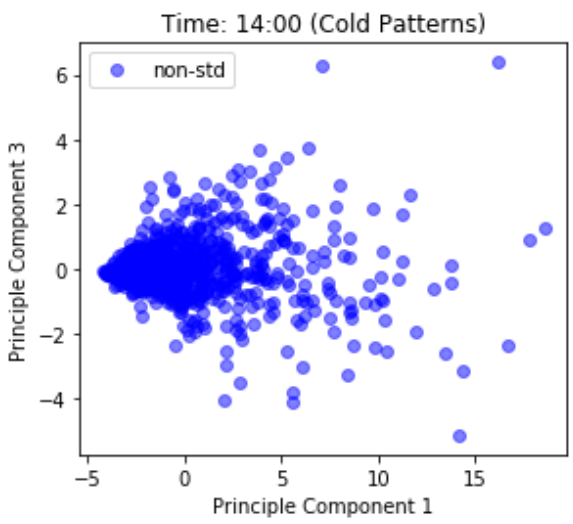

Time: 09:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

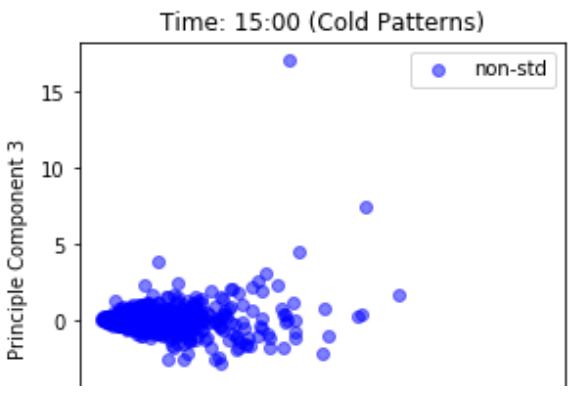

Time: 10:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

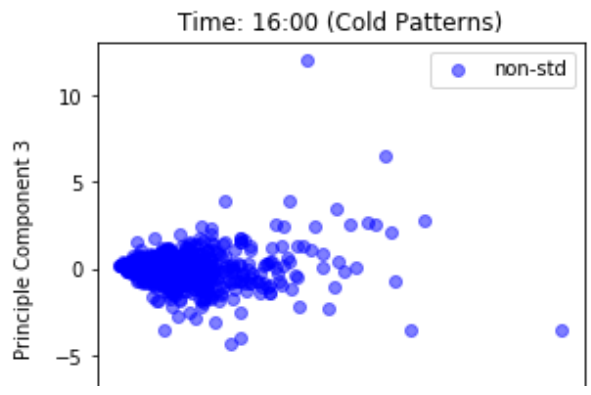

Time: 11:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

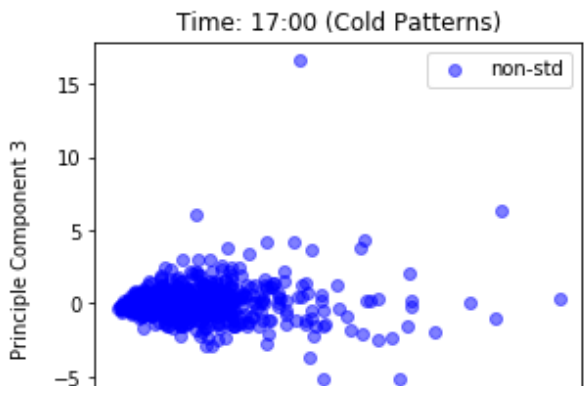

Time: 12:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

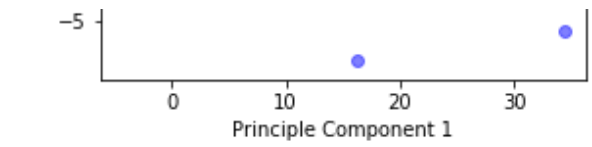

Time: 13:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

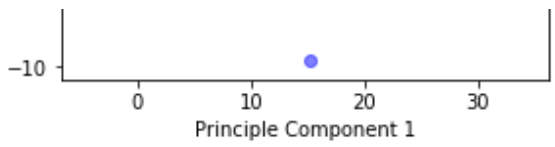

Time: 14:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

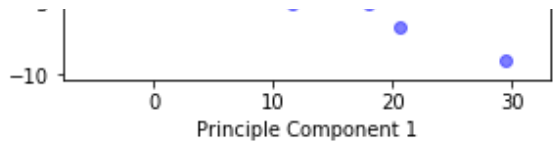

Time: 15:00 (Cold Patterns)
non-std
Principle Component 3

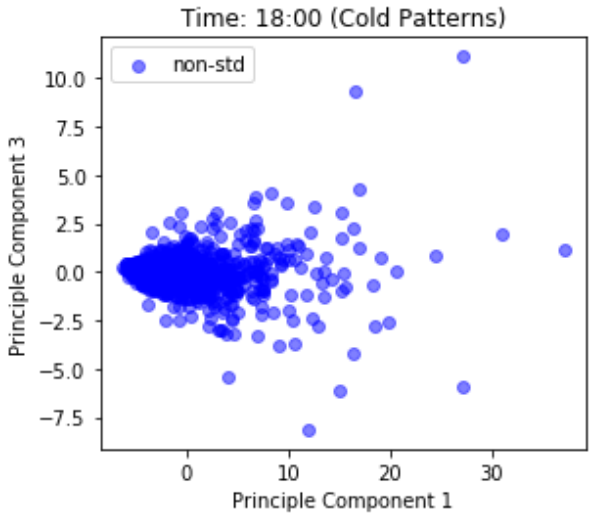

Time: 16:00 (Cold Patterns)
non-std
Principle Component 3

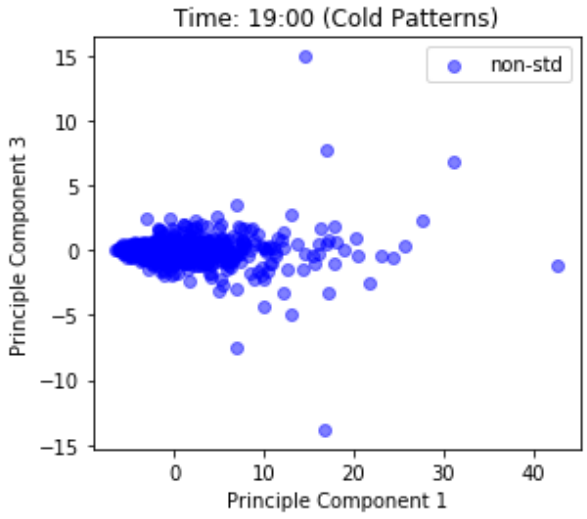

Time: 17:00 (Cold Patterns)
non-std
Principle Component 3

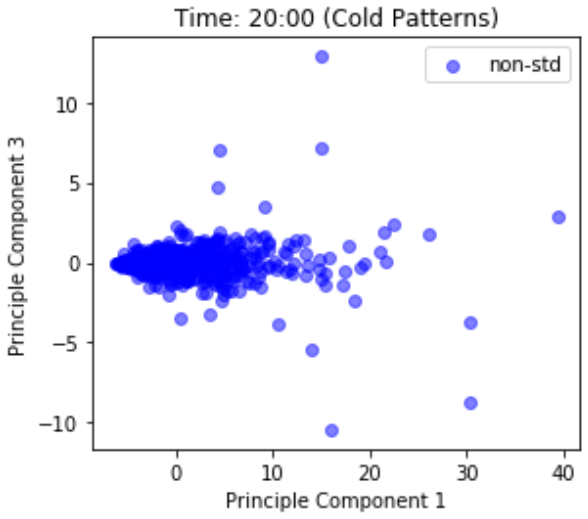

Principle Component 1

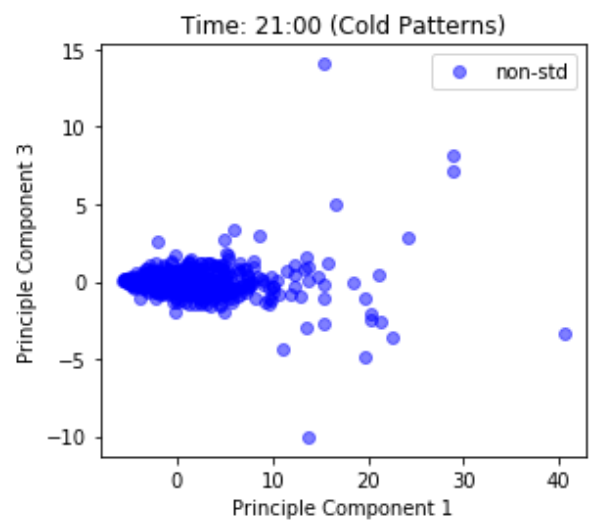

Principle Component 1

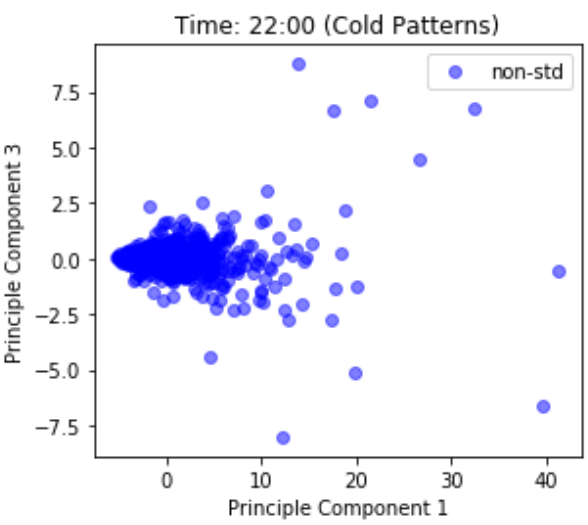

Principle Component 1

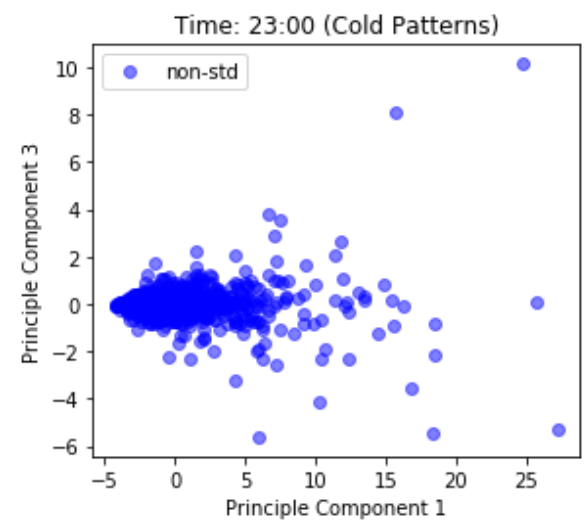

Time: 18:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1


Time: 19:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

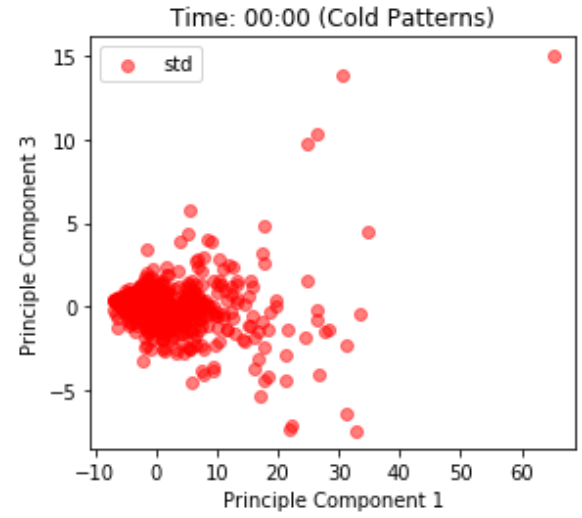

Time: 20:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

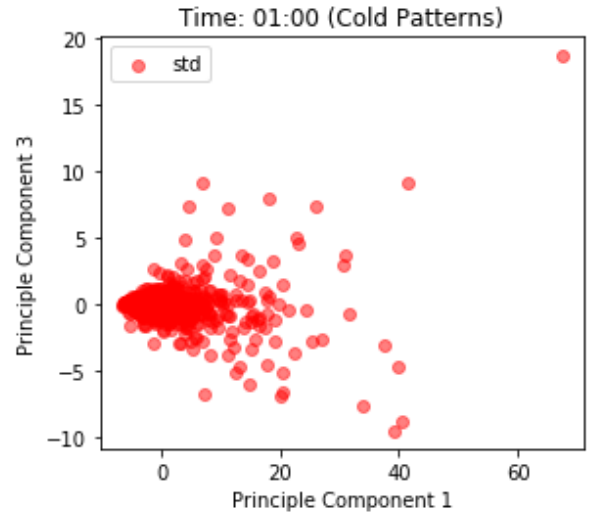

Time: 21:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

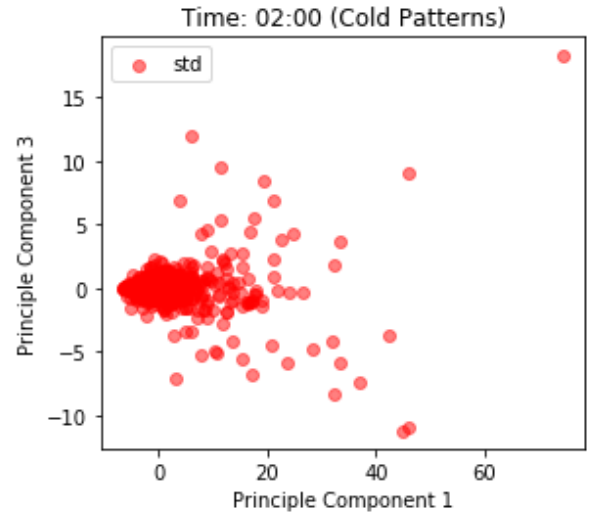

Time: 22:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

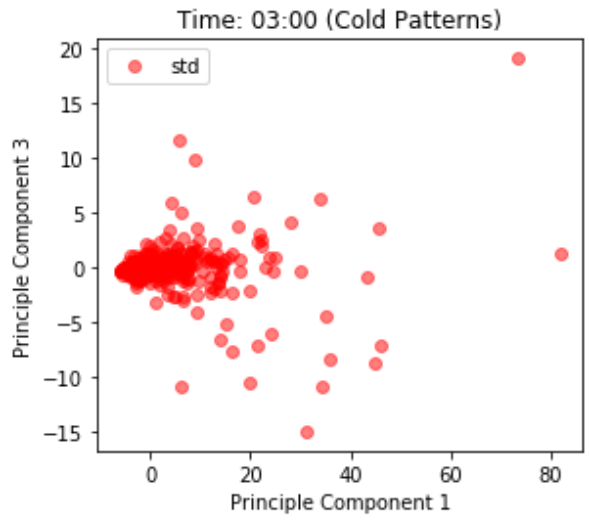

Time: 23:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 1

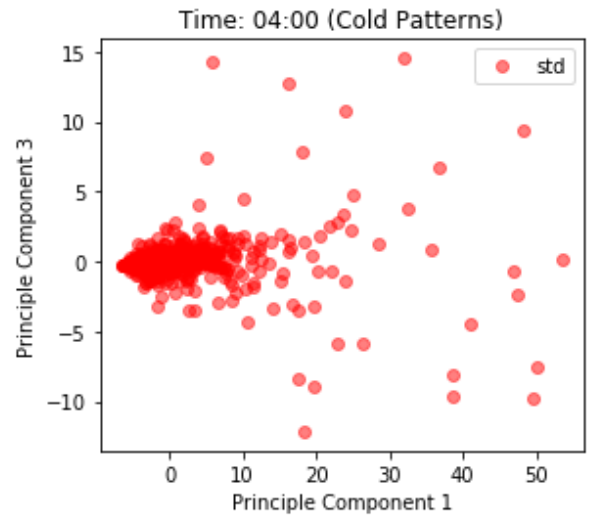
In [256]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[0], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[0], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 3')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[0], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[0], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 1')
        ax.set_ylabel('Principle Component 3')
plt.tight_layout()
```

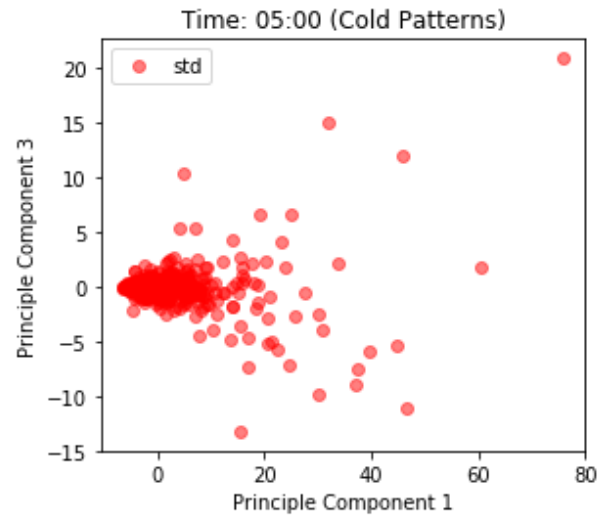

Time: 00:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

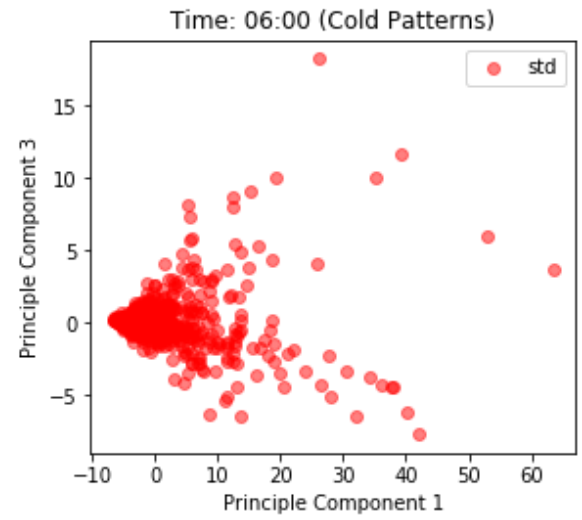

Time: 01:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

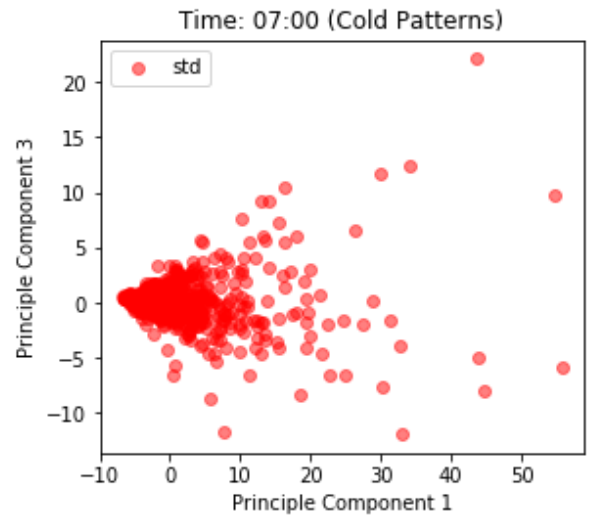

Time: 02:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

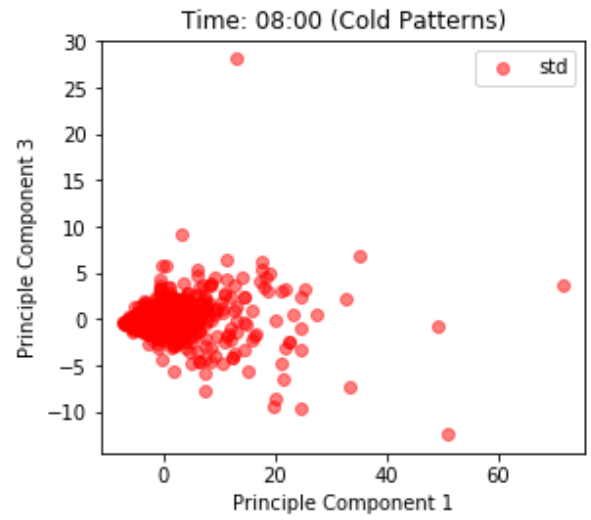

Time: 03:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

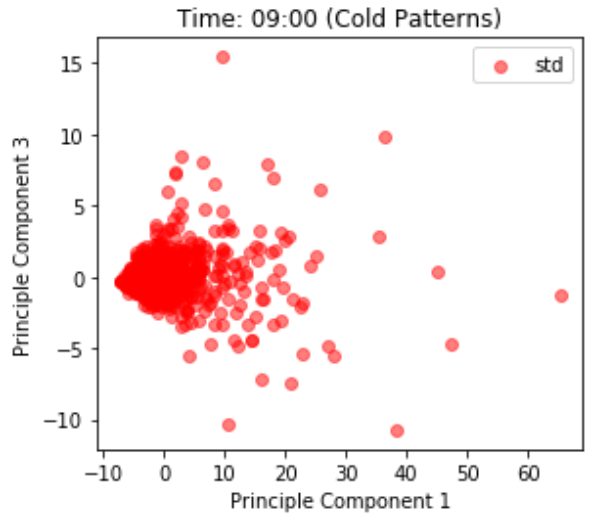

Time: 04:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

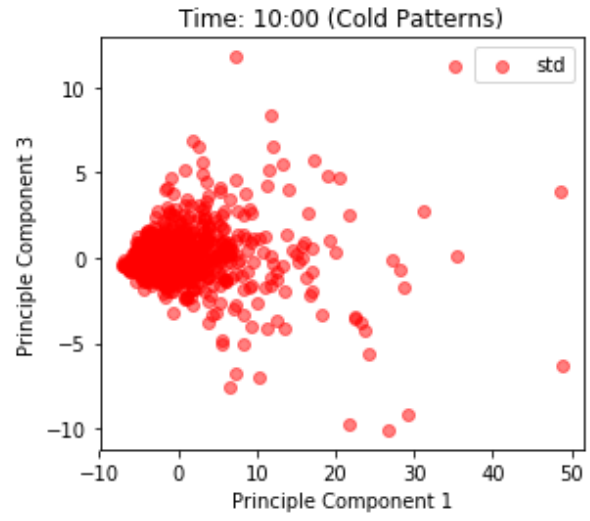

Time: 05:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

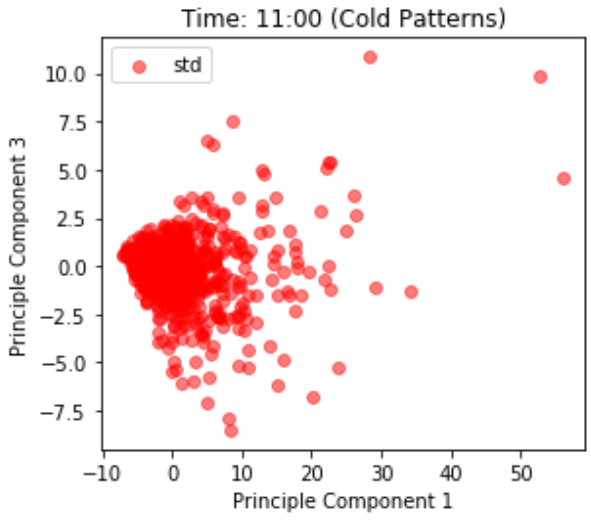

Time: 06:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

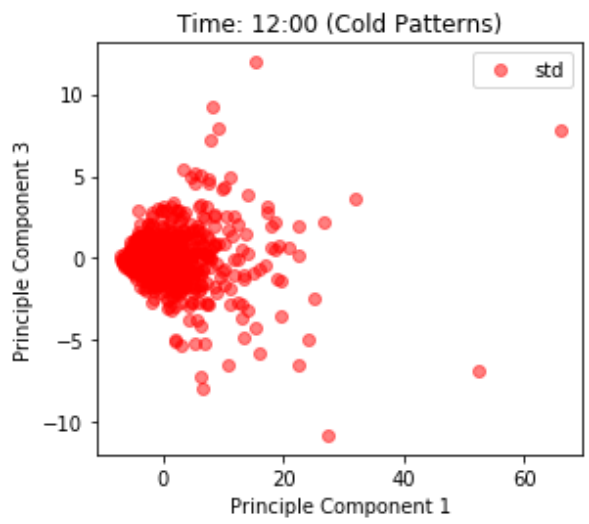

Time: 07:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

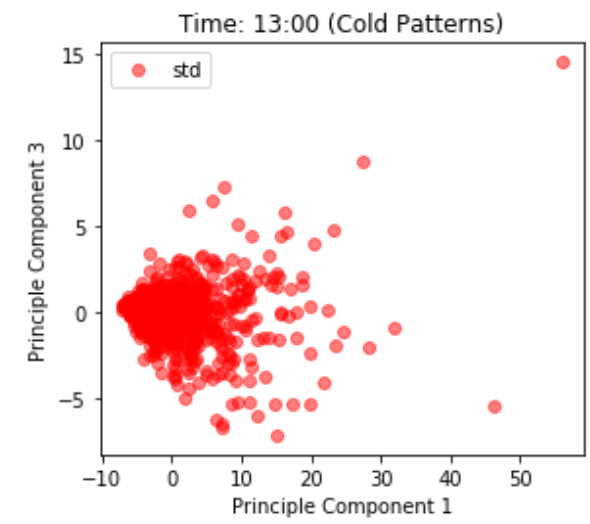

Time: 08:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

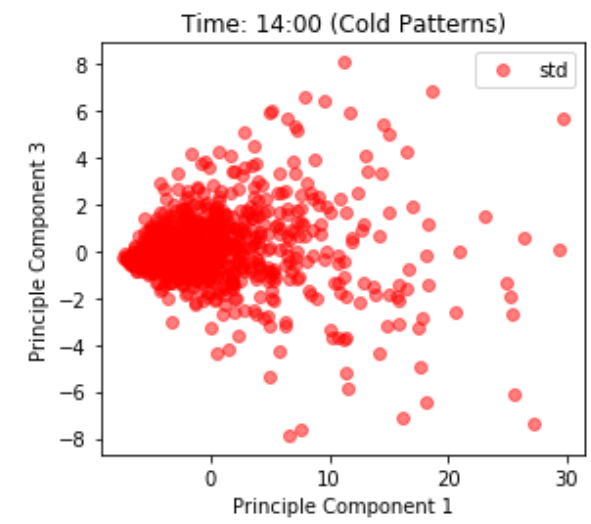

Time: 09:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

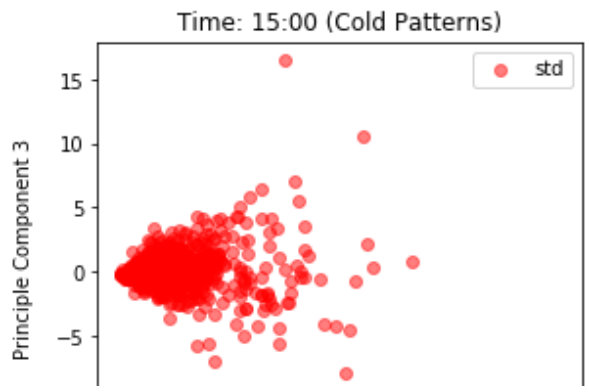

Time: 10:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

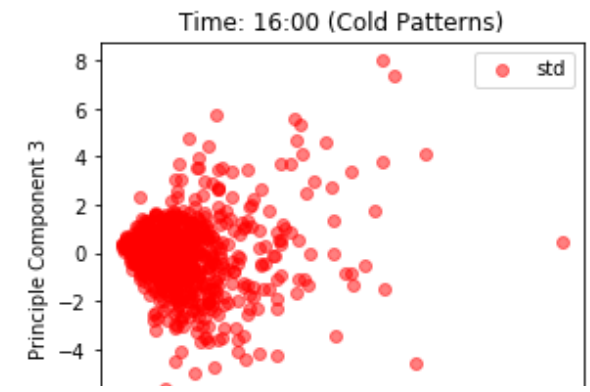

Time: 11:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

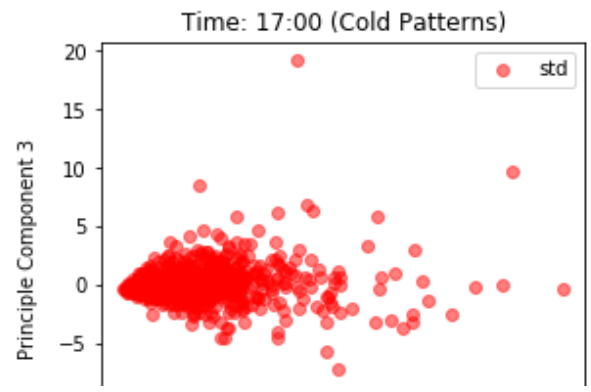

Time: 12:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1

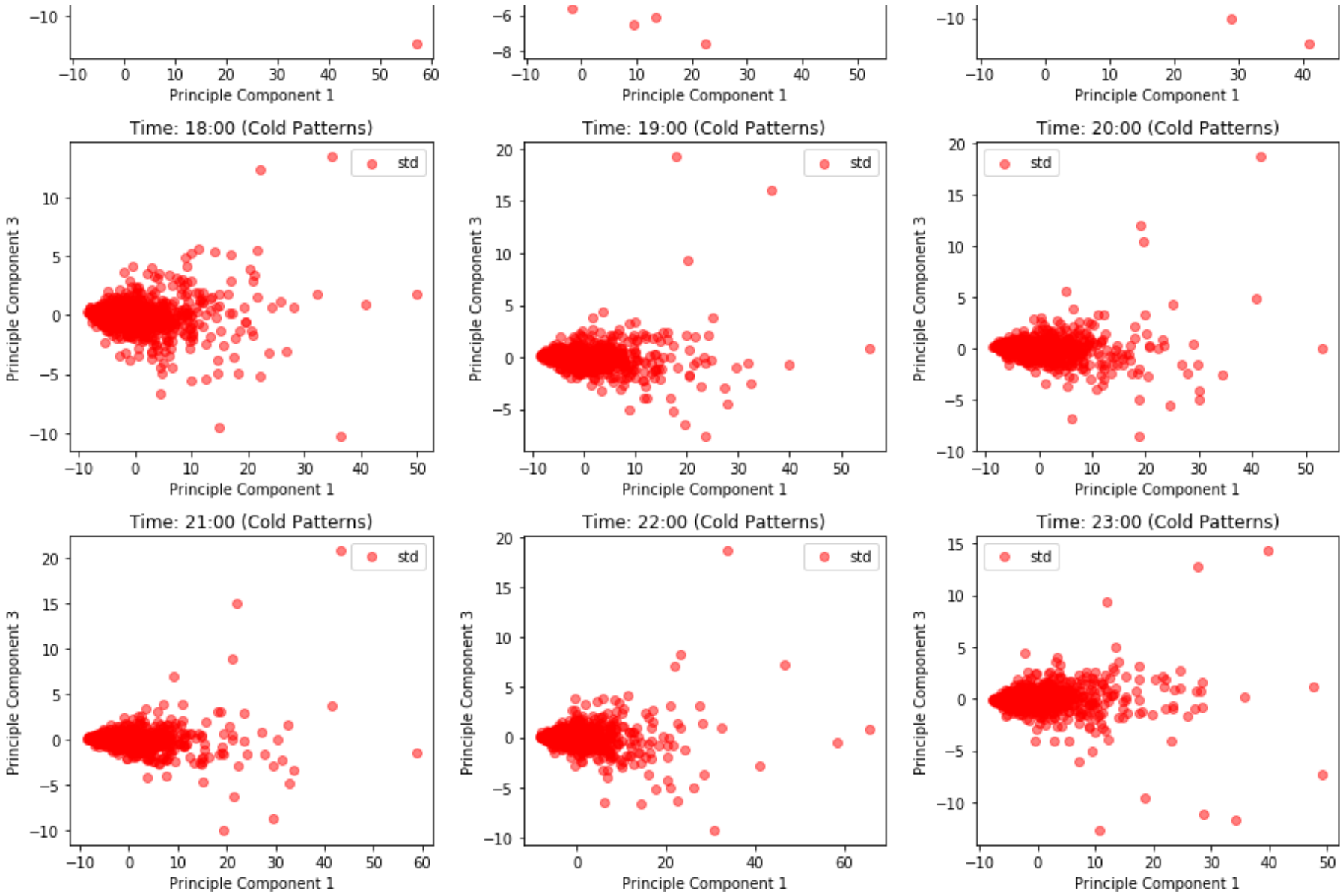

Time: 13:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1


Time: 14:00 (Cold Patterns)
std
Principle Component 3
Principle Component 1


Time: 15:00 (Cold Patterns)
std
Principle Component 3

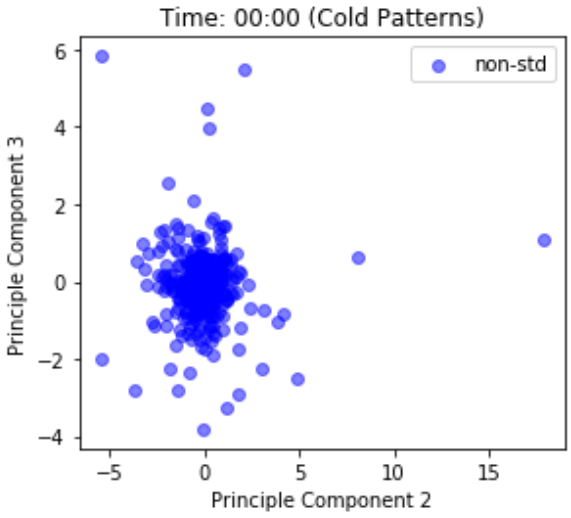

Time: 16:00 (Cold Patterns)
std
Principle Component 3

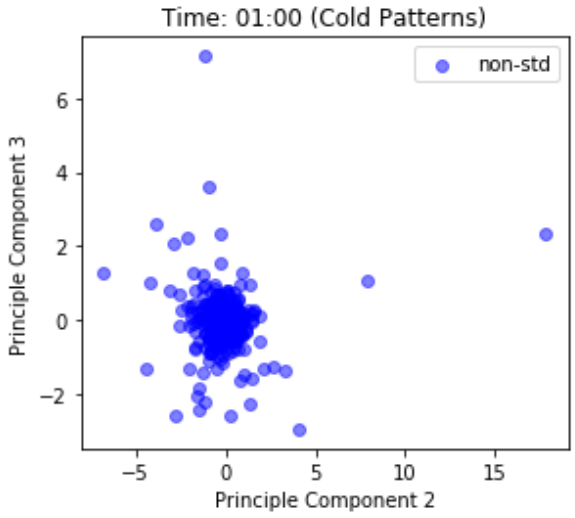

Time: 17:00 (Cold Patterns)
std
Principle Component 3

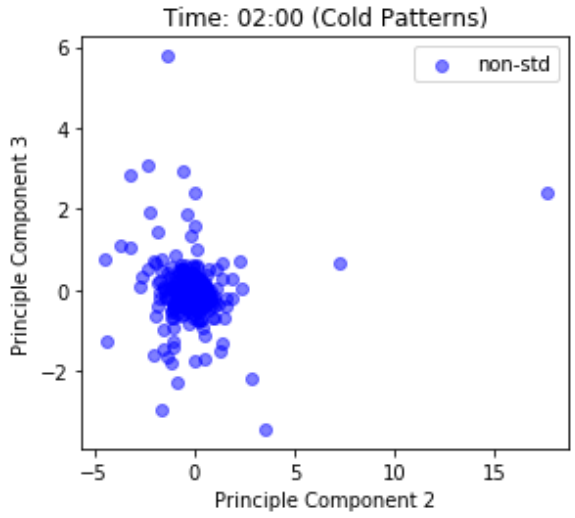


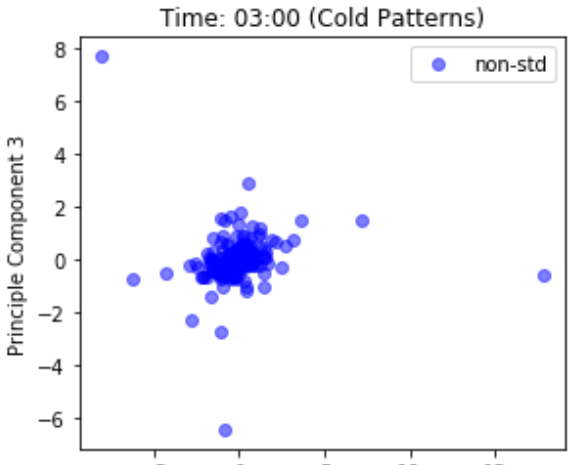
Component 2 and Component 3 relationship is shown below

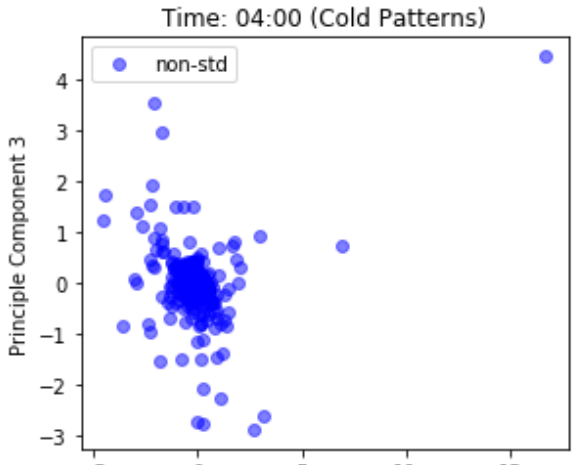
In [257]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[1], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 2')
        ax.set_ylabel('Principle Component 3')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
#         ax.scatter(principleDf[1], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 2')
        ax.set_ylabel('Principle Component 3')
plt.tight_layout()
```

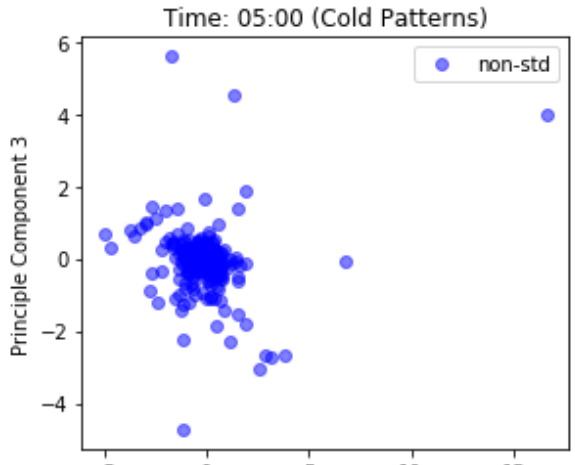

Time: 00:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

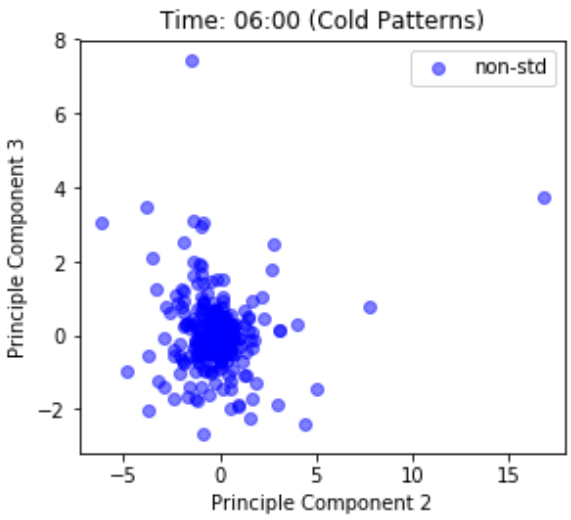

Time: 01:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

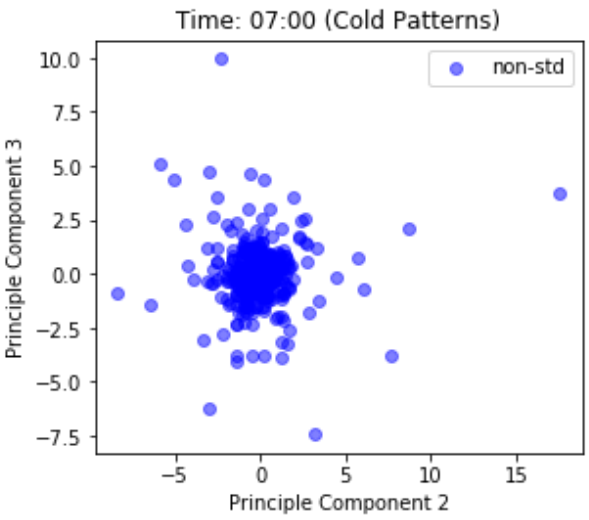

Time: 02:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

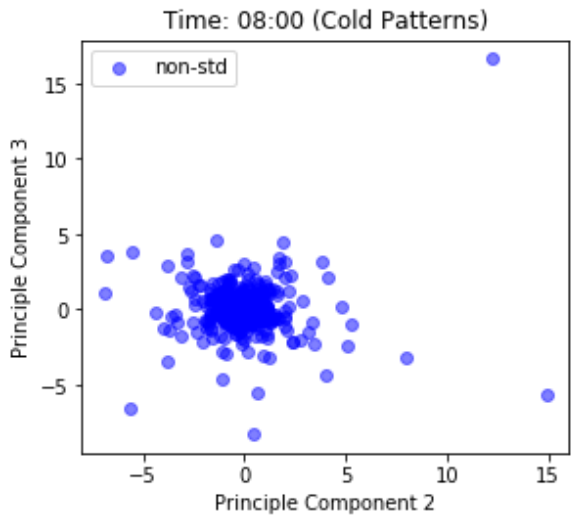

Time: 03:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

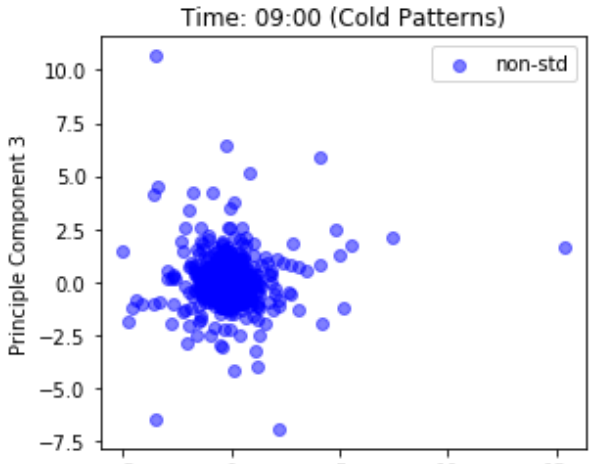

Time: 04:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

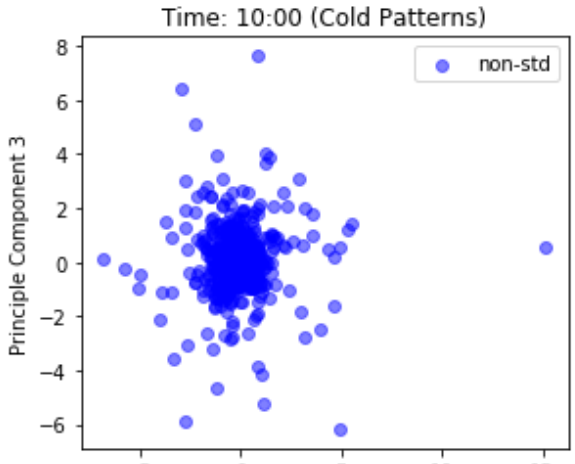

Time: 05:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

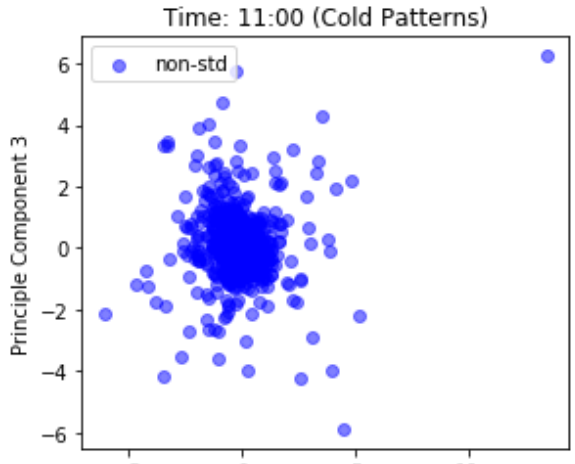

Time: 06:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

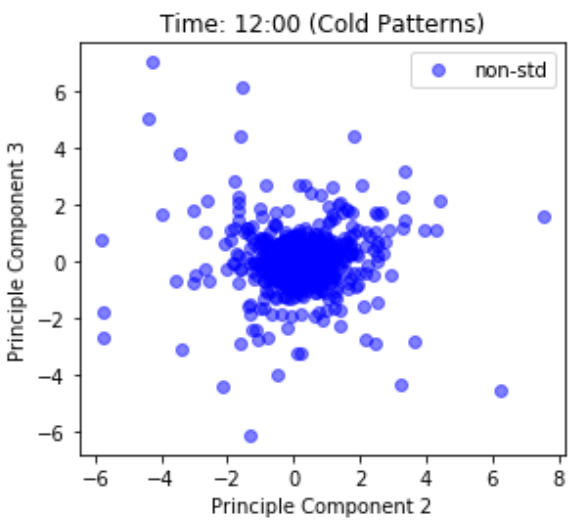

Time: 07:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

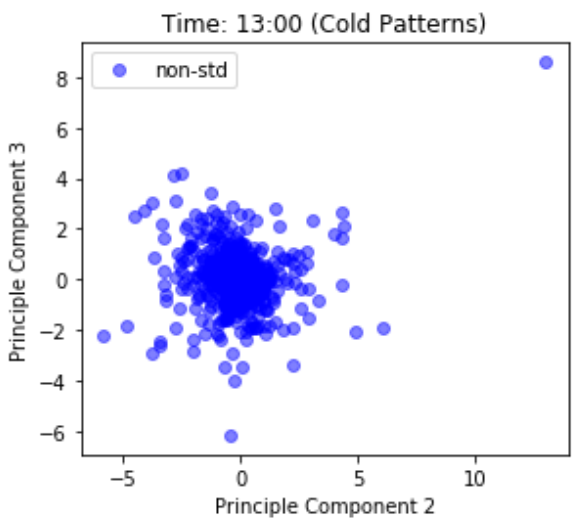

Time: 08:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

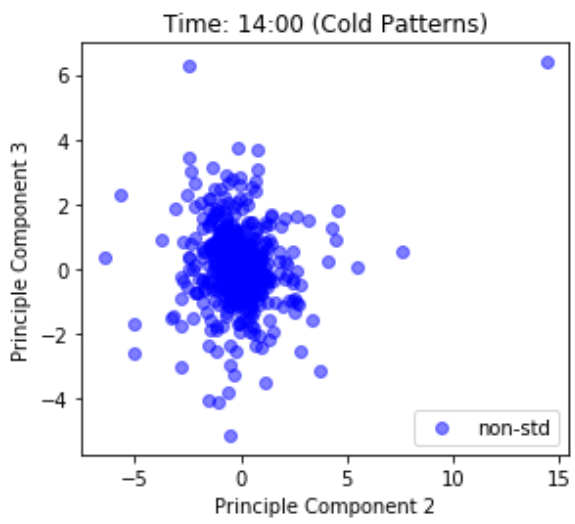

Time: 09:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

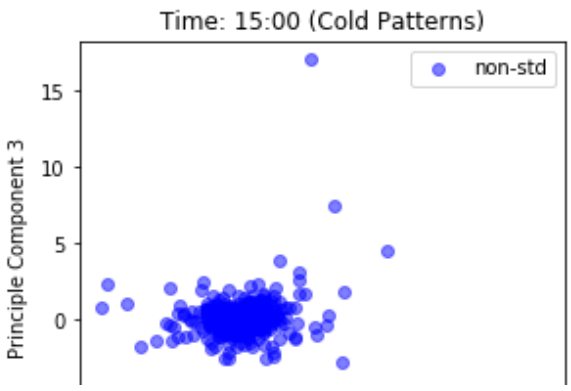

Time: 10:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

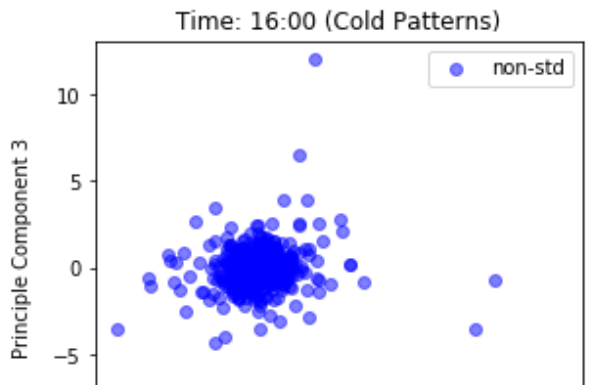

Time: 11:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

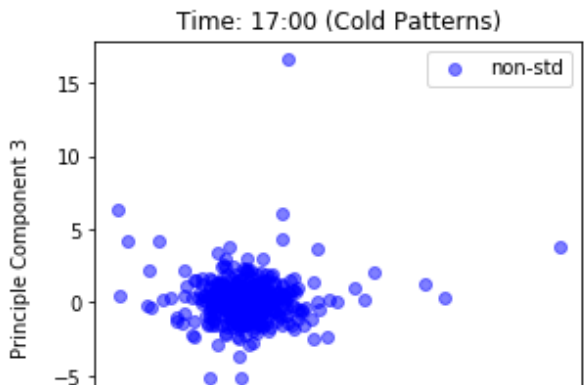

Time: 12:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

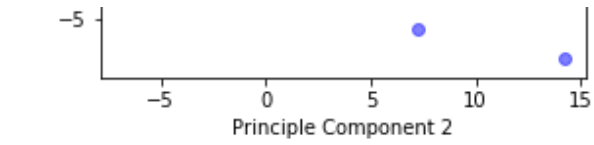

Time: 13:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

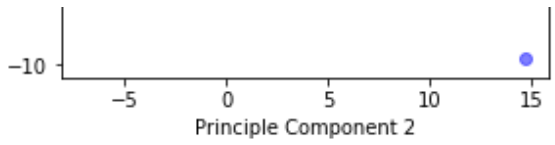

Time: 14:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

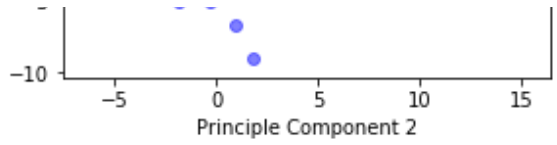

Time: 15:00 (Cold Patterns)
non-std
Principle Component 3

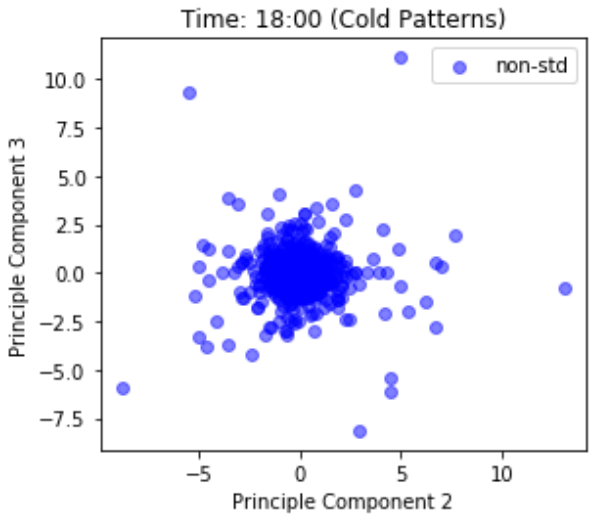

Time: 16:00 (Cold Patterns)
non-std
Principle Component 3

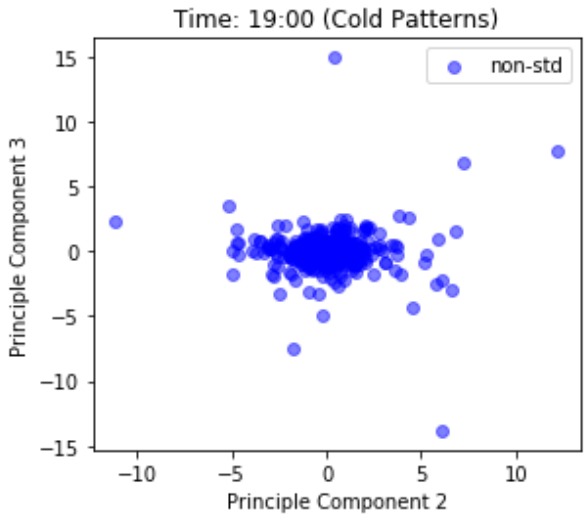

Time: 17:00 (Cold Patterns)
non-std
Principle Component 3

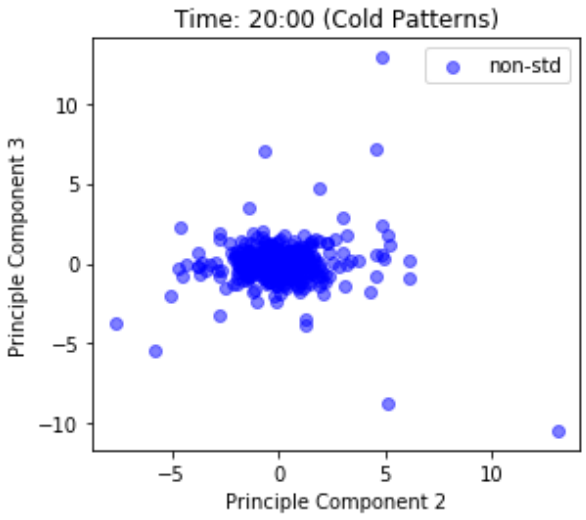

Principle Component 2

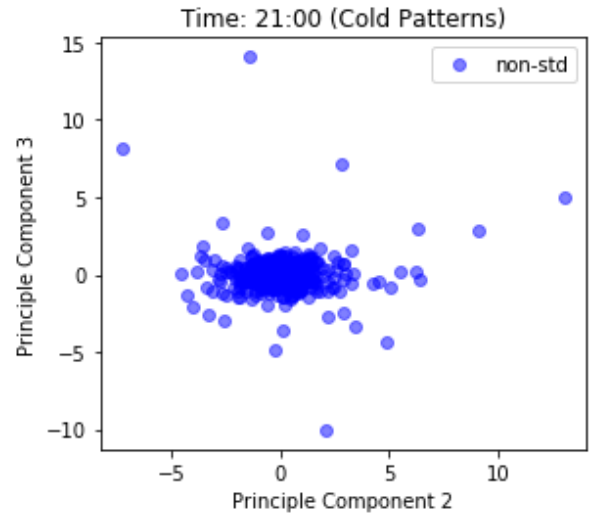

Principle Component 2

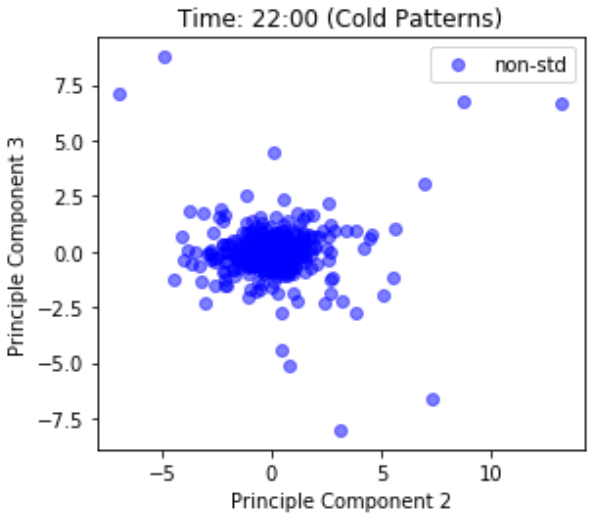

Principle Component 2

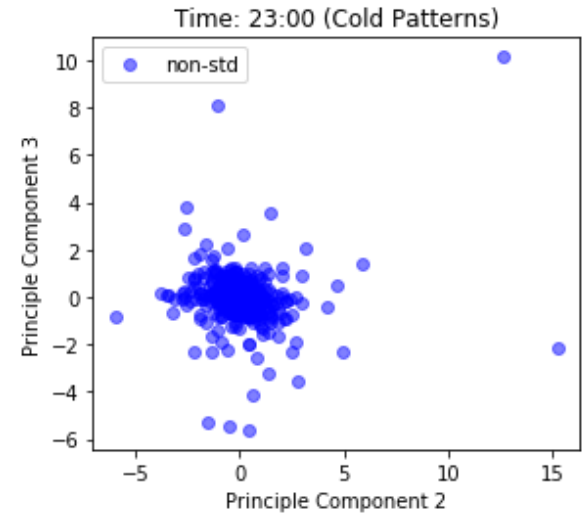

Time: 18:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2


Time: 19:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

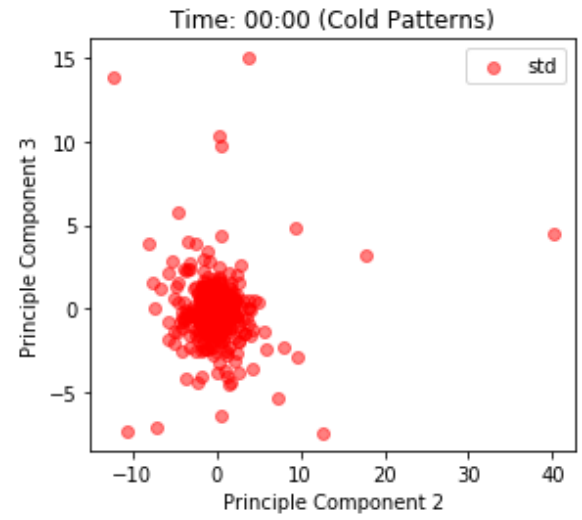

Time: 20:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

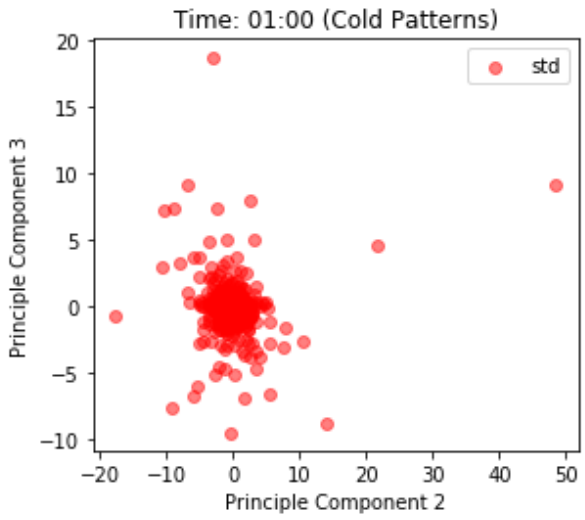

Time: 21:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

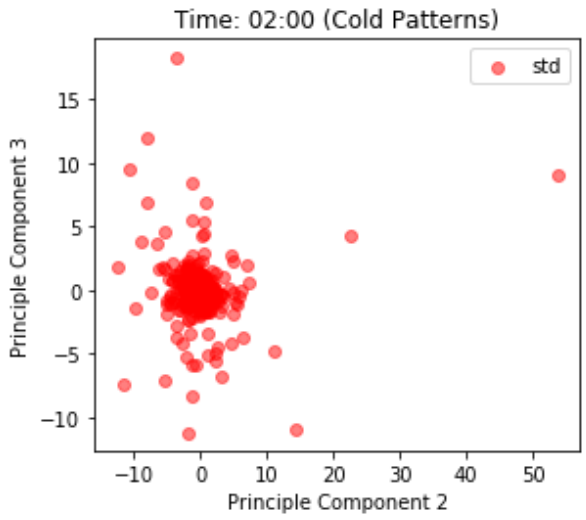

Time: 22:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

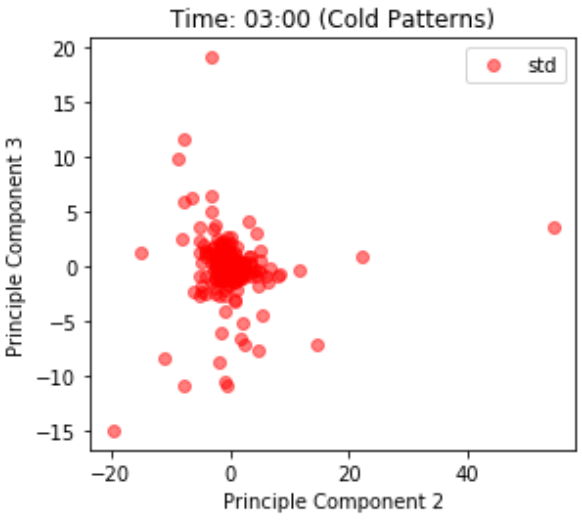

Time: 23:00 (Cold Patterns)
non-std
Principle Component 3
Principle Component 2

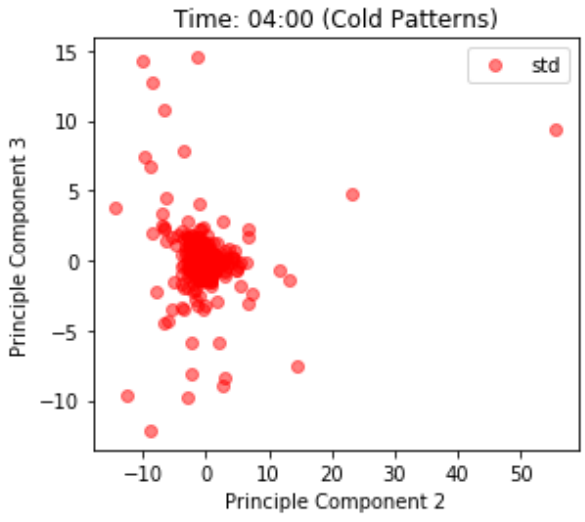
In [258]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[1], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[1], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 2')
        ax.set_ylabel('Principle Component 3')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         ax.scatter(principleDf_nstd[1], principleDf_nstd[2], c = 'blue', label = 'non-std', alpha = 0.5)
        # PCA with standardization
        x = x.values
        x = StandardScaler().fit_transform(x)
        pca = PCA()# keep all columns to cover 100% variance
        principleComponents = pca.fit_transform(x)
        principleDf = pd.DataFrame(data = principleComponents)
        ax.scatter(principleDf[1], principleDf[2], c = 'red', label = 'std', alpha = 0.5)
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Principle Component 2')
        ax.set_ylabel('Principle Component 3')
plt.tight_layout()
```

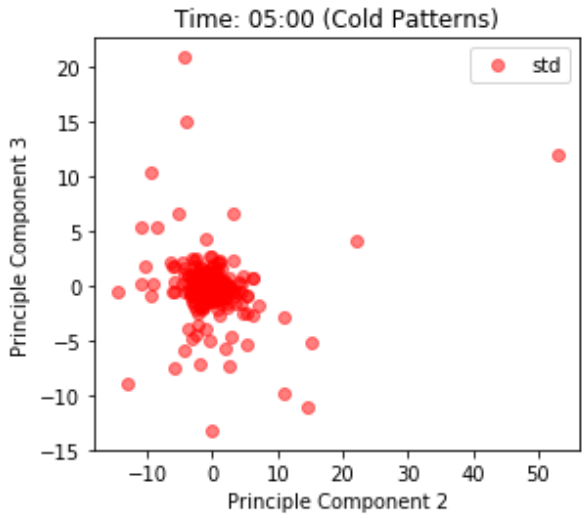

Time: 00:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

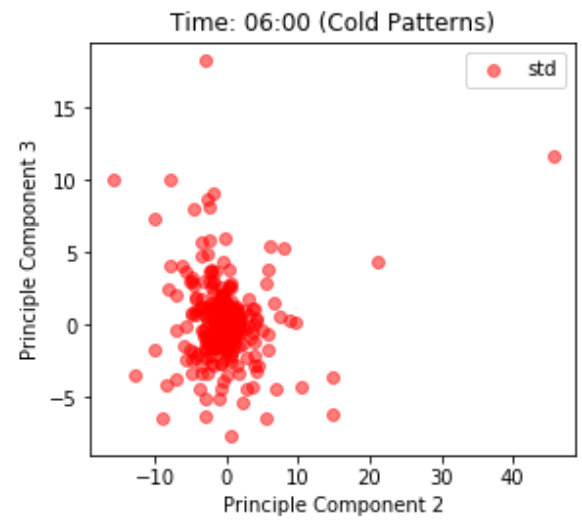

Time: 01:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

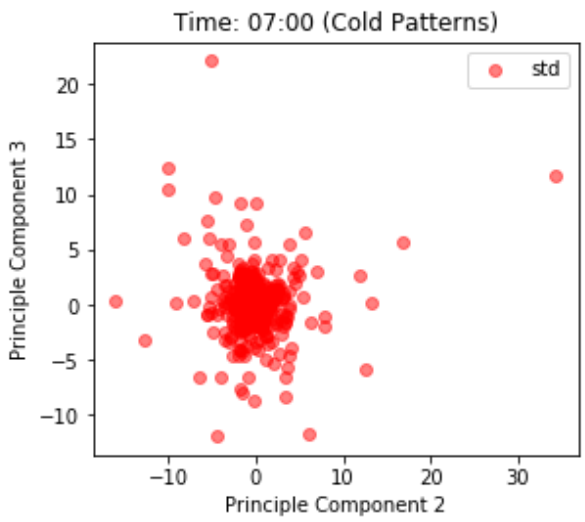

Time: 02:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

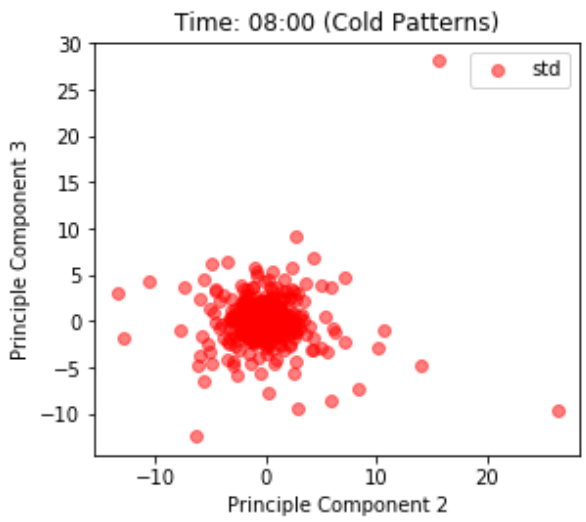

Time: 03:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

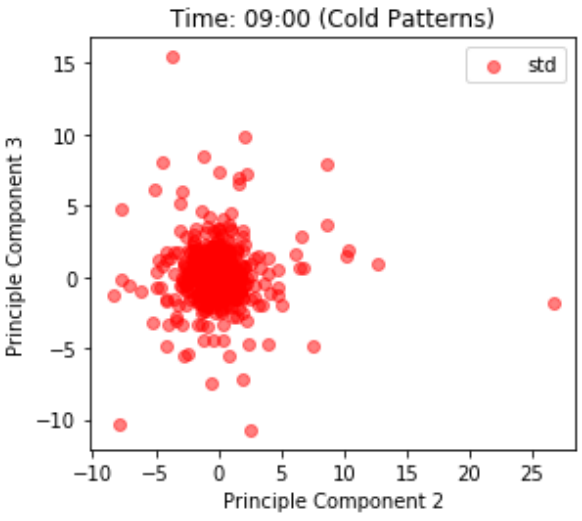

Time: 04:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

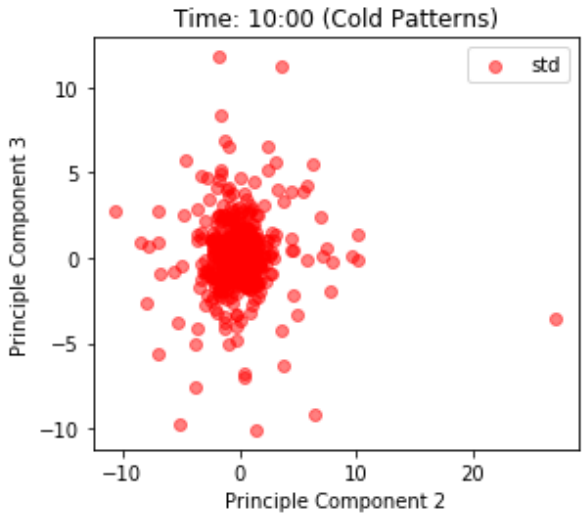

Time: 05:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

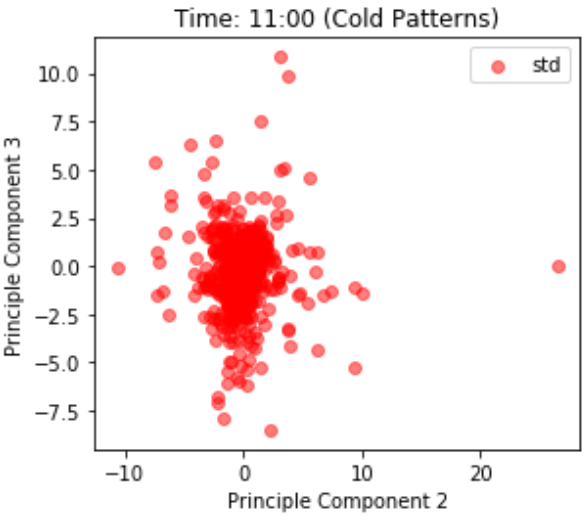

Time: 06:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

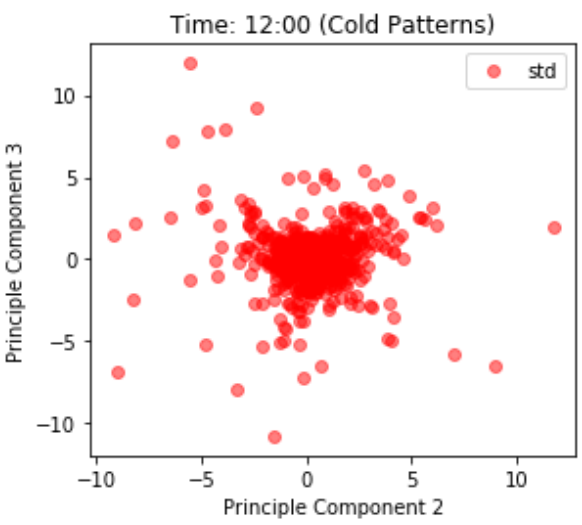

Time: 07:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

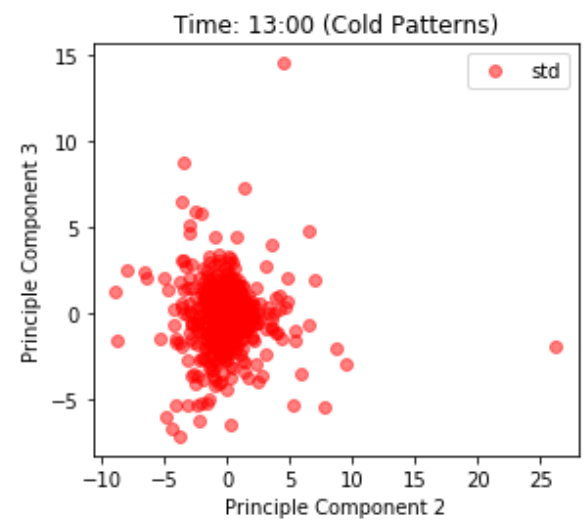

Time: 08:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

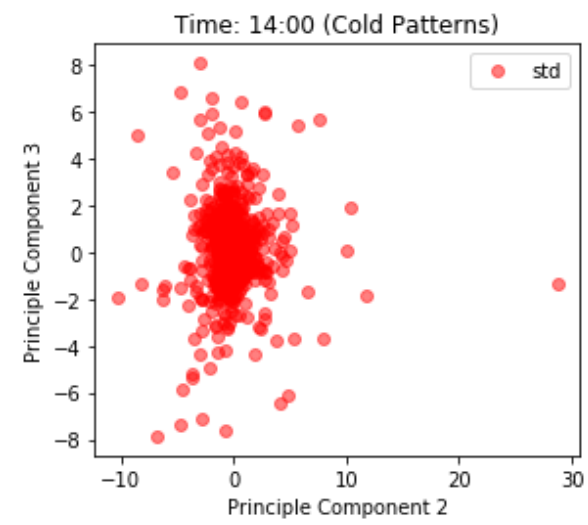

Time: 09:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

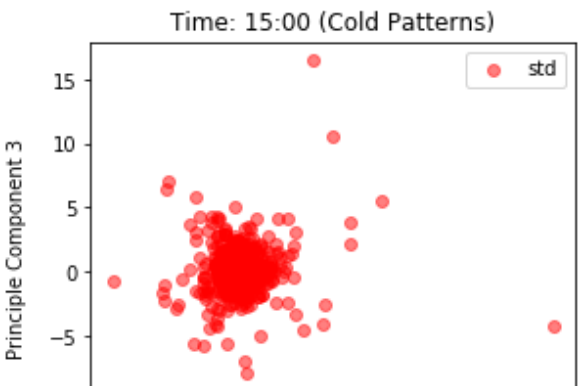

Time: 10:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

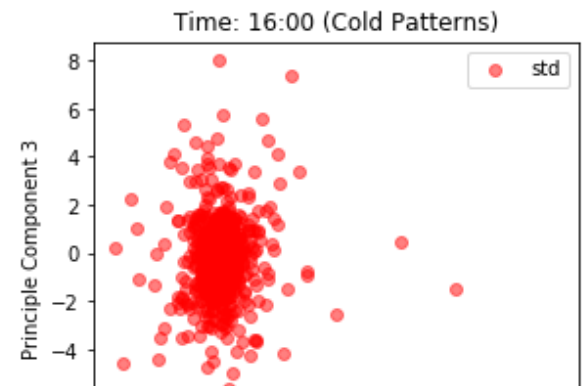

Time: 11:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

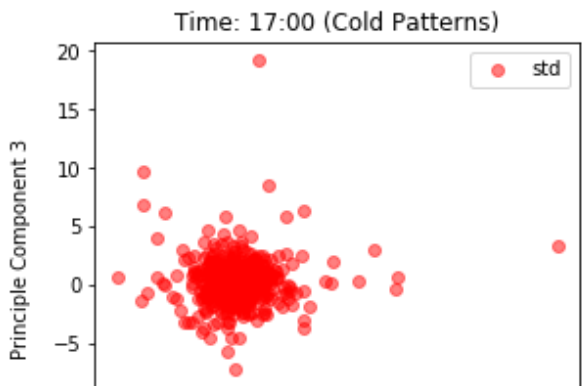

Time: 12:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2

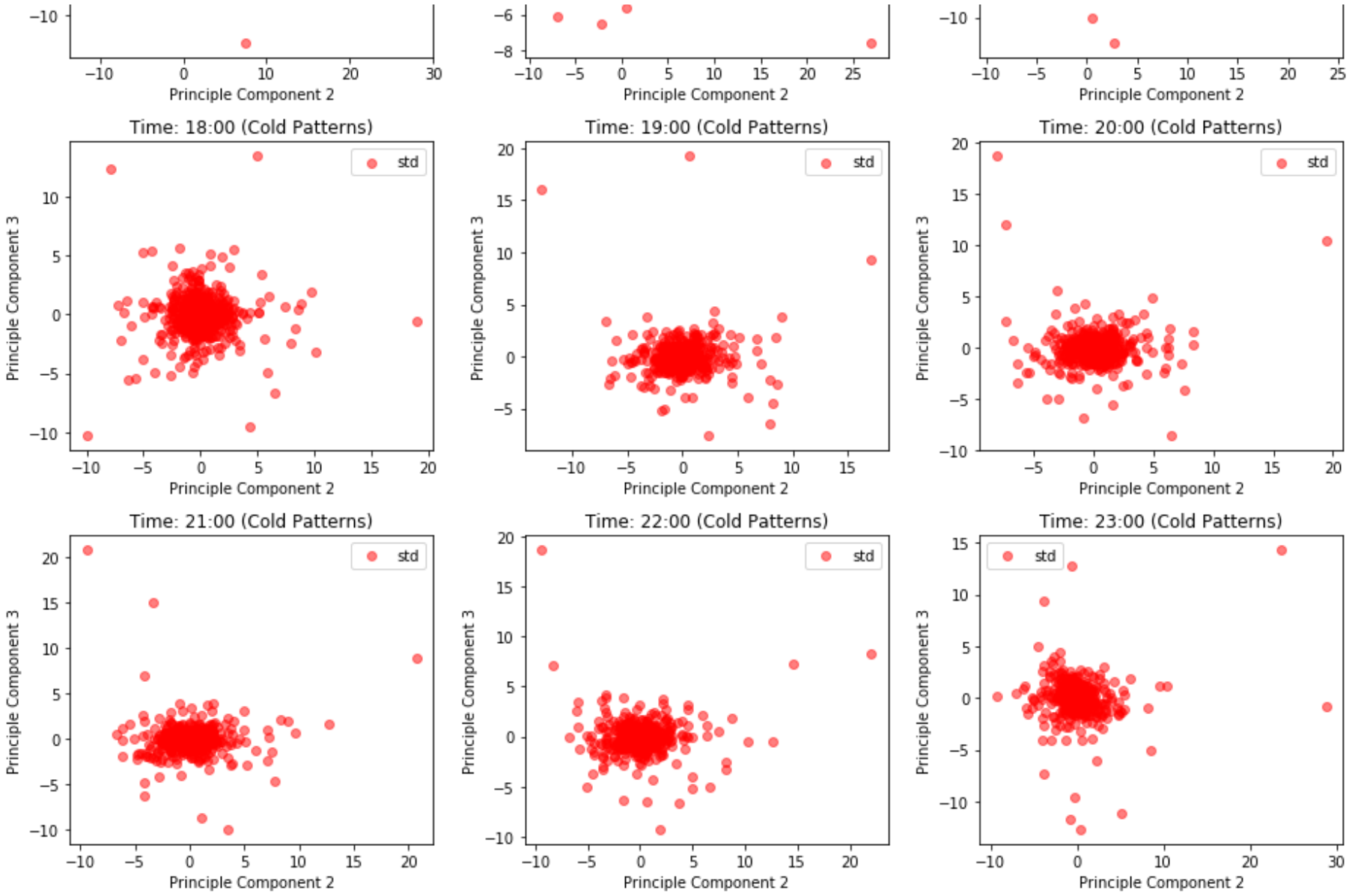

Time: 13:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2


Time: 14:00 (Cold Patterns)
std
Principle Component 3
Principle Component 2


Time: 15:00 (Cold Patterns)
std
Principle Component 3


Time: 16:00 (Cold Patterns)
std
Principle Component 3


Time: 17:00 (Cold Patterns)
std
Principle Component 3

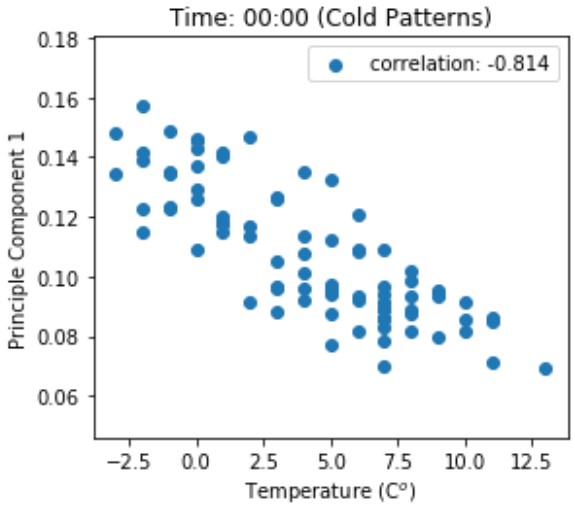


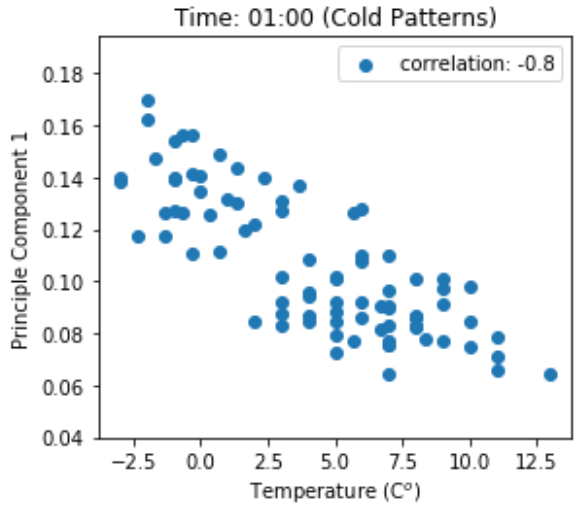
The above three types of plots show that std don't have significant advantage in this case, so we will use non-std in our later analysis.

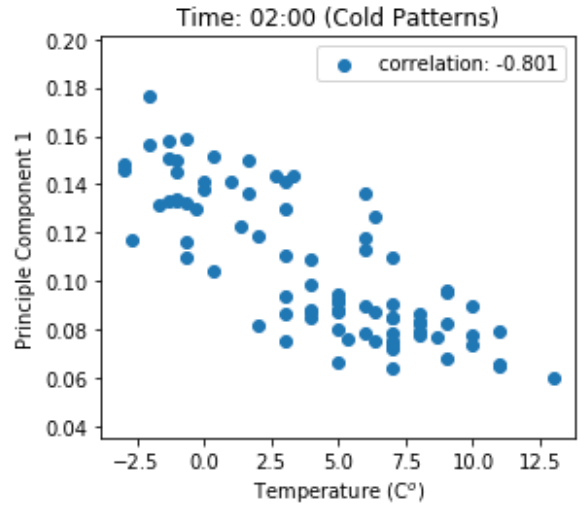
Let's try to interprete PCs' meaning:

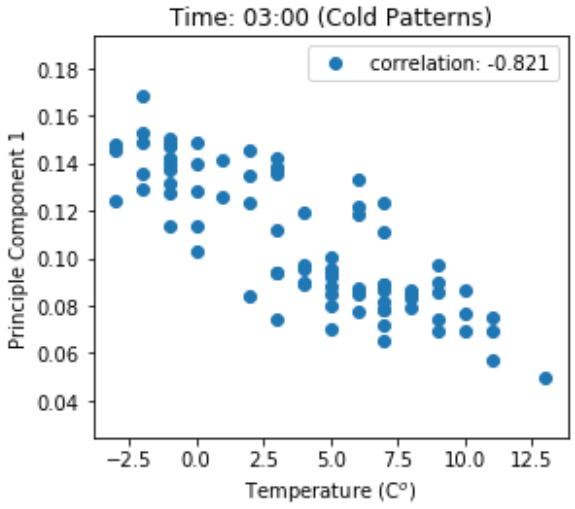
(1) PC1's coefficients vs Temperature:

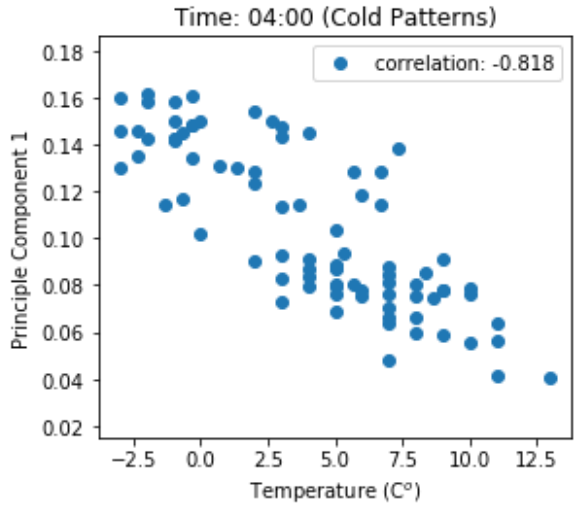
In [260]:
```
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(df_wea1h_nf_cold.TempC[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')], pca_ns
td.components_[0], label = 'correlation: ' + str(round(np.corrcoef(pca_nstd.components_[0], df_wea1h_nf_cold.
TempC[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')])[0][1], 3)))
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Principle Component 1')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(df_wea1h_nf_cold.TempC[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')], pca_nstd.com
ponents_[0], label = 'correlation: ' + str(round(np.corrcoef(pca_nstd.components_[0], df_wea1h_nf_cold.TempC[
df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')])[0][1], 3)))
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Principle Component 1')
plt.tight_layout()
```

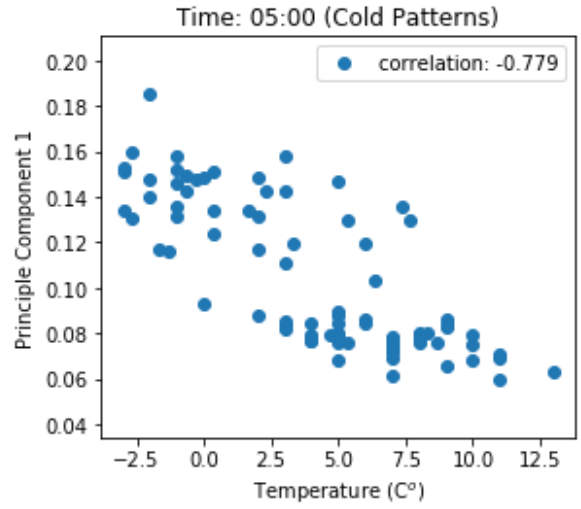

Time: 00:00 (Cold Patterns)
correlation: -0.814
Principle Component 1
Temperature (C°)

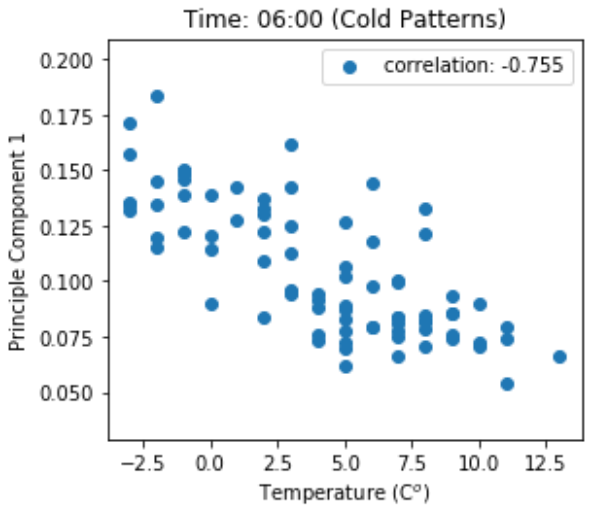

Time: 01:00 (Cold Patterns)
correlation: -0.8
Principle Component 1
Temperature (C°)

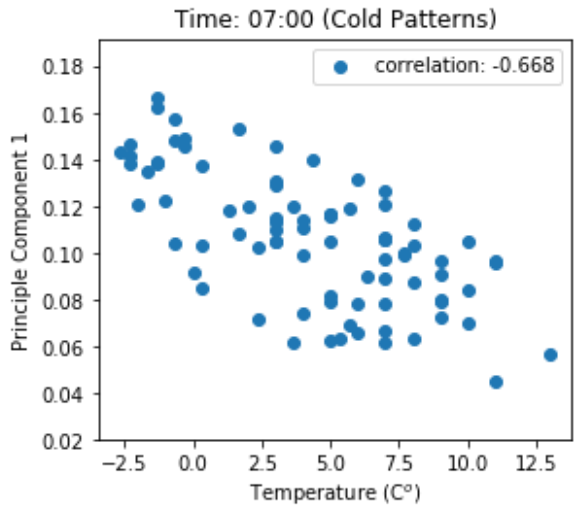

Time: 02:00 (Cold Patterns)
correlation: -0.801
Principle Component 1
Temperature (C°)

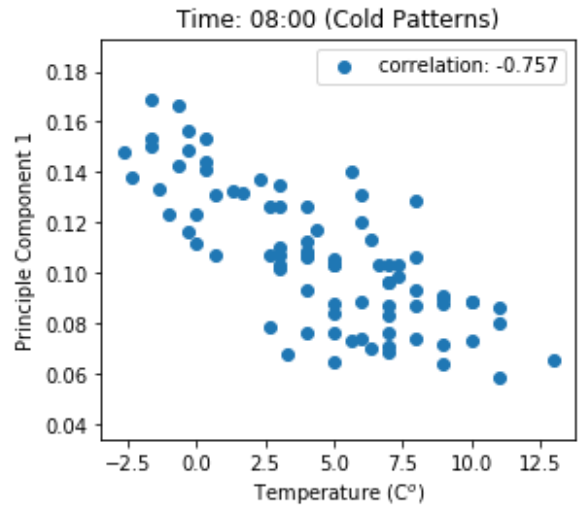

Time: 03:00 (Cold Patterns)
correlation: -0.821
Principle Component 1
Temperature (C°)

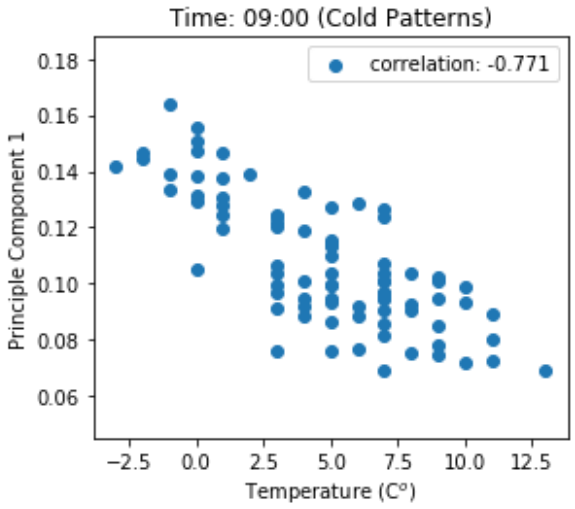

Time: 04:00 (Cold Patterns)
correlation: -0.818
Principle Component 1
Temperature (C°)

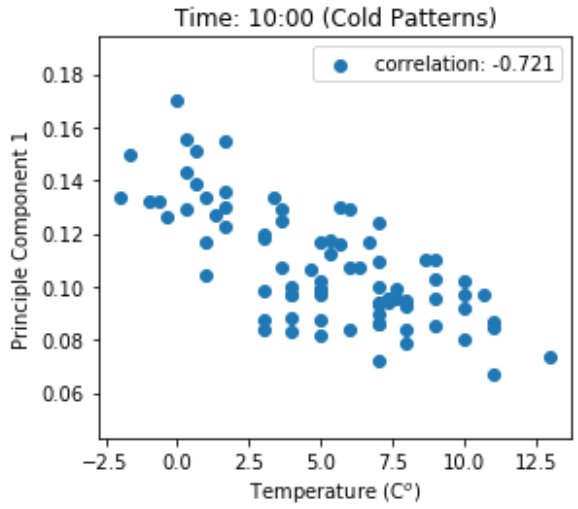

Time: 05:00 (Cold Patterns)
correlation: -0.779
Principle Component 1
Temperature (C°)

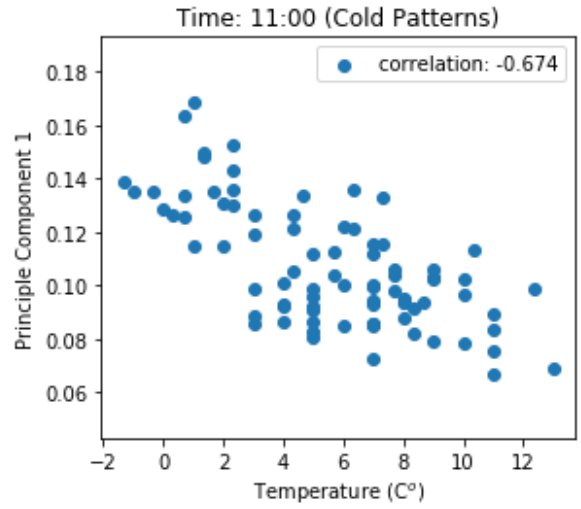

Time: 06:00 (Cold Patterns)
correlation: -0.755
Principle Component 1
Temperature (C°)

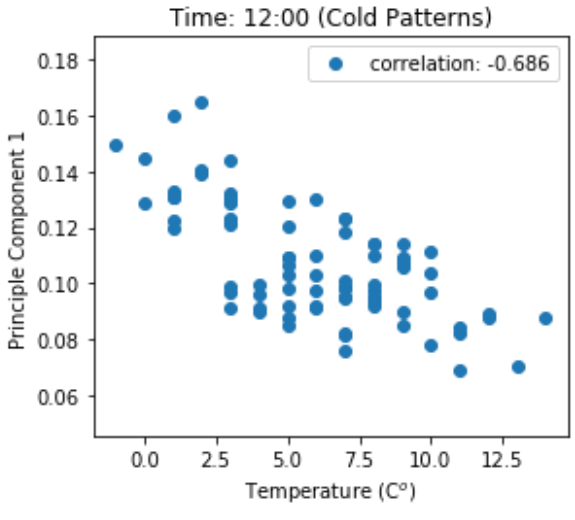

Time: 07:00 (Cold Patterns)
correlation: -0.668
Principle Component 1
Temperature (C°)

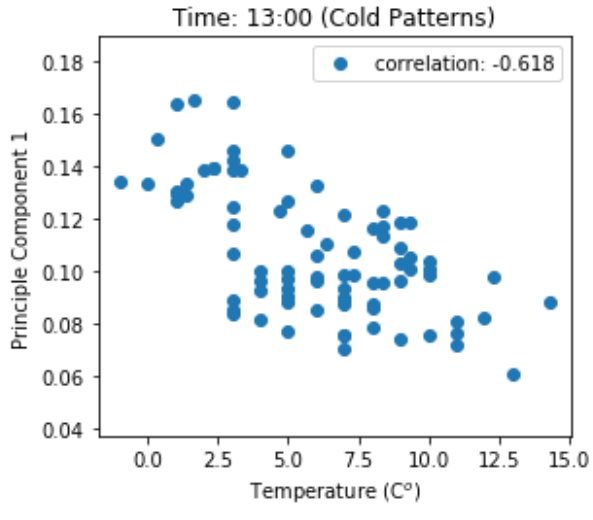

Time: 08:00 (Cold Patterns)
correlation: -0.757
Principle Component 1
Temperature (C°)

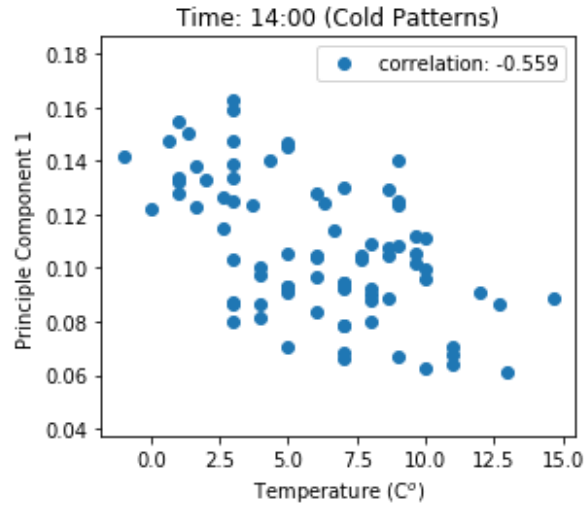

Time: 09:00 (Cold Patterns)
correlation: -0.771
Principle Component 1
Temperature (C°)


Time: 10:00 (Cold Patterns)
correlation: -0.721
Principle Component 1
Temperature (C°)


Time: 11:00 (Cold Patterns)
correlation: -0.674
Principle Component 1
Temperature (C°)


Time: 12:00 (Cold Patterns)
correlation: -0.686
Principle Component 1
Temperature (C°)

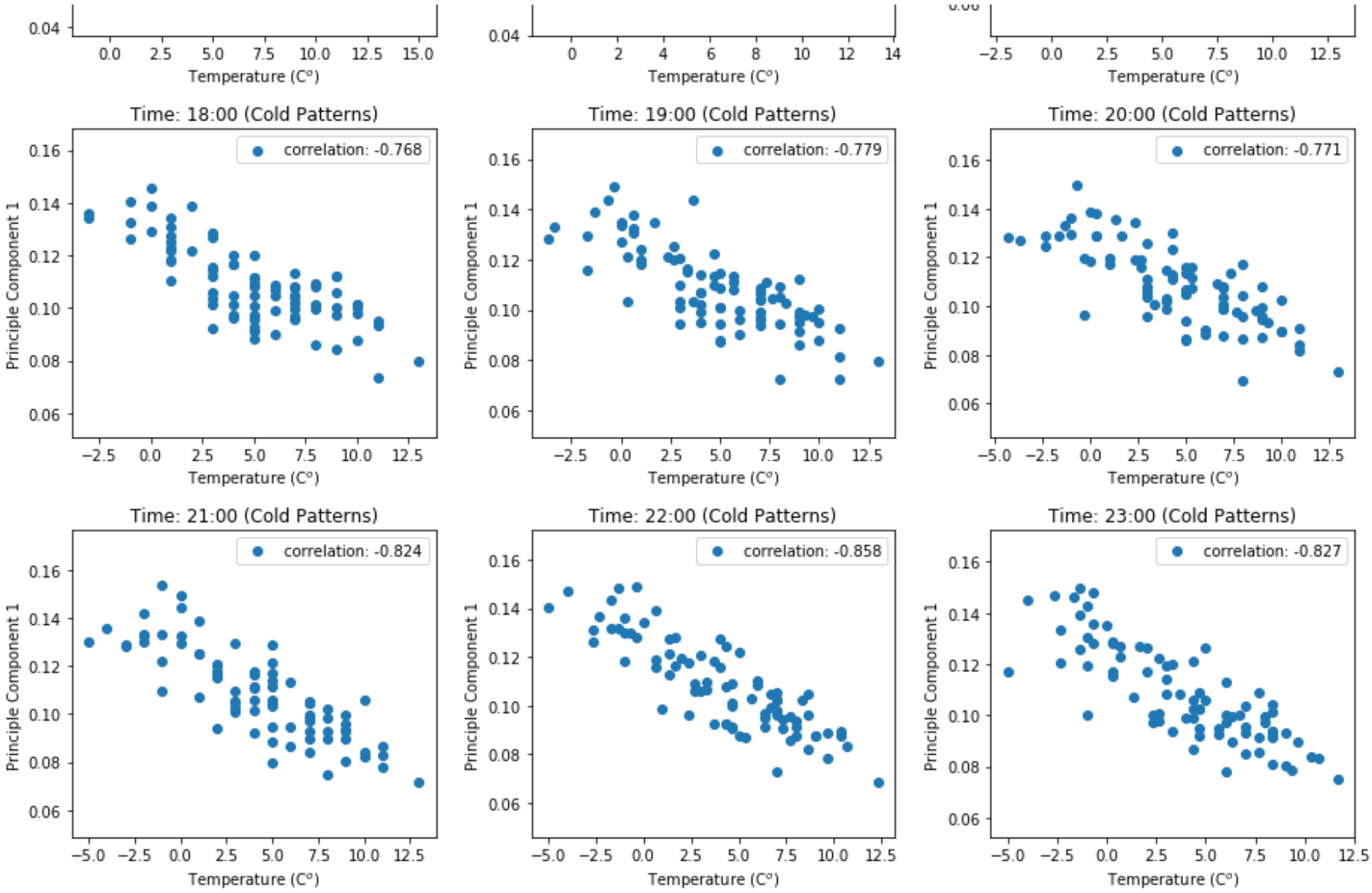

Time: 13:00 (Cold Patterns)
correlation: -0.618
Principle Component 1
Temperature (C°)


Time: 14:00 (Cold Patterns)
correlation: -0.559
Principle Component 1
Temperature (C°)

Time: 15:00 (Cold Patterns)
correlation: -0.62
Principle Component 1

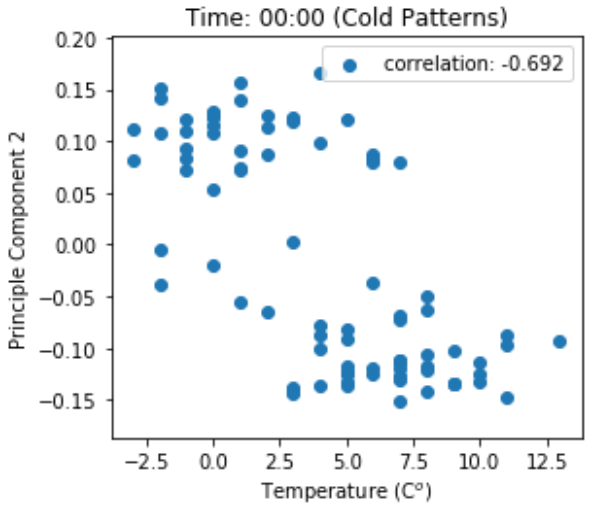
Time: 16:00 (Cold Patterns)
correlation: -0.733
Principle Component 1

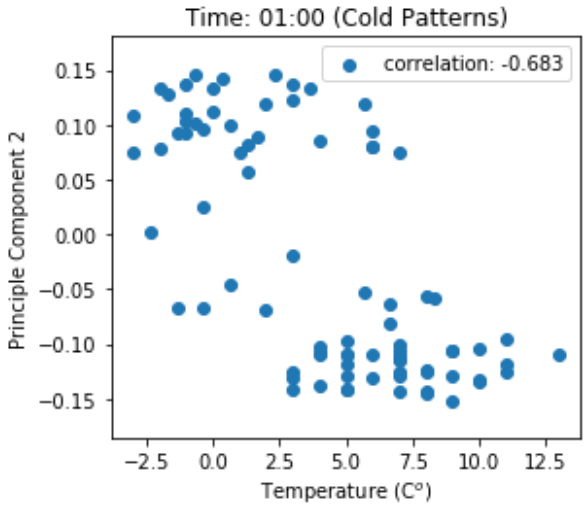
Time: 17:00 (Cold Patterns)
correlation: -0.748
Principle Component 1

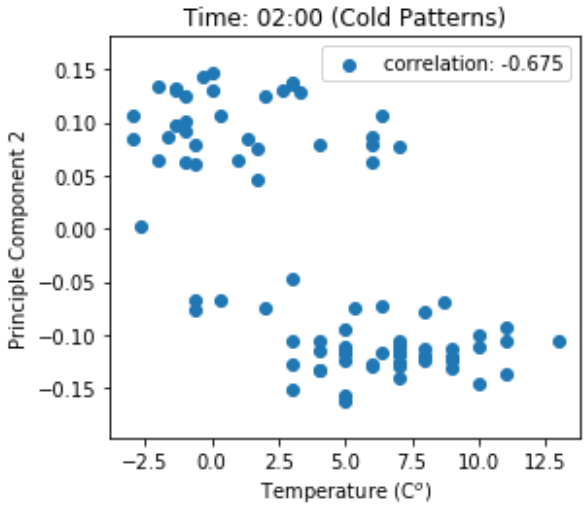


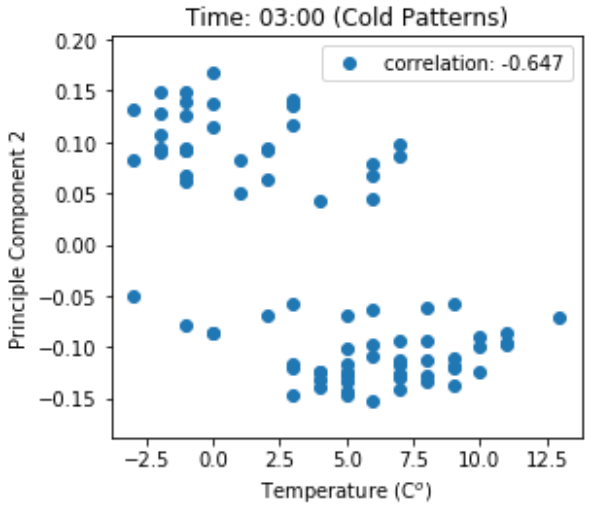
2) PC2's coefficients vs Temperature:

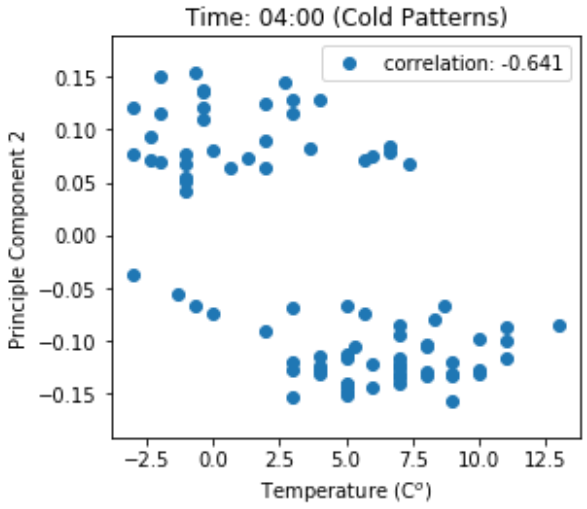
In [234]:
```
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(df_wea1h_nf_cold.TempC[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')], pca_ns
td.components_[1], label = 'correlation: ' + str(round(np.corrcoef(pca_nstd.components_[1], df_wea1h_nf_cold.
TempC[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')])[0][1], 3)))
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Principle Component 2')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(df_wea1h_nf_cold.TempC[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')], pca_nstd.com
ponents_[1], label = 'correlation: ' + str(round(np.corrcoef(pca_nstd.components_[1], df_wea1h_nf_cold.TempC[
df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')])[0][1], 3)))
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Principle Component 2')
plt.tight_layout()
```

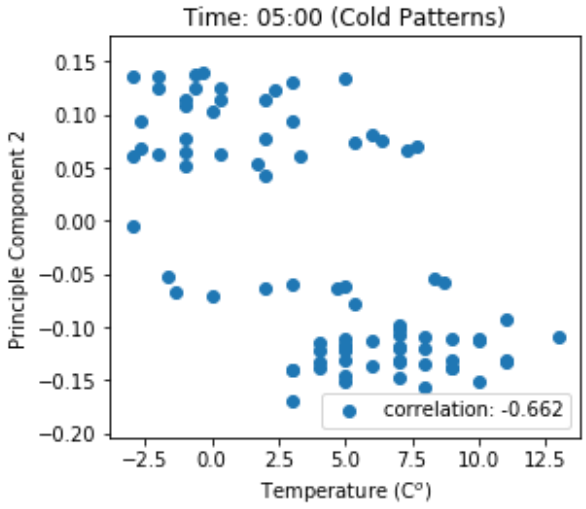

Time: 00:00 (Cold Patterns)
correlation: -0.692
Principle Component 2
Temperature (C°)

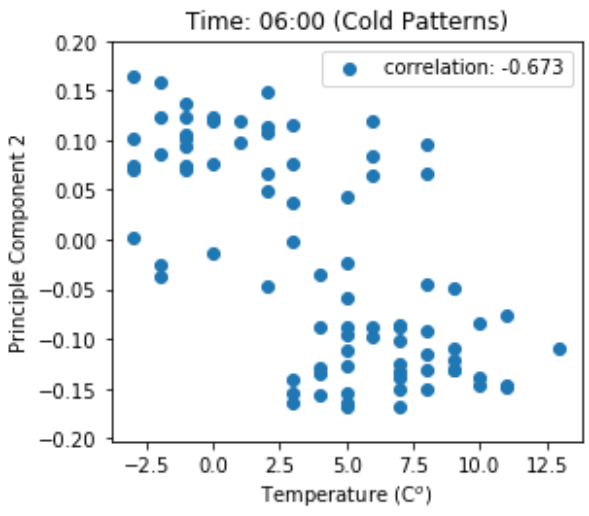

Time: 01:00 (Cold Patterns)
correlation: -0.683
Principle Component 2
Temperature (C°)

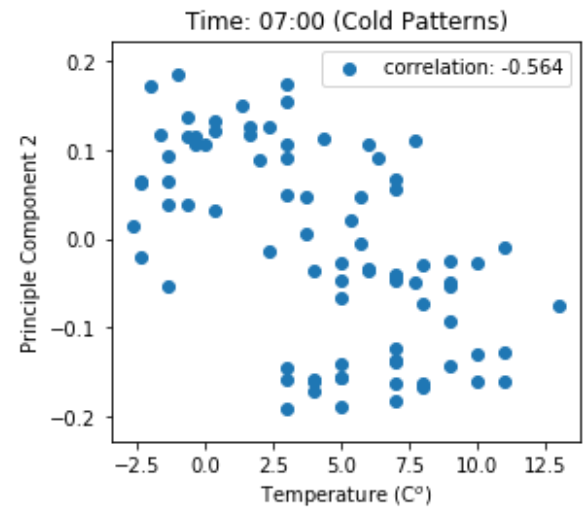

Time: 02:00 (Cold Patterns)
correlation: -0.675
Principle Component 2
Temperature (C°)

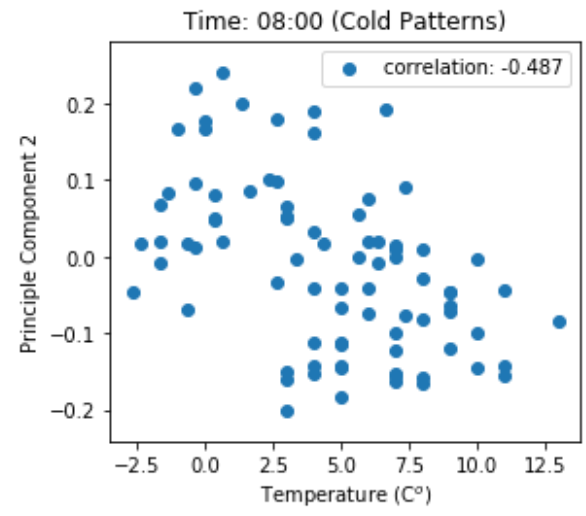

Time: 03:00 (Cold Patterns)
correlation: -0.647
Principle Component 2
Temperature (C°)

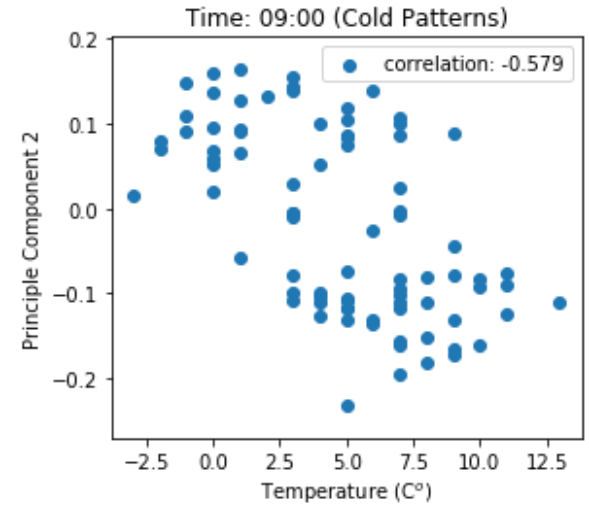

Time: 04:00 (Cold Patterns)
correlation: -0.641
Principle Component 2
Temperature (C°)

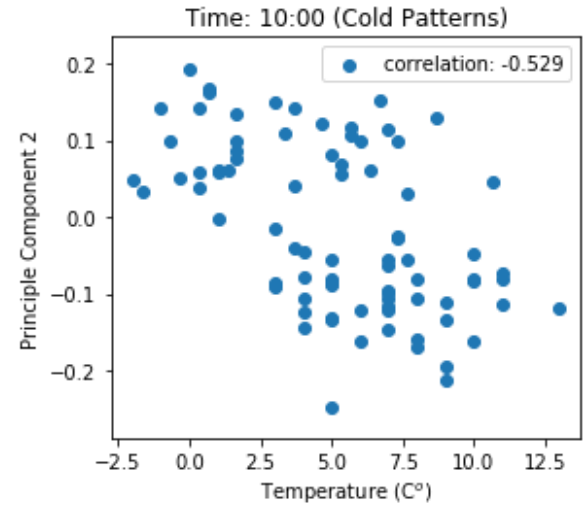

Time: 05:00 (Cold Patterns)
correlation: -0.662
Principle Component 2
Temperature (C°)

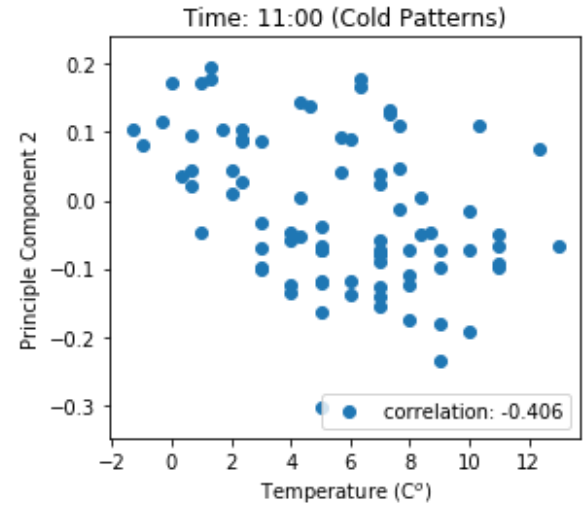

Time: 06:00 (Cold Patterns)
correlation: -0.673
Principle Component 2
Temperature (C°)

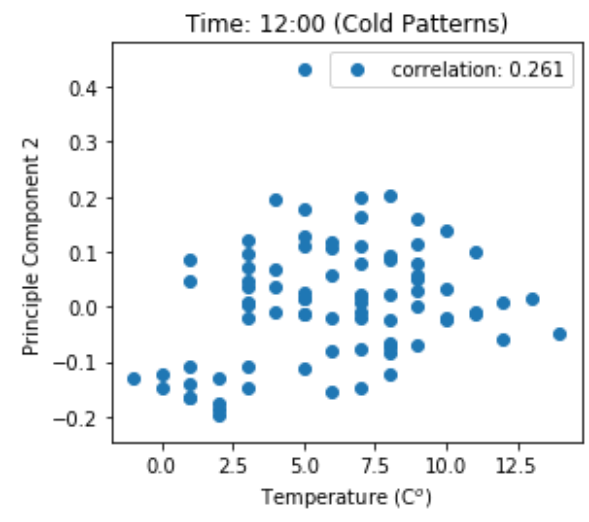

Time: 07:00 (Cold Patterns)
correlation: -0.564
Principle Component 2
Temperature (C°)

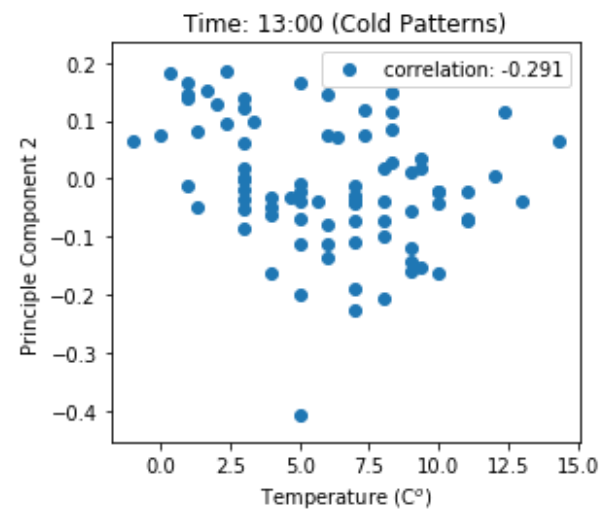

Time: 08:00 (Cold Patterns)
correlation: -0.487
Principle Component 2
Temperature (C°)

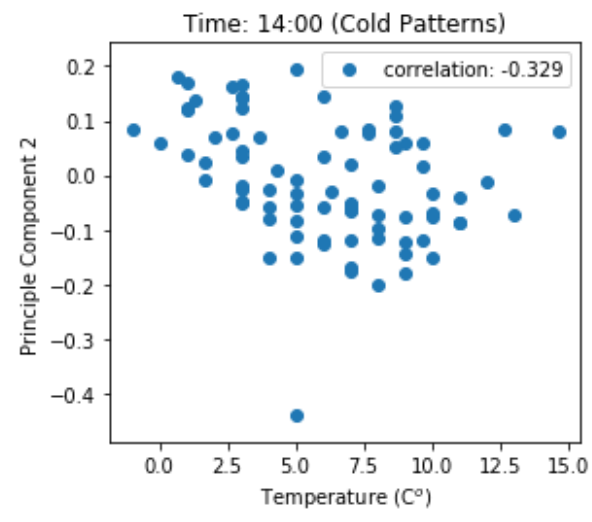

Time: 09:00 (Cold Patterns)
correlation: -0.579
Principle Component 2
Temperature (C°)

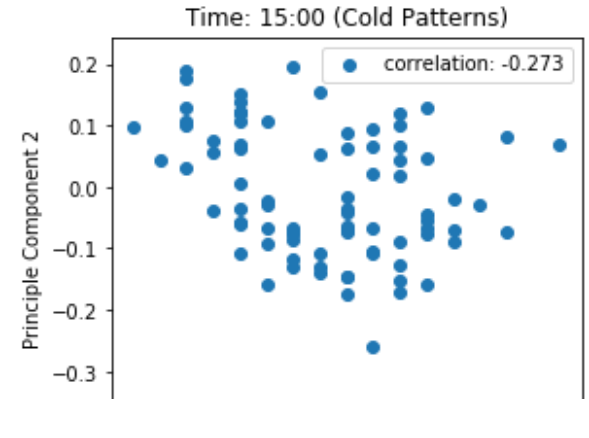

Time: 10:00 (Cold Patterns)
correlation: -0.529
Principle Component 2
Temperature (C°)

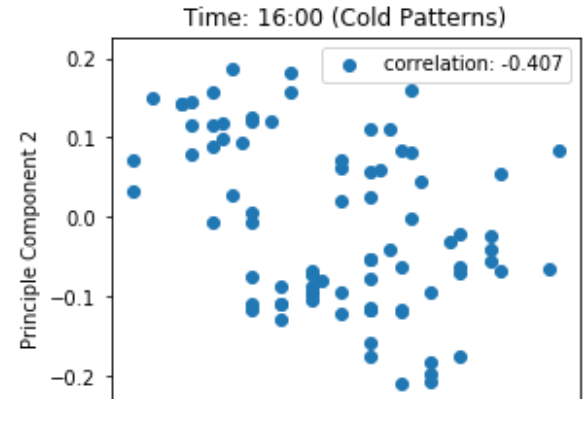

Time: 11:00 (Cold Patterns)
correlation: -0.406
Principle Component 2
Temperature (C°)

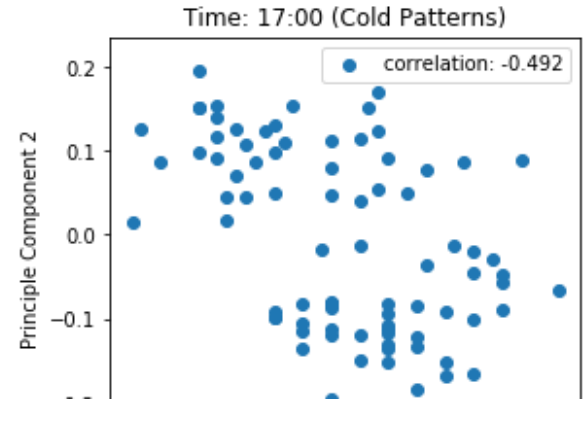

Time: 12:00 (Cold Patterns)
correlation: 0.261
Principle Component 2
Temperature (C°)

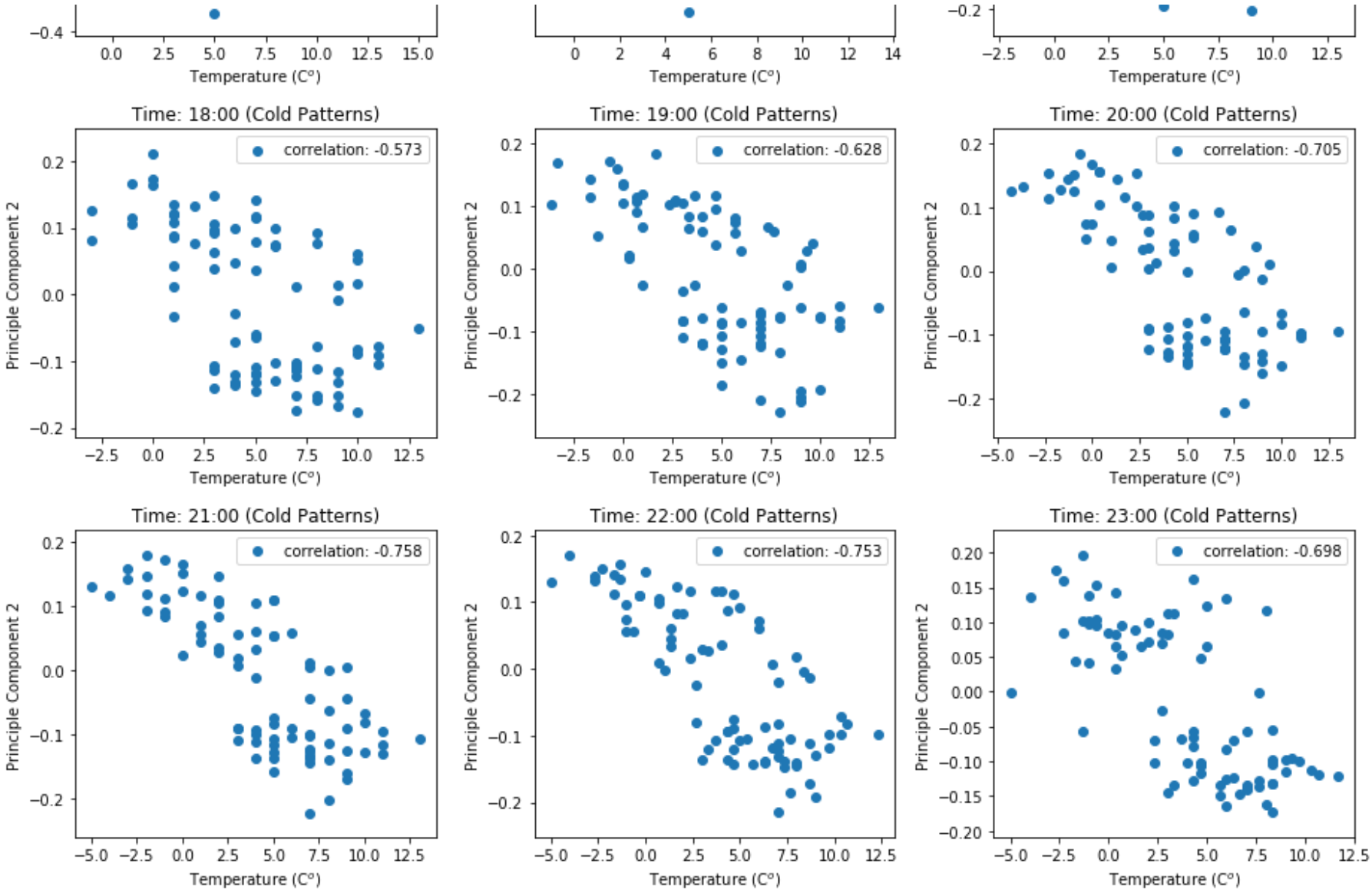

Time: 13:00 (Cold Patterns)
correlation: -0.291
Principle Component 2
Temperature (C°)


Time: 14:00 (Cold Patterns)
correlation: -0.329
Principle Component 2
Temperature (C°)


Time: 15:00 (Cold Patterns)
correlation: -0.273
Principle Component 2


Time: 16:00 (Cold Patterns)
correlation: -0.407
Principle Component 2

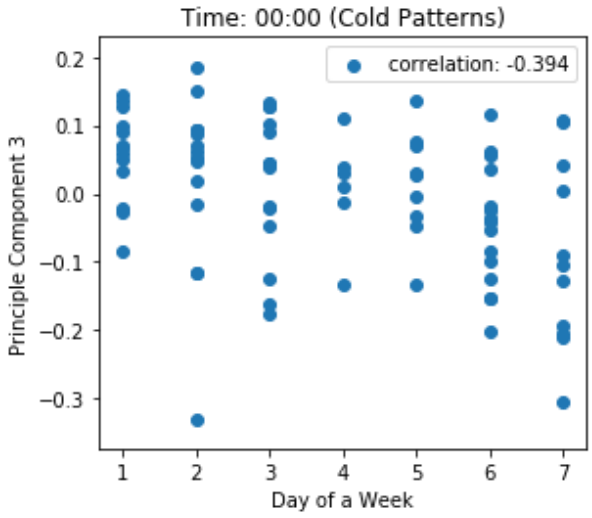

Time: 17:00 (Cold Patterns)
correlation: -0.492
Principle Component 2

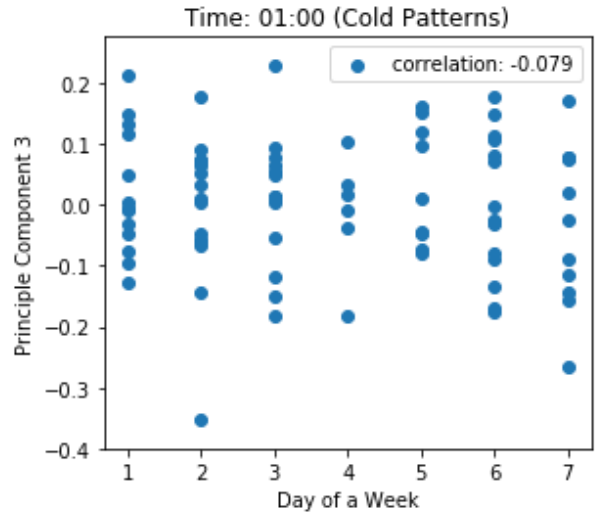


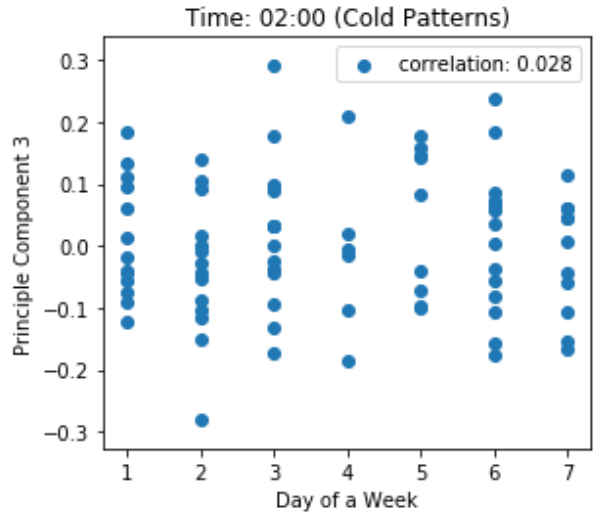
The above figures show that PC2 still has certain amount of correlation with temperature but not as significant as PC1 and temperature have.

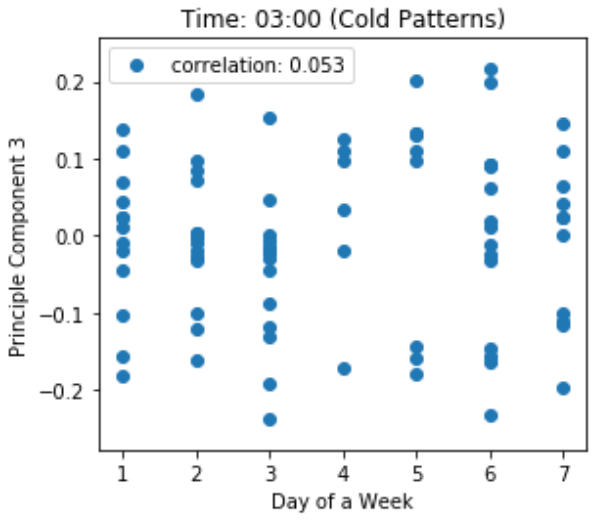
3) PC3's coefficients vs Day of a Week:

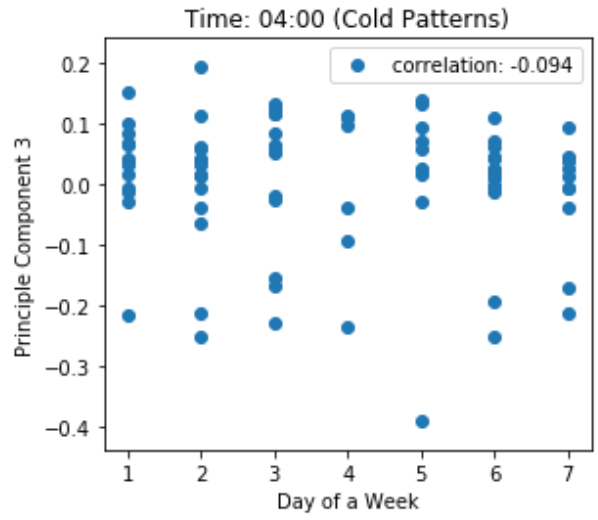
In [233]:
```
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(dayofWeek + 1, pca_nstd.components_[2], label = 'correlation: ' + str(round(np.corrcoef(pc
a_nstd.components_[2], dayofWeek.GMT)[0][1], 3)))
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Day of a Week')
        ax.set_ylabel('Principle Component 3')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        ax.scatter(dayofWeek + 1, pca_nstd.components_[2], label = 'correlation: ' + str(round(np.corrcoef(pc
a_nstd.components_[2], dayofWeek.GMT)[0][1], 3)))
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel('Day of a Week')
        ax.set_ylabel('Principle Component 3')
plt.tight_layout()
```

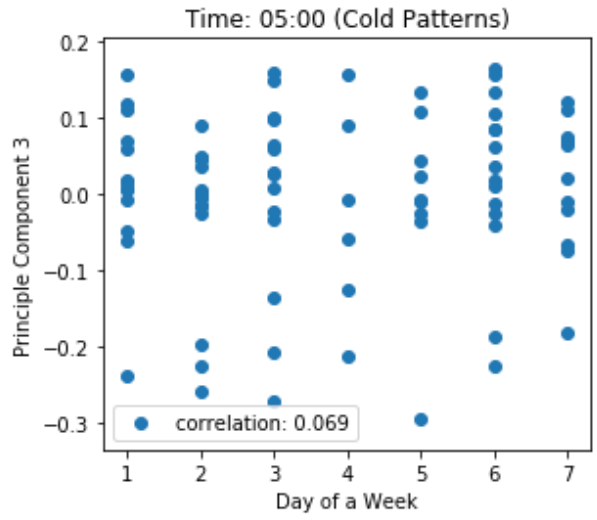

Time: 00:00 (Cold Patterns)
correlation: -0.394
Principle Component 3
Day of a Week

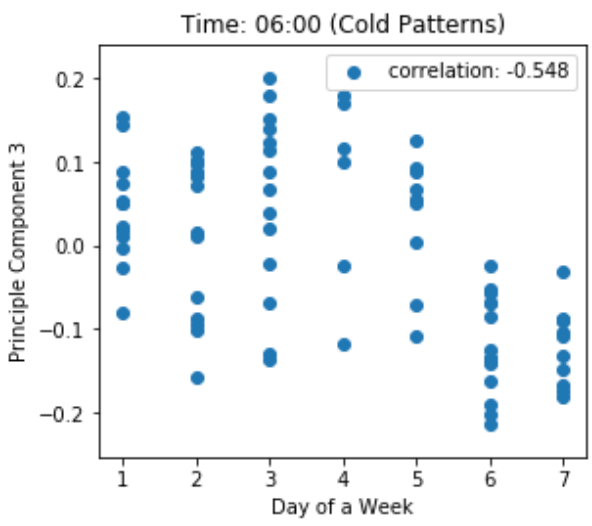

Time: 01:00 (Cold Patterns)
correlation: -0.079
Principle Component 3
Day of a Week

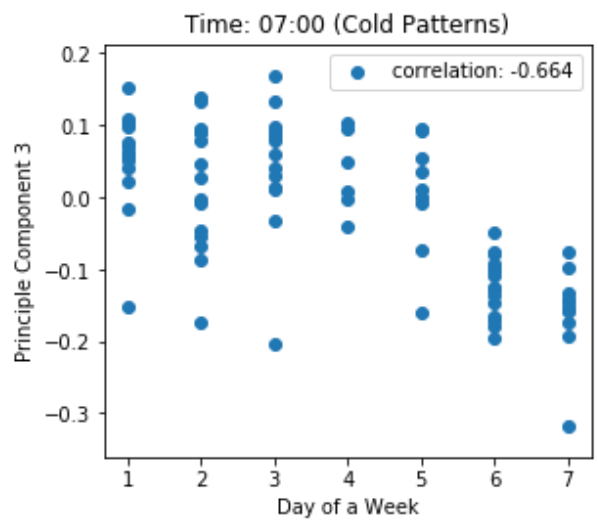

Time: 02:00 (Cold Patterns)
correlation: 0.028
Principle Component 3
Day of a Week

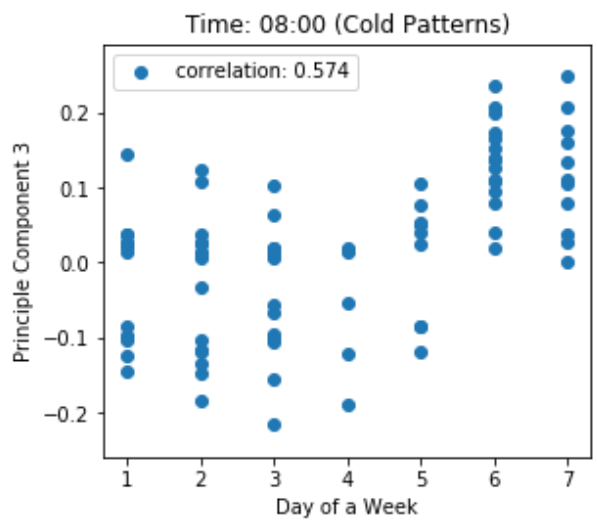

Time: 03:00 (Cold Patterns)
correlation: 0.053
Principle Component 3
Day of a Week

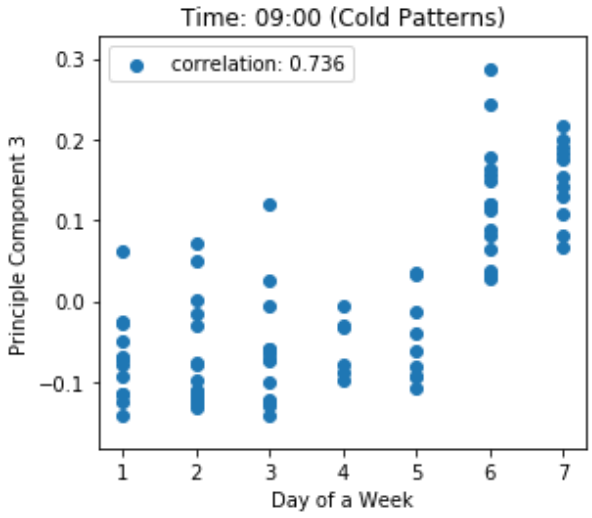

Time: 04:00 (Cold Patterns)
correlation: -0.094
Principle Component 3
Day of a Week

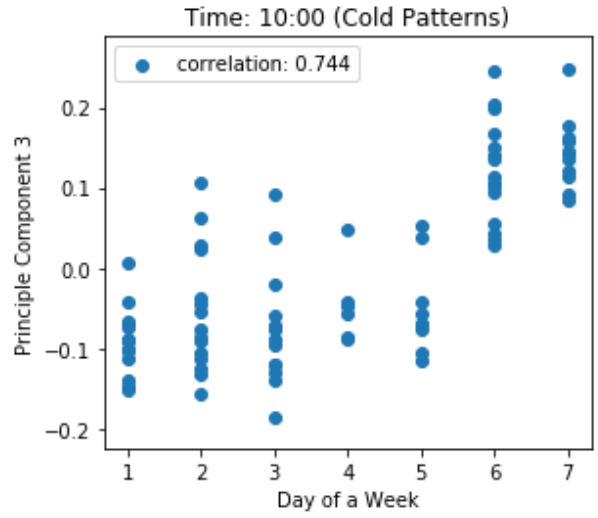

Time: 05:00 (Cold Patterns)
correlation: 0.069
Principle Component 3
Day of a Week

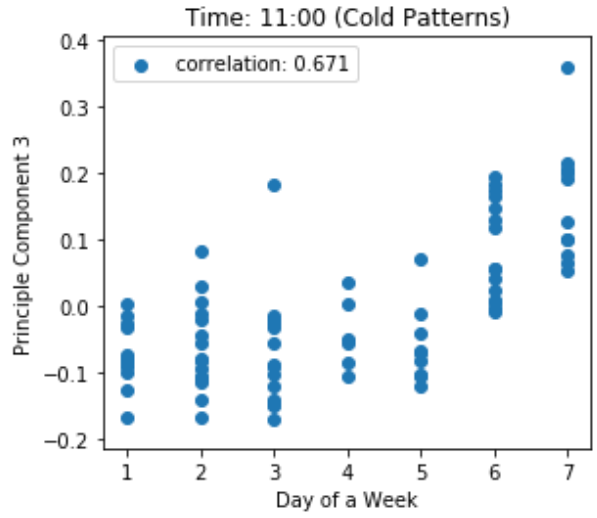

Time: 06:00 (Cold Patterns)
correlation: -0.548
Principle Component 3
Day of a Week

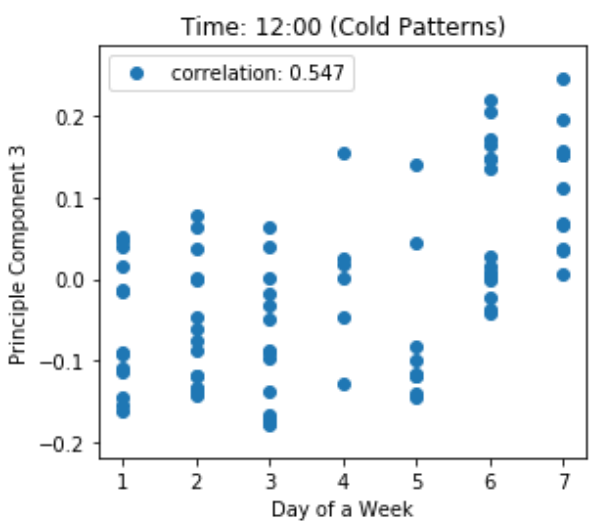

Time: 07:00 (Cold Patterns)
correlation: -0.664
Principle Component 3
Day of a Week

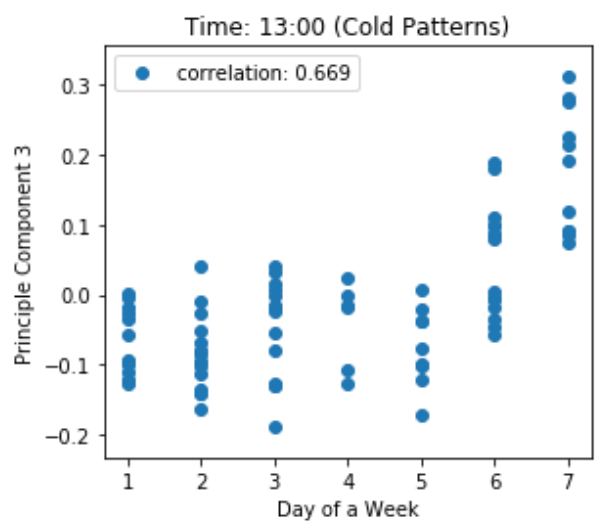

Time: 08:00 (Cold Patterns)
correlation: 0.574
Principle Component 3
Day of a Week

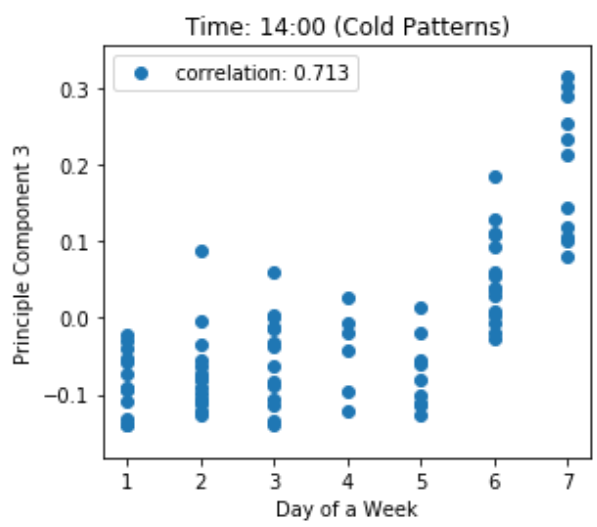

Time: 09:00 (Cold Patterns)
correlation: 0.736
Principle Component 3
Day of a Week

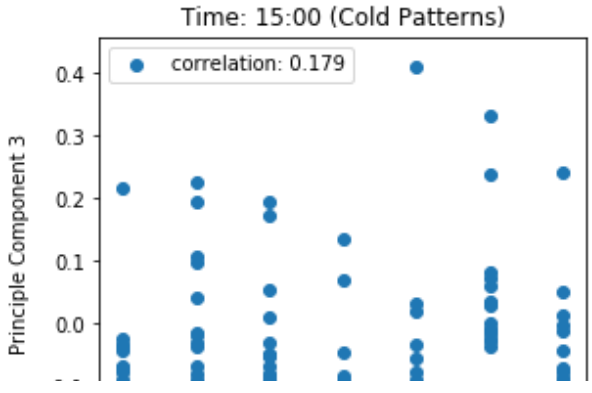

Time: 10:00 (Cold Patterns)
correlation: 0.744
Principle Component 3
Day of a Week

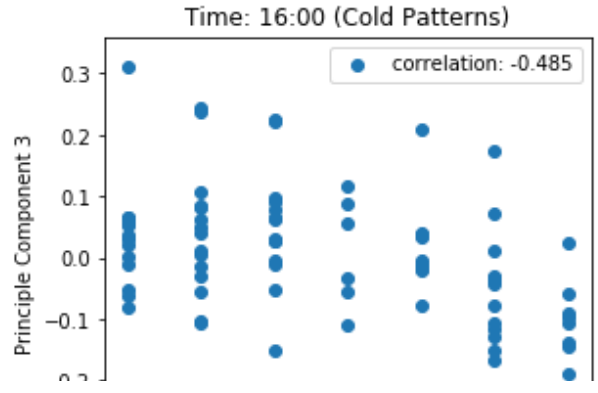

Time: 11:00 (Cold Patterns)
correlation: 0.671
Principle Component 3
Day of a Week

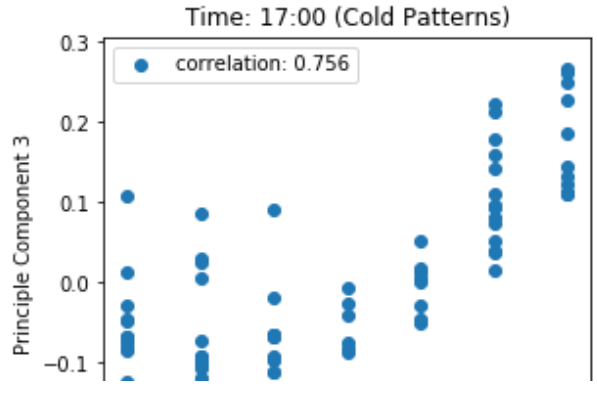

Time: 12:00 (Cold Patterns)
correlation: 0.547
Principle Component 3
Day of a Week

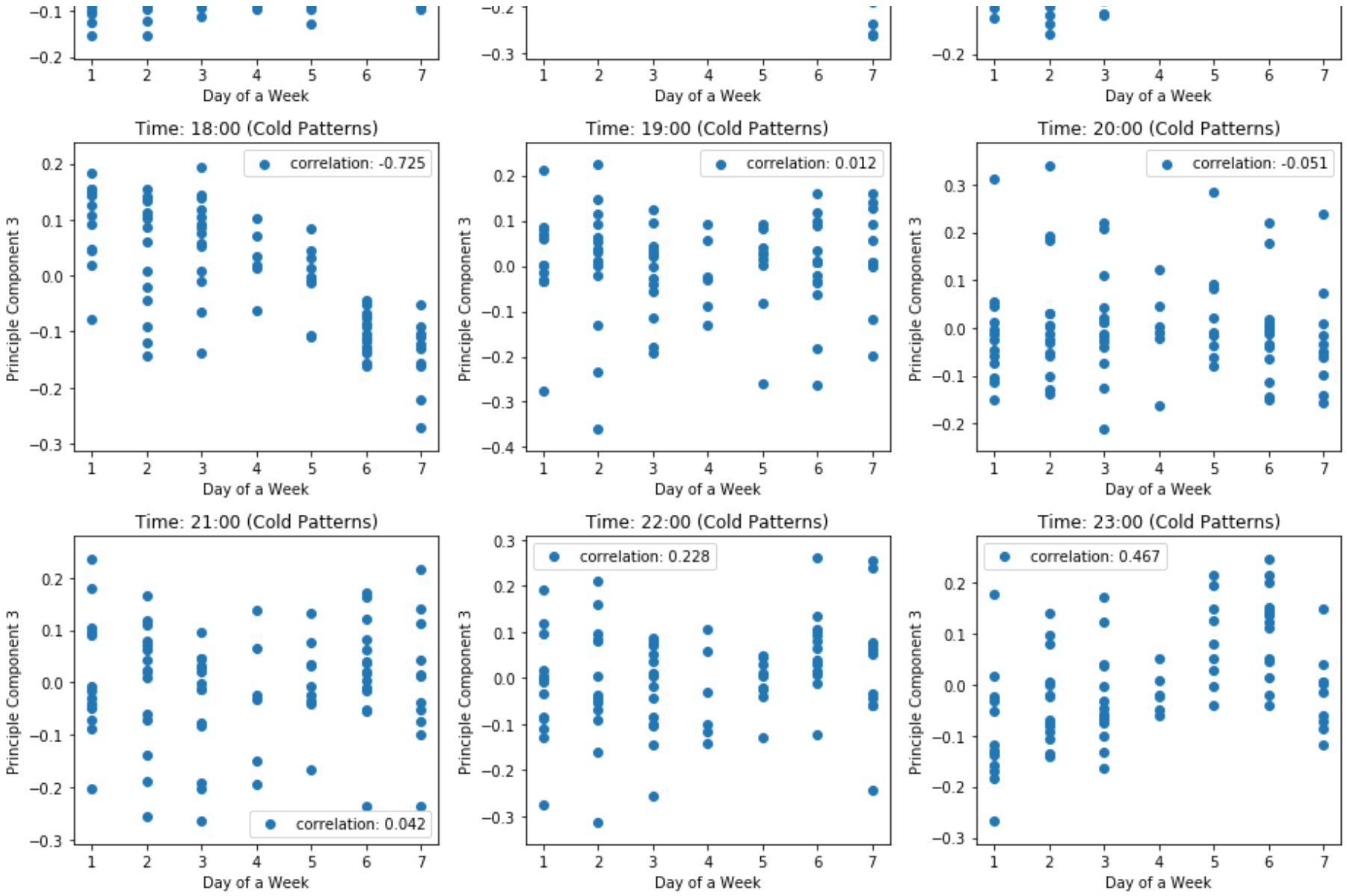

Time: 13:00 (Cold Patterns)
correlation: 0.669
Principle Component 3
Day of a Week


Time: 14:00 (Cold Patterns)
correlation: 0.713
Principle Component 3
Day of a Week


Time: 15:00 (Cold Patterns)
correlation: 0.179
Principle Component 3


Time: 16:00 (Cold Patterns)
correlation: -0.485
Principle Component 3


Time: 17:00 (Cold Patterns)
correlation: 0.756
Principle Component 3



2. K-mean clustering

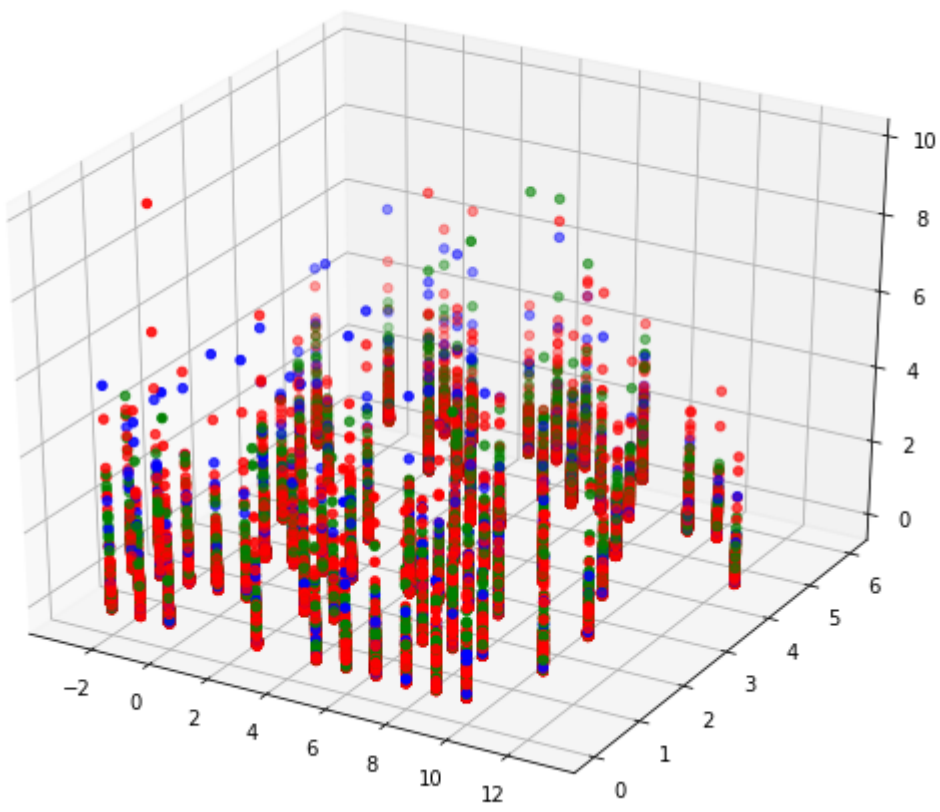
Milestone 4: K-mean clustering

The first three PCs are selected as inputs for K-mean clustering, due to their significance and interpretability

```
In [413]: t2.columns[-1]
```

```
Out[413]: 'DayofWeek'
```

In [431]:
```
x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]
t = x.dropna(axis = 'columns')
t2 = t[t.columns[np.insert(kmeans.labels_ == 0, 0, True)]].copy() # select all data from the n-th cluster
t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['TempC'].values) # add
temperature column to dataframe
t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['GMT']).dt.d
ayofweek # add day of a week column to dataframe
t2
fig_all = plt.figure(figsize = (10,8))
ax = fig_all.add_subplot(1, 1, 1, projection = '3d')
for i in range(1, 1 + 528):
    ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[i]], color = 'red')

t2 = t[t.columns[np.insert(kmeans.labels_ == 1, 0, True)]].copy() # select all data from the n-th cluster
t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['TempC'].values) # add
temperature column to dataframe
t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['GMT']).dt.d
ayofweek # add day of a week column to dataframe
for i in range(1, 1 + 107):
    ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[i]], color = 'green')

t2 = t[t.columns[np.insert(kmeans.labels_ == 2, 0, True)]].copy() # select all data from the n-th cluster
t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['TempC'].values) # add
temperature column to dataframe
t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['GMT']).dt.d
ayofweek # add day of a week column to dataframe
for i in range(1, 1 + 25):
    ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[i]], color = 'blue')
```

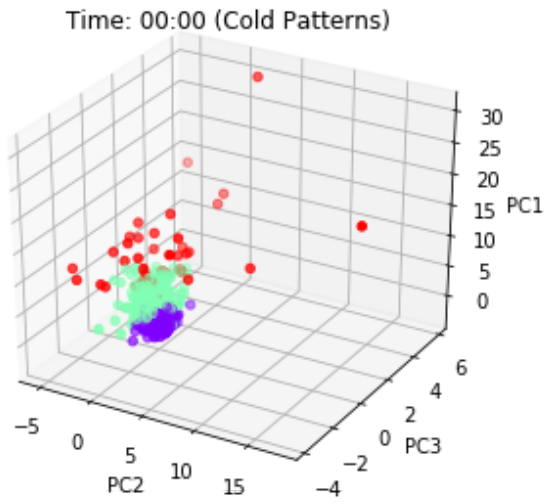


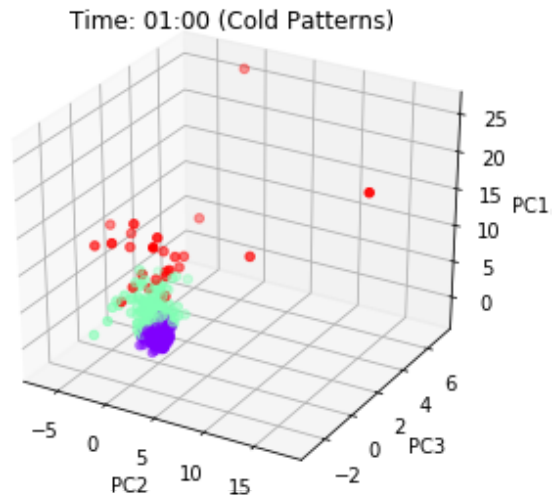
In [424]:
```
# the first groups consumption dataframe
(kmeans.labels_ == 2).sum()
```

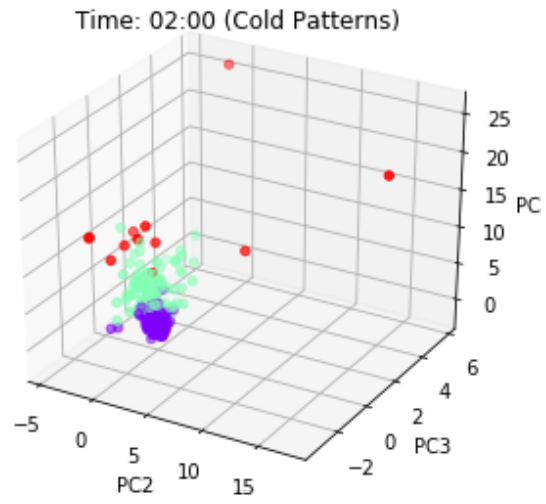
Out[424]: 25

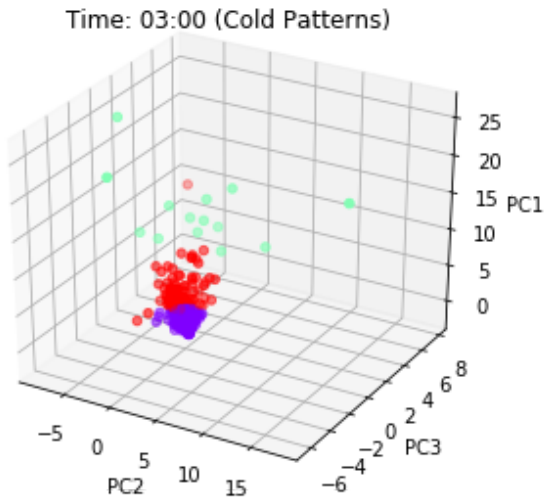
In [ ]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = 3).fit(principleDf_nstd.iloc[:,:3])
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10), cm
ap = 'rainbow')
        # what is the center of the cluster
        kmeans.cluster_centers_
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.set_xlabel('PC2')
        ax.set_ylabel('PC3')
        ax.set_zlabel('PC1')
#     else:
#         x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
#         dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
#         x.set_index('GMT', inplace = True)
#         x = x.transpose().dropna() # data for the specific hour, and filtering out null value
#         # PCA without standardization
#         pca_nstd = PCA()
#         principleComponents_nstd = pca_nstd.fit_transform(x)
#         principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
#         kmeans = KMeans(n_clusters = 3).fit(principleDf_nstd.iloc[:,:3])
#         ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10),
 cmap = 'rainbow')
#         ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
#         ax.set_xlabel('PC2')
#         ax.set_ylabel('PC3')
#         ax.set_zlabel('PC1')
plt.tight_layout()
```

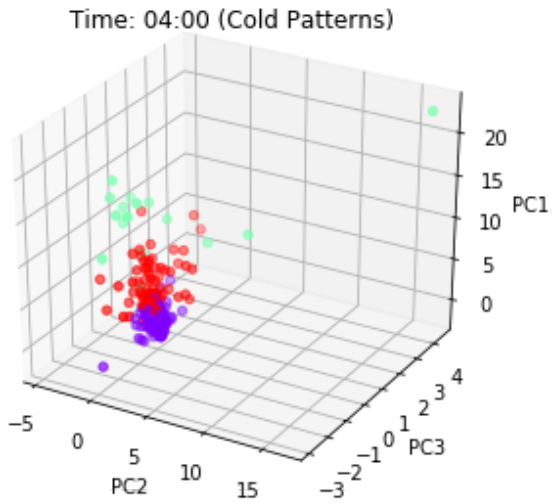
In [297]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = 3).fit(principleDf_nstd.iloc[:,:3])
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10), cm
ap = 'rainbow')
        # what is the center of the cluster
        kmeans.cluster_centers_
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.set_xlabel('PC2')
        ax.set_ylabel('PC3')
        ax.set_zlabel('PC1')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = 3).fit(principleDf_nstd.iloc[:,:3])
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10), cm
ap = 'rainbow')
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.set_xlabel('PC2')
        ax.set_ylabel('PC3')
        ax.set_zlabel('PC1')
plt.tight_layout()
```

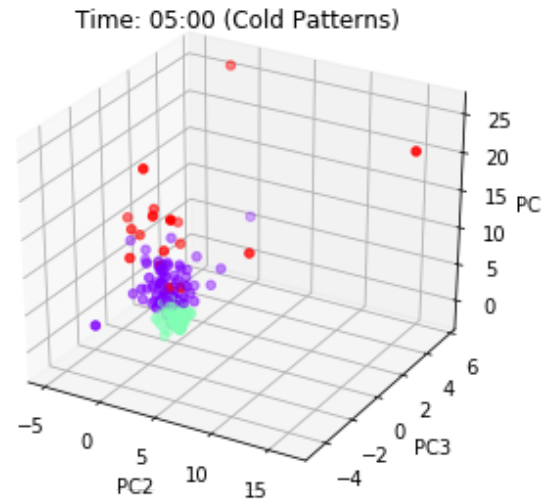

Time: 00:00 (Cold Patterns)
PC1
PC2
PC3

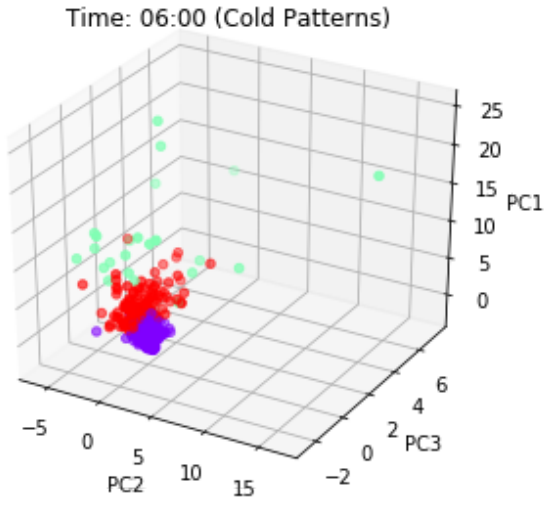

Time: 01:00 (Cold Patterns)
PC1
PC2
PC3

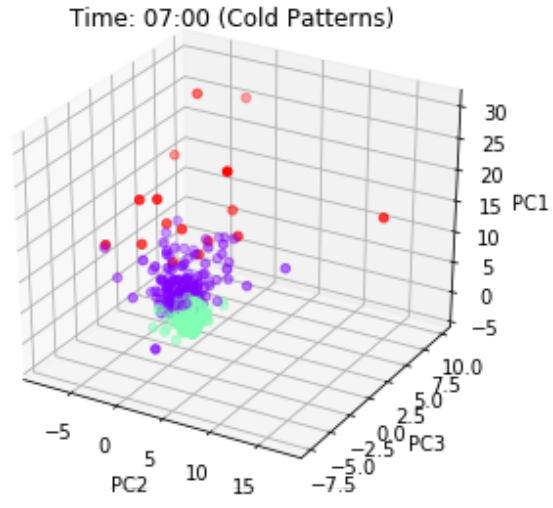

Time: 02:00 (Cold Patterns)
PC
PC2
PC3

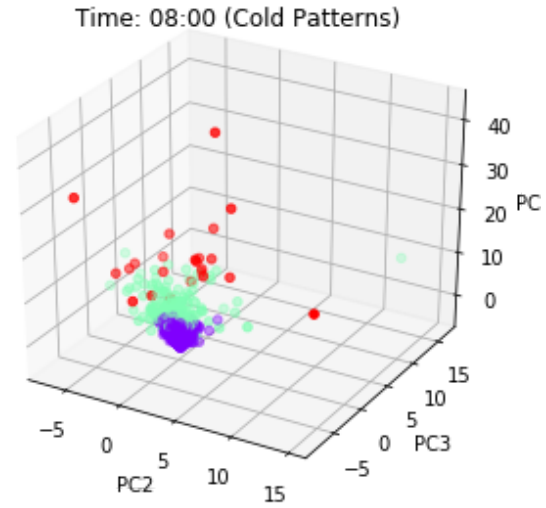

Time: 03:00 (Cold Patterns)
PC1
PC2
PC3

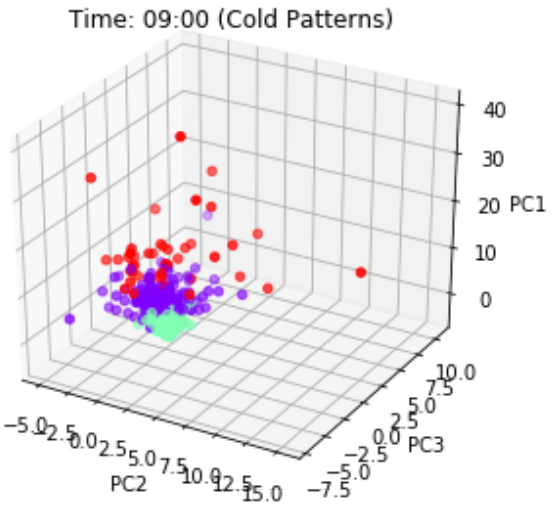

Time: 04:00 (Cold Patterns)
PC1
PC2
PC3

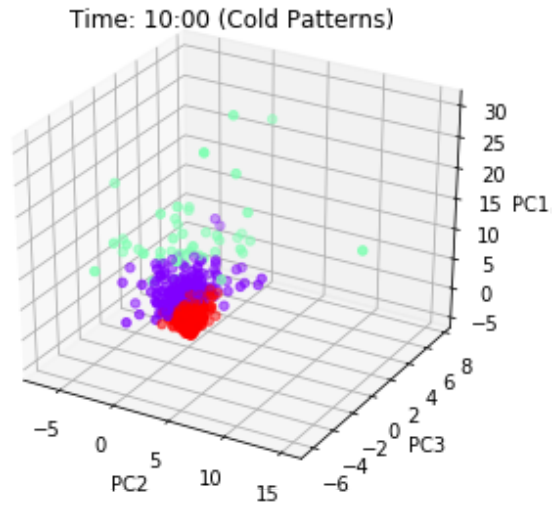

Time: 05:00 (Cold Patterns)
PC
PC2
PC3

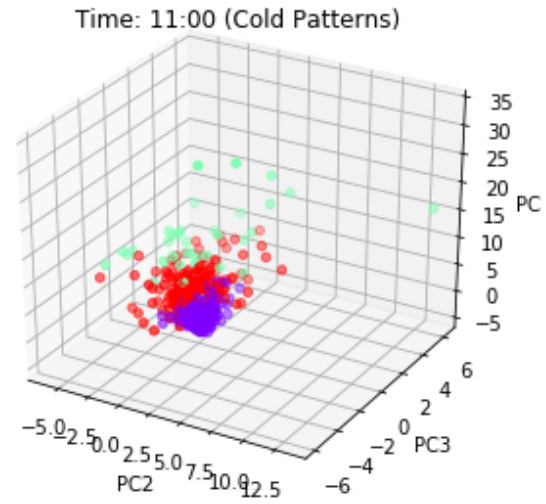

Time: 06:00 (Cold Patterns)
PC1
PC2
PC3

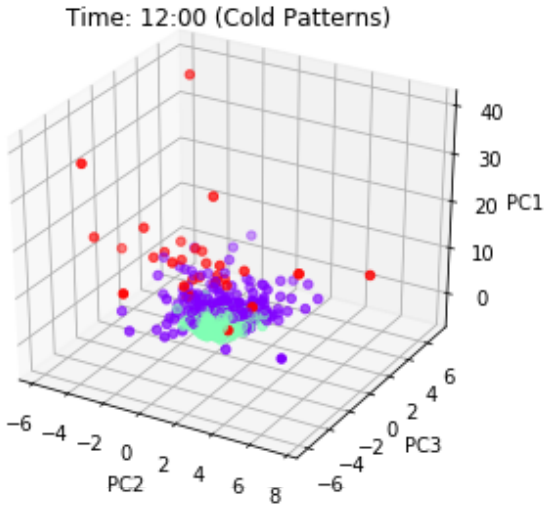

Time: 07:00 (Cold Patterns)
PC1
PC2
PC3

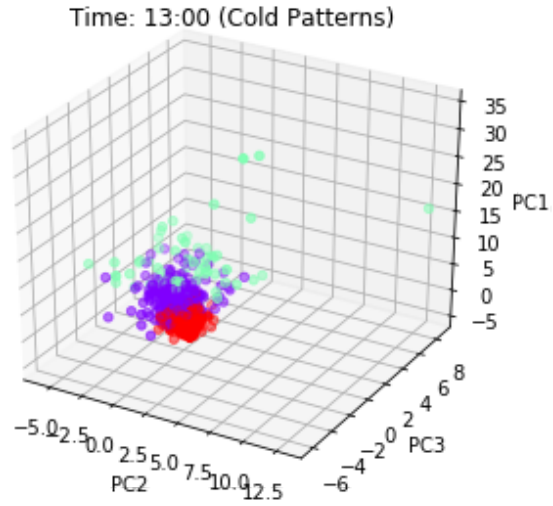

Time: 08:00 (Cold Patterns)
PC
PC2
PC3

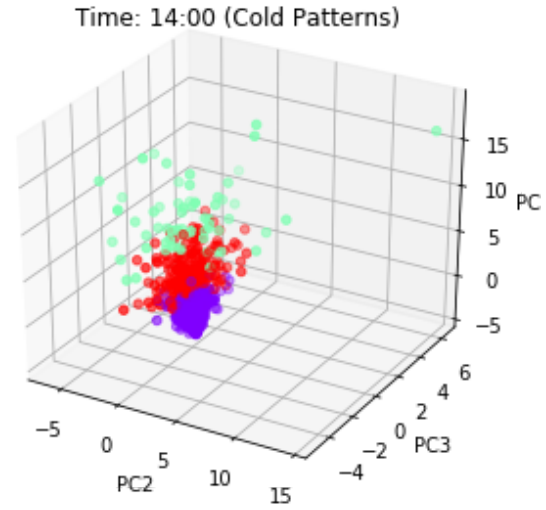

Time: 09:00 (Cold Patterns)
PC1
PC2
PC3

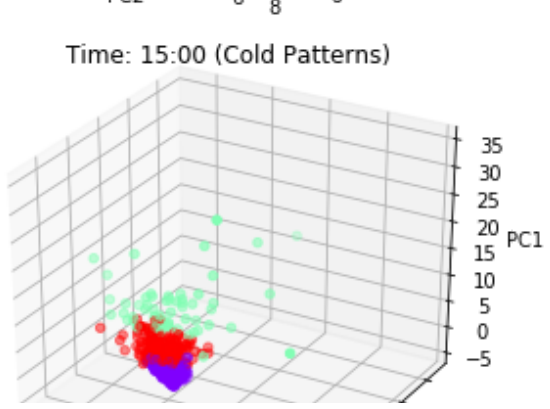

Time: 10:00 (Cold Patterns)
PC1
PC2
PC3

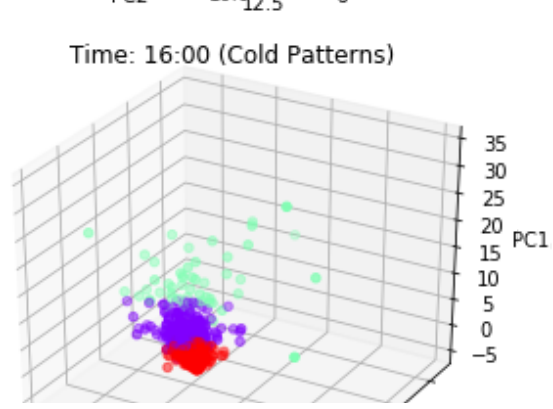

Time: 11:00 (Cold Patterns)
PC
PC2
PC3

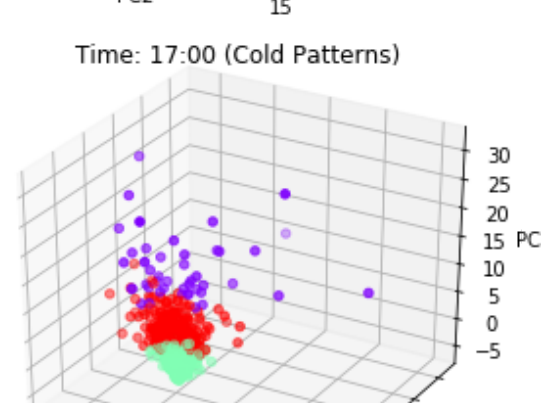

Time: 12:00 (Cold Patterns)
PC1
PC2
PC3

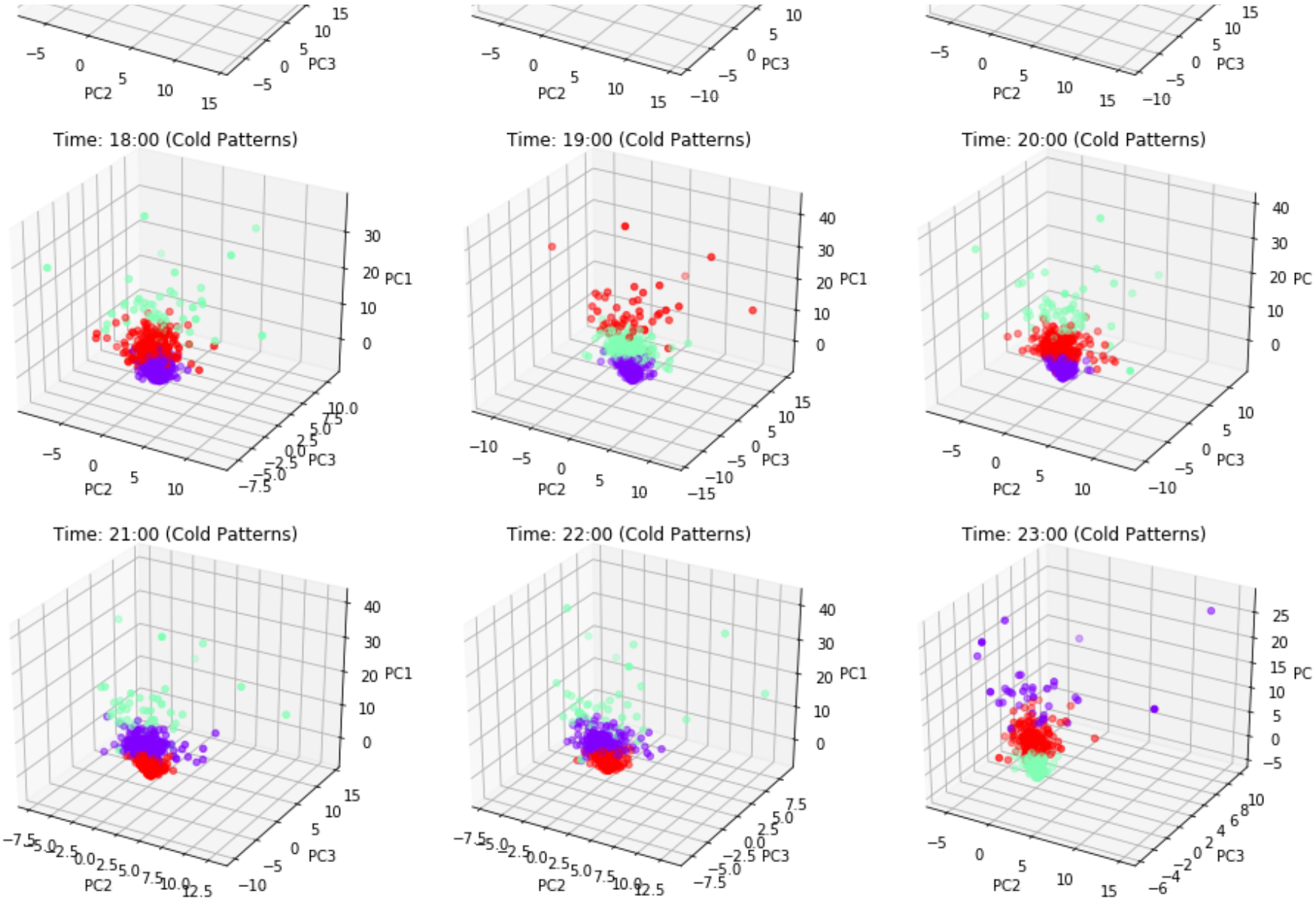

Time: 13:00 (Cold Patterns)
PC1
PC2
PC3


Time: 14:00 (Cold Patterns)
PC
PC2
PC3


Time: 15:00 (Cold Patterns)
PC1


Time: 16:00 (Cold Patterns)
PC1


Time: 17:00 (Cold Patterns)
PC



The ways of checking the clustering theta threshold condition

In [323]:
```
kmeans.cluster_centers_[1]
```

Out[323]:
```
array([13.69917824,  0.46278999, -0.14054859])
```

In [348]:
```
np.square(kmeans.cluster_centers_[1]).sum() * 0.2
```

Out[348]:
```
37.580282564512224
```

In [324]:
```
principleDf_nstd.iloc[kmeans.labels_ == 1][2].mean()
```

Out[324]:
```
-0.14054858878175358
```

In [366]:
```
(7+189+441)/976
```

Out[366]:
```
0.6526639344262295
```

In [379]:
```
test = principleDf_nstd.iloc[kmeans.labels_ == 0][[0,1,2]] - kmeans.cluster_centers_[0]
sum(np.square(test).sum(axis = 1) > np.square(kmeans.cluster_centers_[0]).sum() * 0.2)
```

Out[379]:
```
0
```

In [385]:
```
test = principleDf_nstd.iloc[kmeans.labels_ == 1][[0,1,2]] - kmeans.cluster_centers_[1]
sum(np.square(test).sum(axis = 1) > np.square(kmeans.cluster_centers_[1]).sum() * 0.2)
```

Out[385]:
```
63
```

In [384]:
```
test = principleDf_nstd.iloc[kmeans.labels_ == 2][[0,1,2]] - kmeans.cluster_centers_[2]
sum(np.square(test).sum(axis = 1) > np.square(kmeans.cluster_centers_[2]).sum() * 0.2)
```

Out[384]:
```
118
```

In [383]:
```
test = principleDf_nstd.iloc[kmeans.labels_ == 3][[0,1,2]] - kmeans.cluster_centers_[3]
sum(np.square(test).sum(axis = 1) > np.square(kmeans.cluster_centers_[3]).sum() * 0.2)
```

Out[383]:
```
1
```

In [395]:
```
tot = 0
for i in range(50):
    test = principleDf_nstd.iloc[kmeans.labels_ == i][[0,1,2]] - kmeans.cluster_centers_[i]
    tot = tot + sum(np.square(test).sum(axis = 1) > np.square(kmeans.cluster_centers_[i]).sum() * 0.2)
tot
```

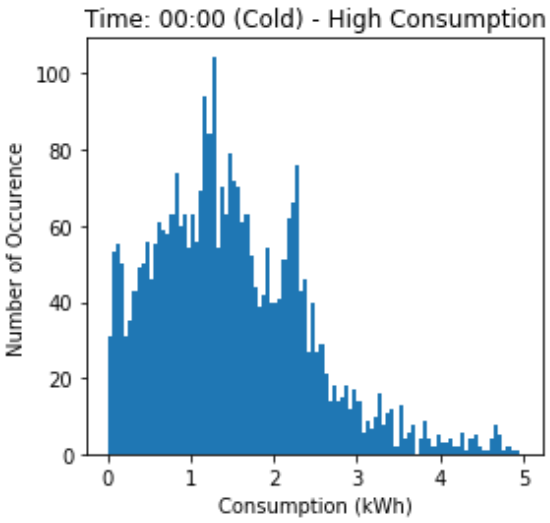
Out[395]:
```
184
```

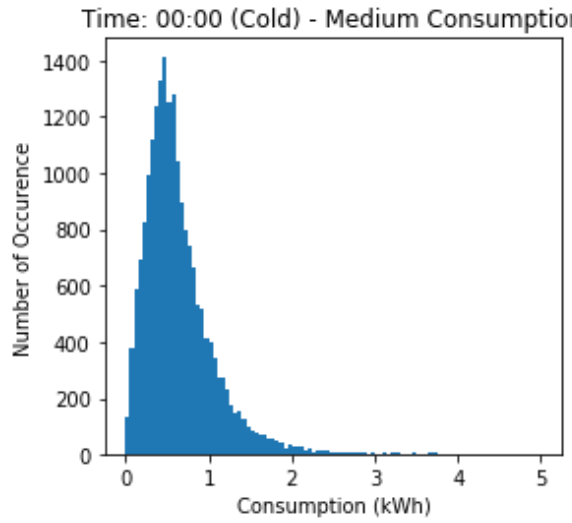
In [359]:
```
kmeans.cluster_centers_[2]
```

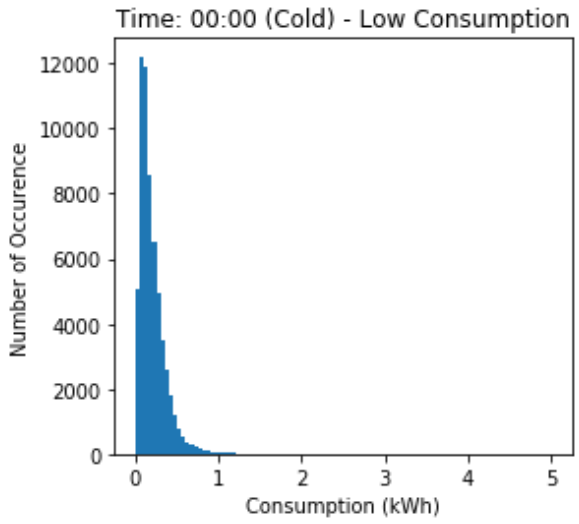
Out[359]:
```
array([-1.85445773,  0.06602678, -0.00851691])
```

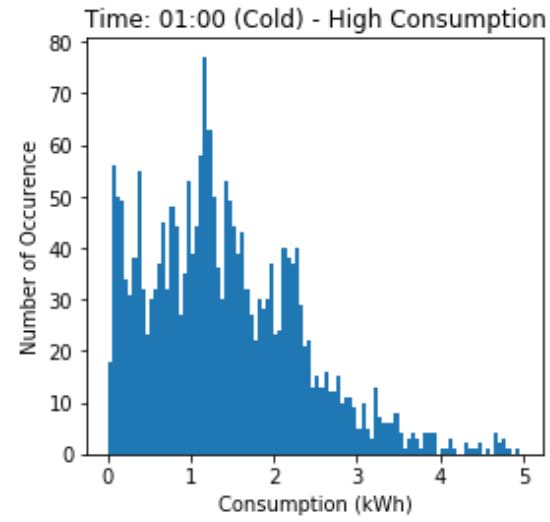
General consumptions distribution property of each cluster at each hour

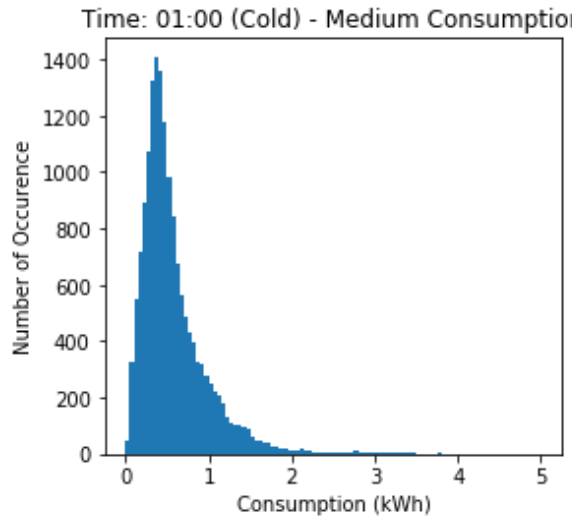
In [17]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
cons_type = ['High Consumption', 'Medium Consumption', 'Low Consumption']
fig_all = plt.figure(figsize = (13,90))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1)
            t2 = t[t.columns[kmeans.labels_ == sorted_label[k][0]]].copy() # select all data from the n-th cl
uster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            ax.hist(t2[t2.columns[1:-1]].values.flatten(), bins=100, range=(0,5))
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1)
            t2 = t[t.columns[kmeans.labels_ == sorted_label[k][0]]].copy() # select all data from the n-th cl
uster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            ax.hist(t2[t2.columns[1:-1]].values.flatten(), bins=100, range=(0,5))
            ax.set_title('Time: ' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Number of Occurence')
plt.tight_layout()
```

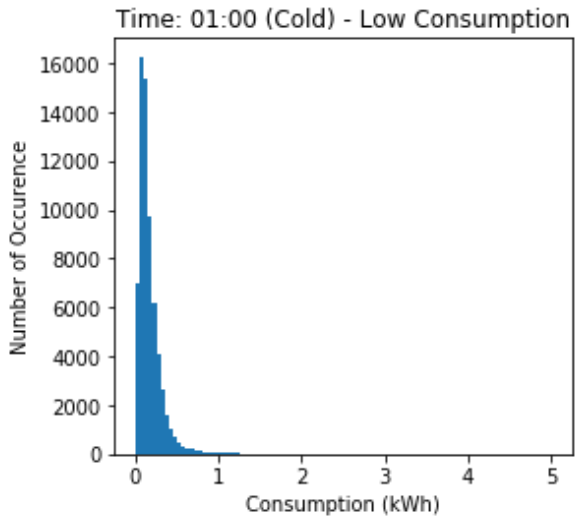

Time: 00:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

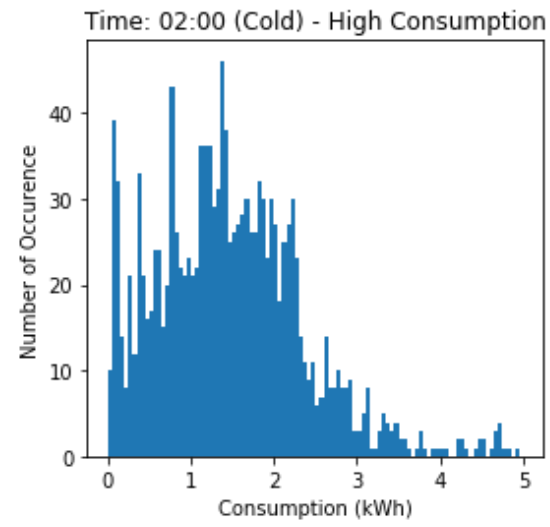

Time: 00:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

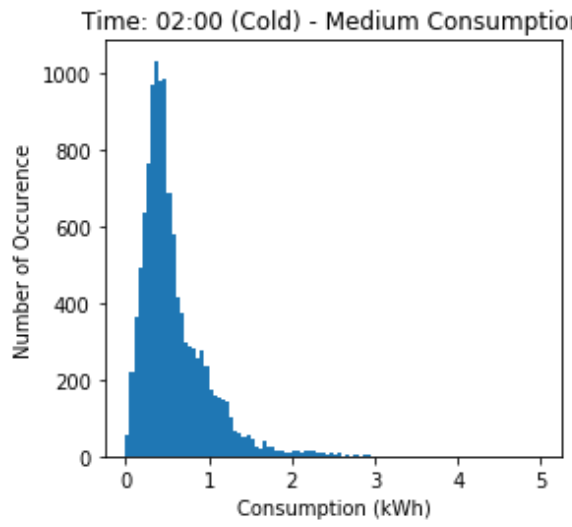

Time: 00:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

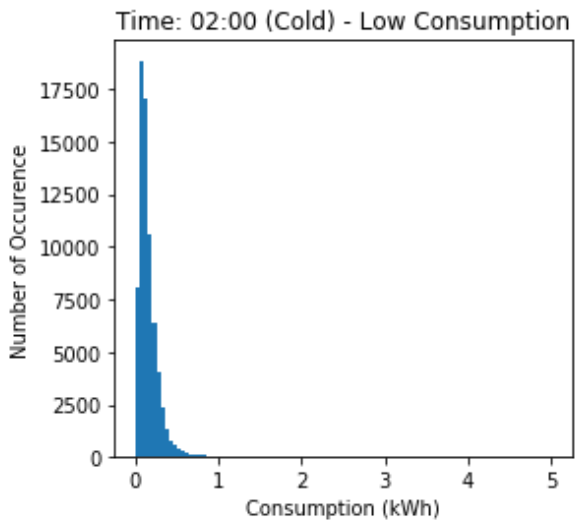

Time: 01:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

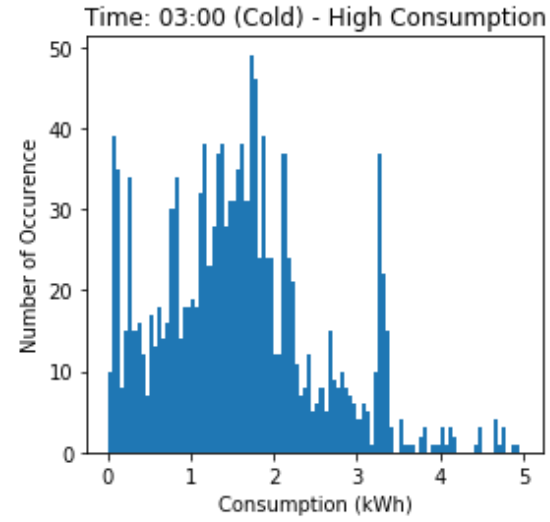

Time: 01:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

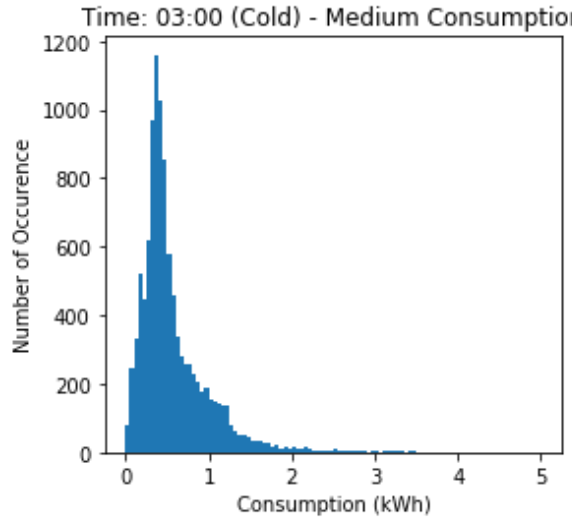

Time: 01:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

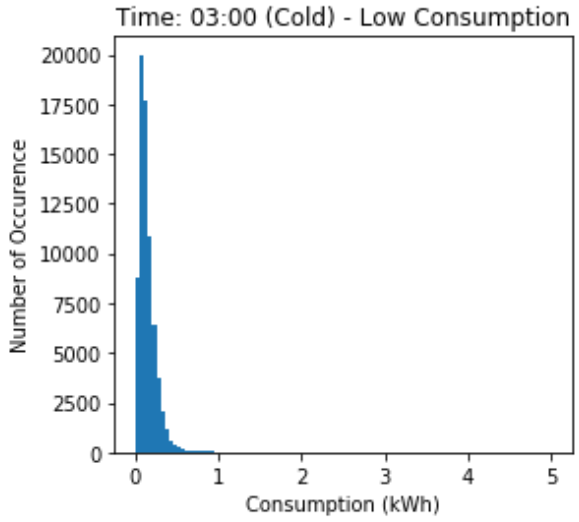

Time: 02:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

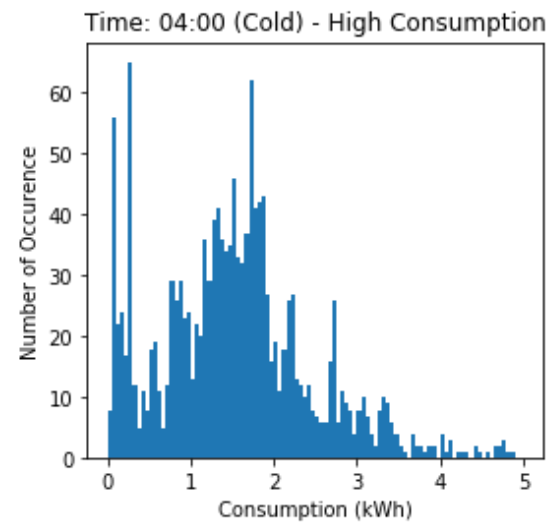

Time: 02:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

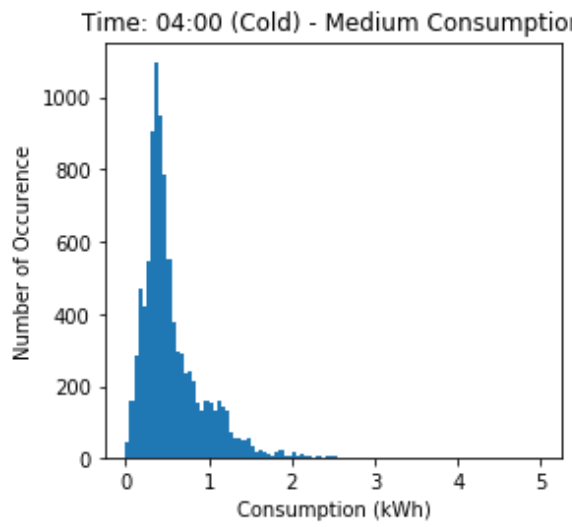

Time: 02:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

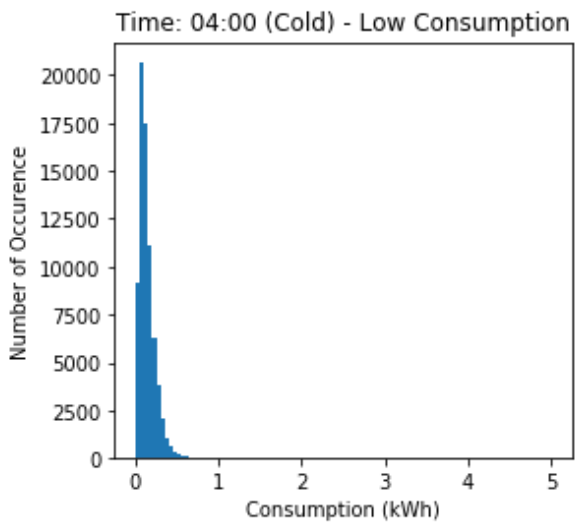

Time: 03:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

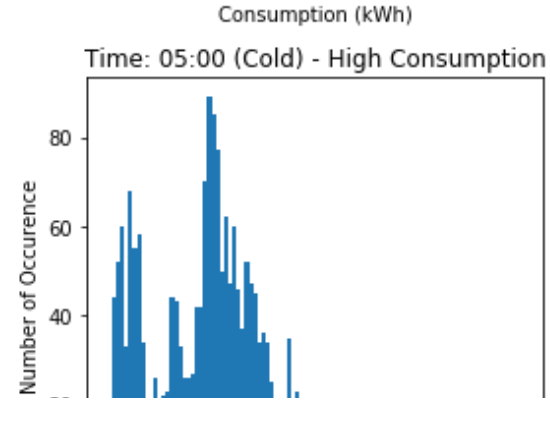

Time: 03:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

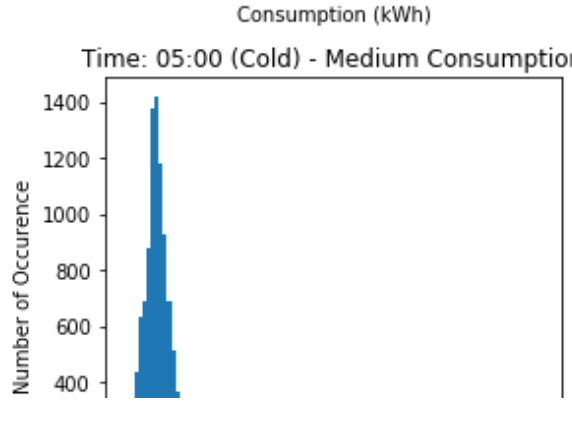

Time: 03:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

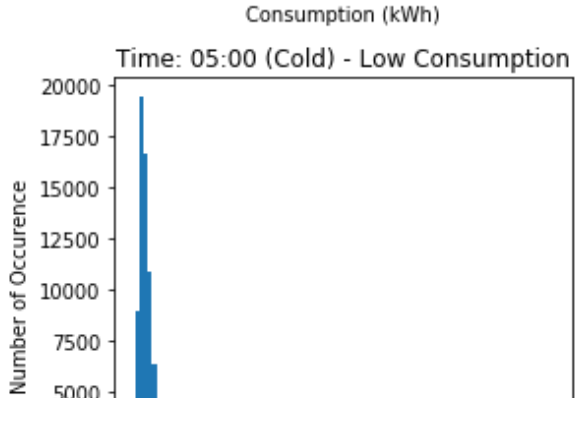

Time: 04:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

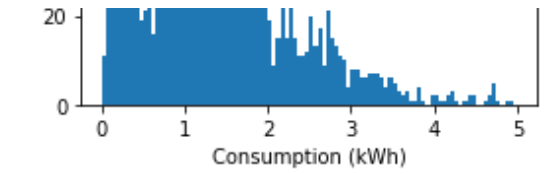

Time: 04:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

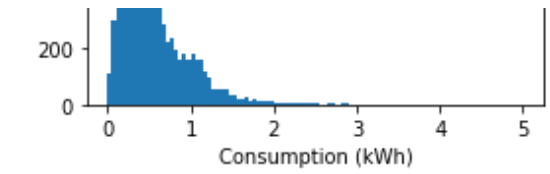

Time: 04:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

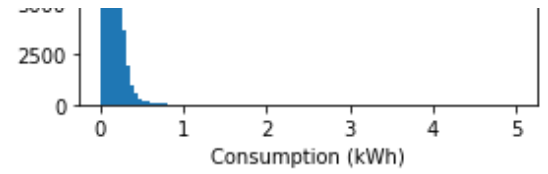

Time: 05:00 (Cold) - High Consumption
Number of Occurence

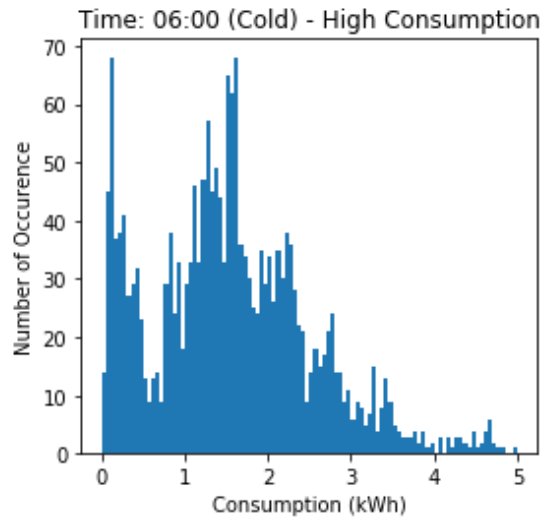

Time: 05:00 (Cold) - Medium Consumption
Number of Occurence

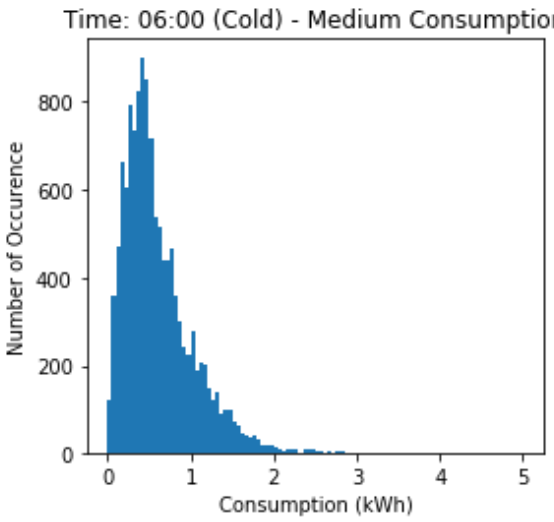

Time: 05:00 (Cold) - Low Consumption
Number of Occurence

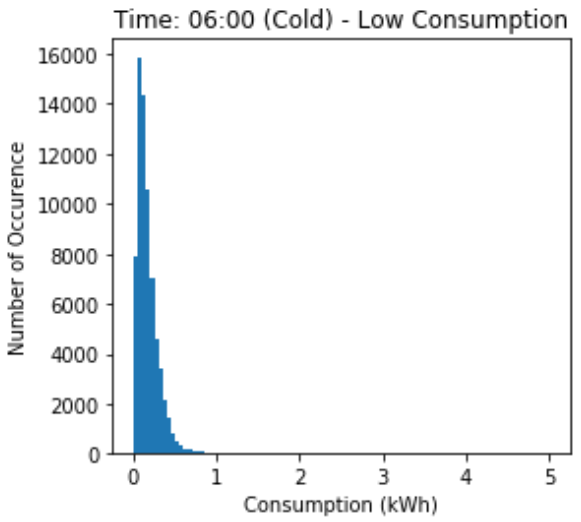

Consumption (kWh)

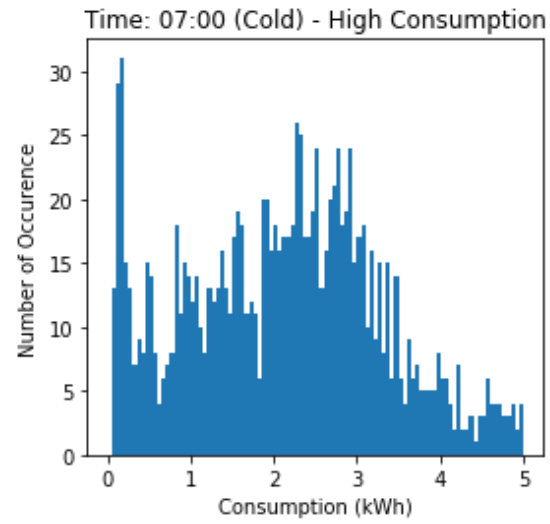

Consumption (kWh)

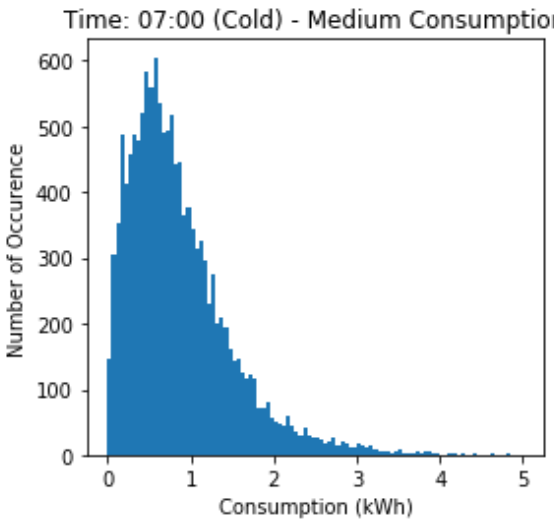

Consumption (kWh)

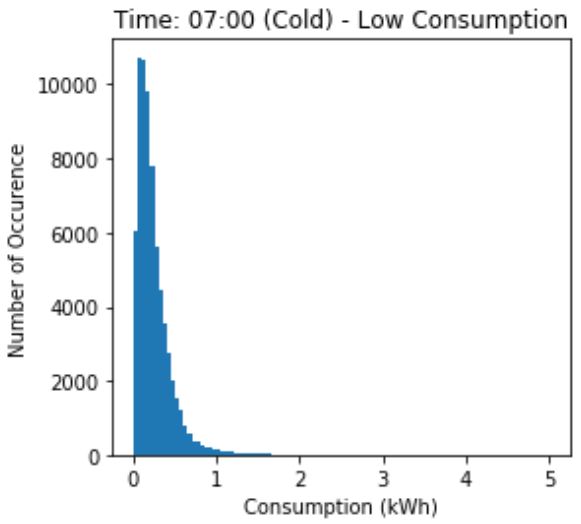

Time: 06:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

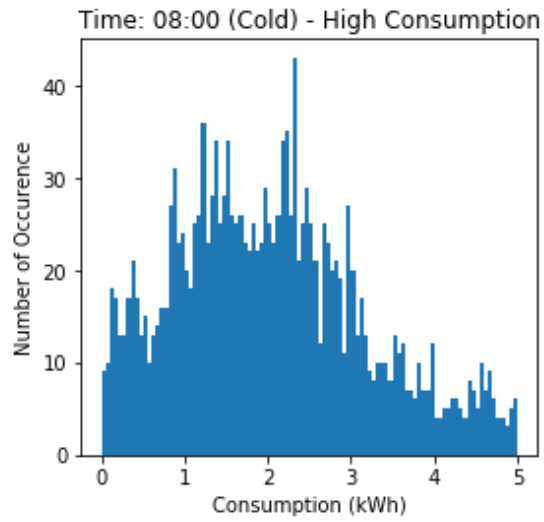

Time: 06:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

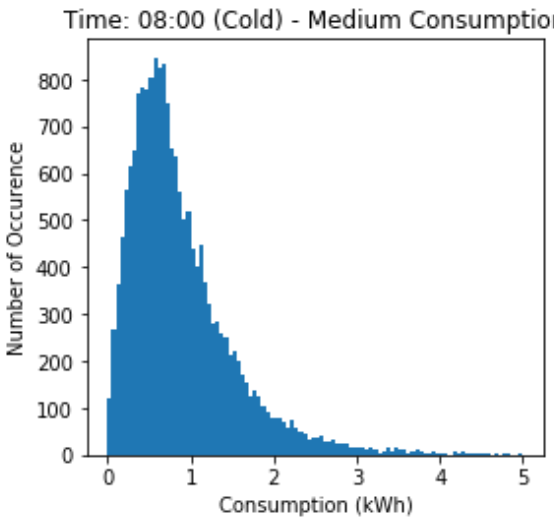

Time: 06:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

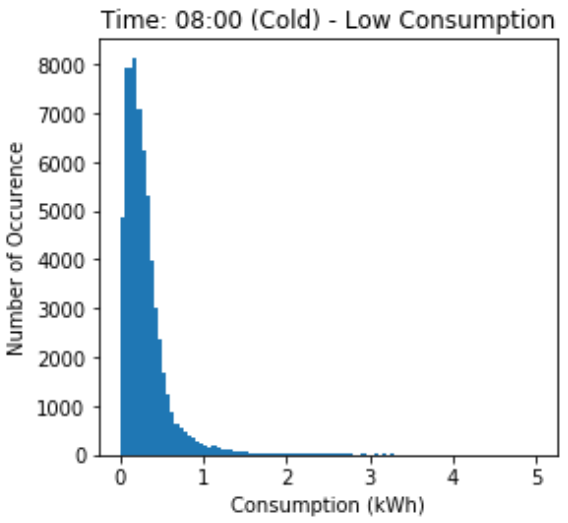

Time: 07:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

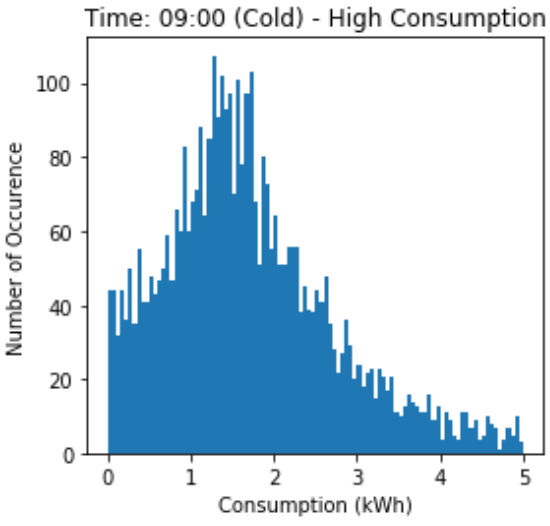

Time: 07:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

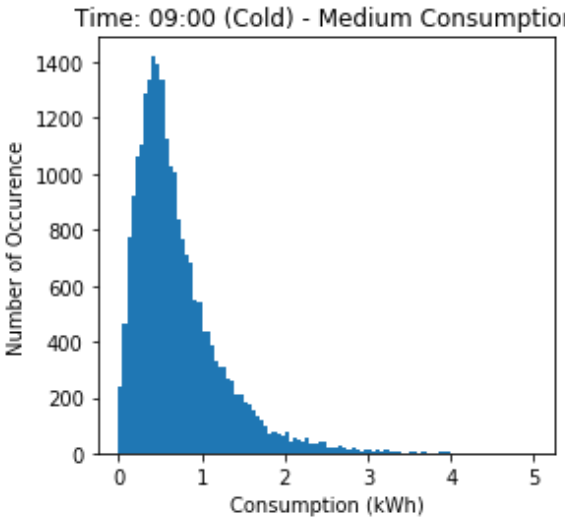

Time: 07:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

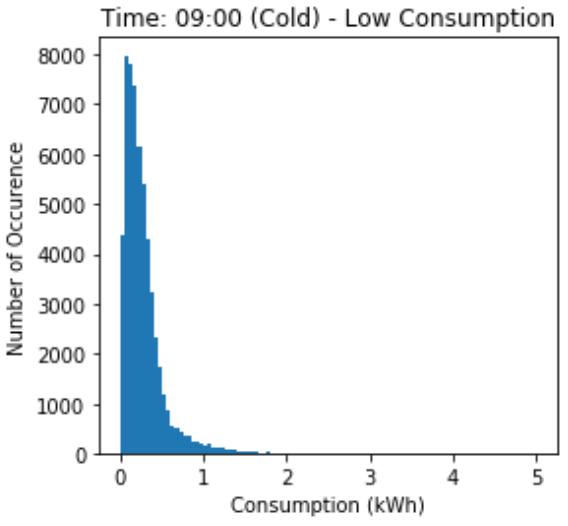

Time: 08:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

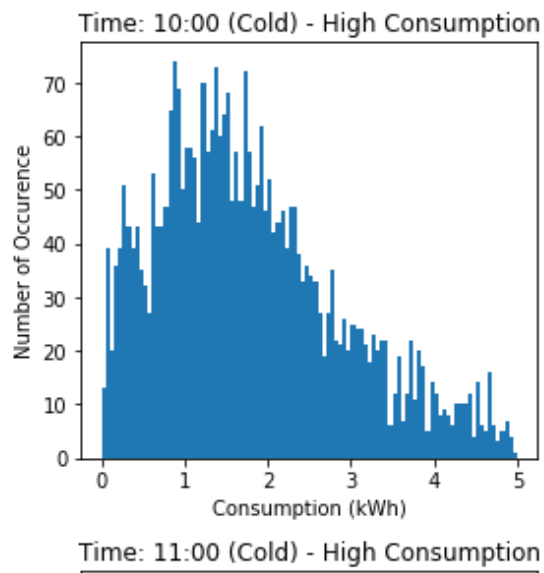

Time: 08:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

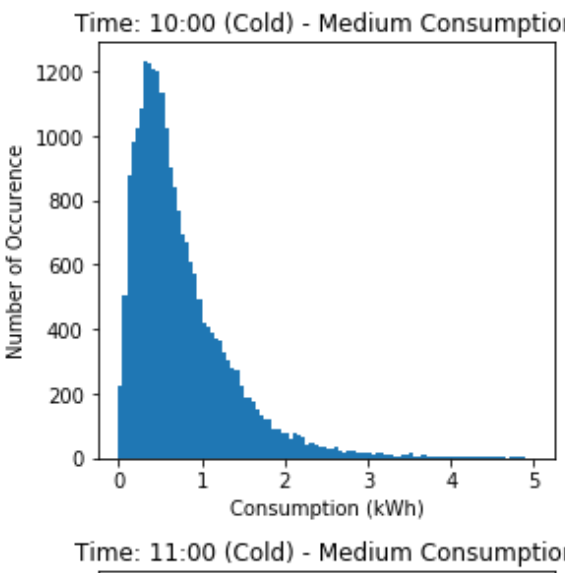

Time: 08:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

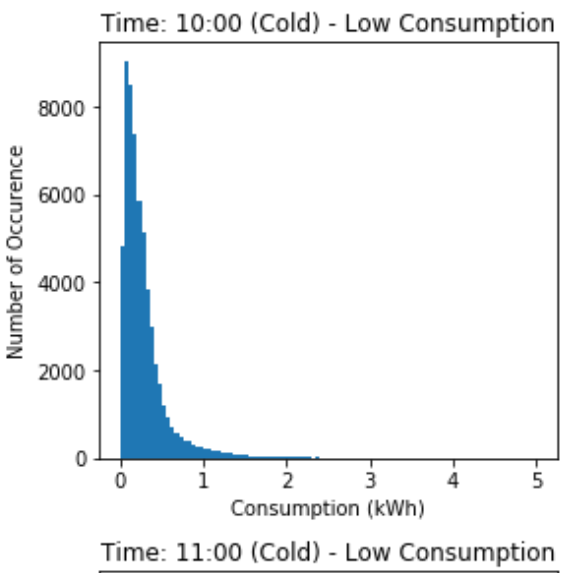

Time: 09:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)


Time: 09:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)


Time: 09:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)


Time: 10:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

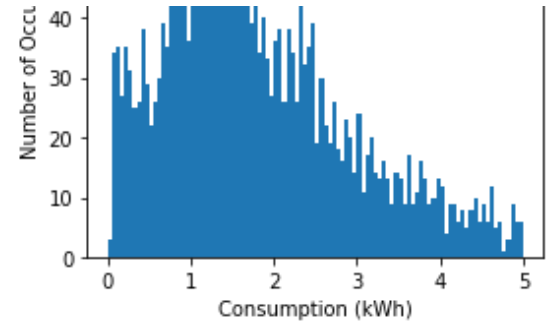

Time: 10:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)


Time: 10:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

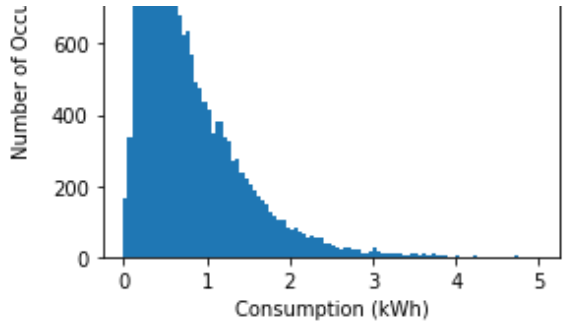
Time: 11:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

Time: 11:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

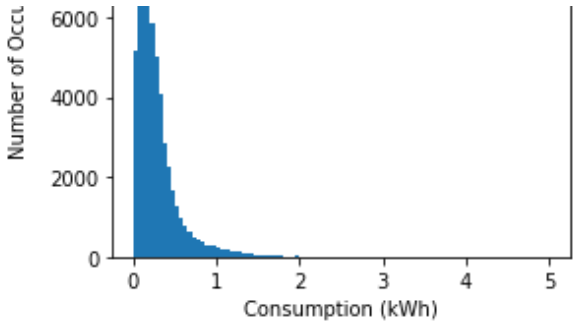
Time: 11:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)



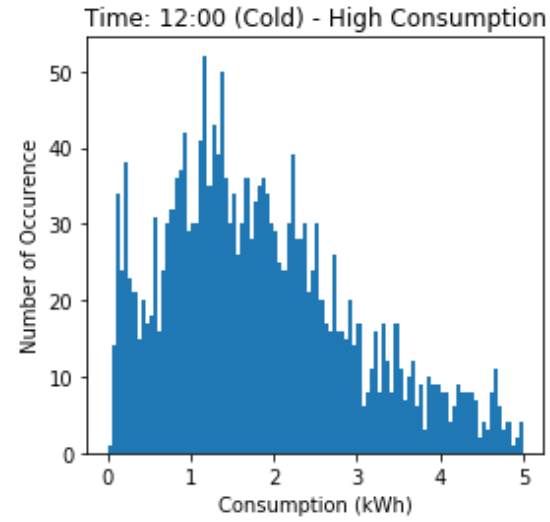
Time: 12:00 (Cold) - High Consumption



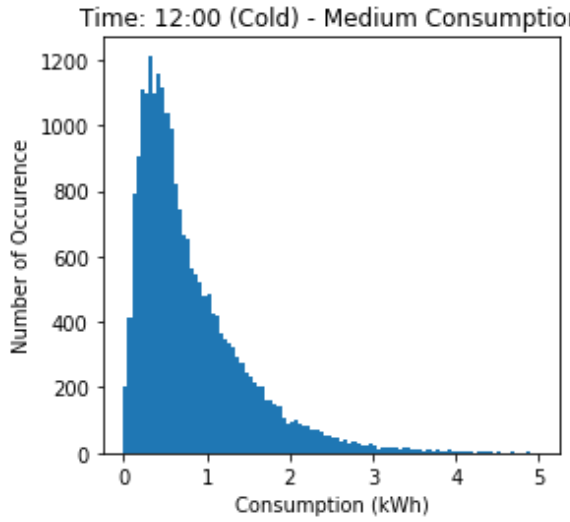
Time: 12:00 (Cold) - Medium Consumption



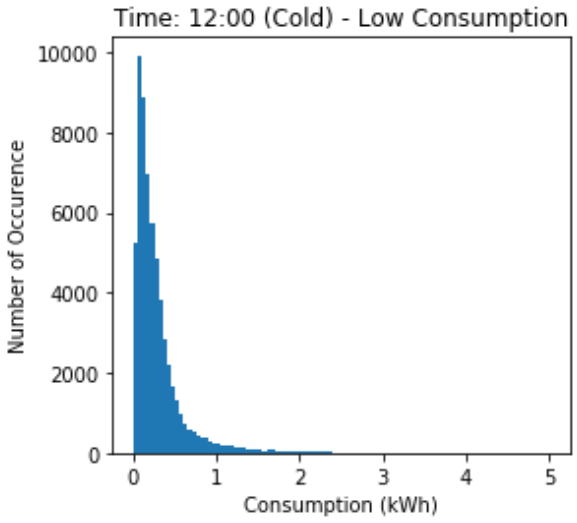
Time: 12:00 (Cold) - Low Consumption



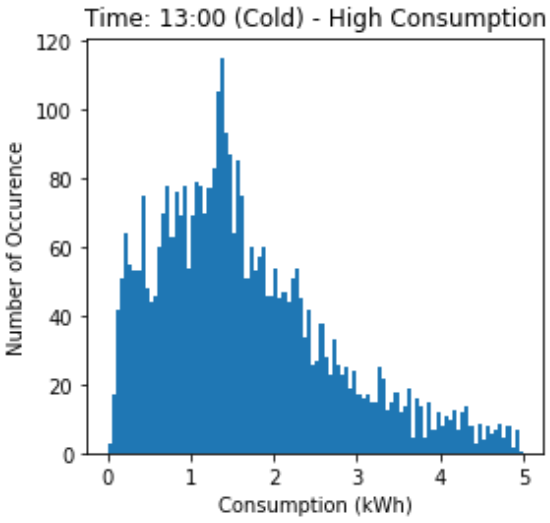
Time: 13:00 (Cold) - High Consumption



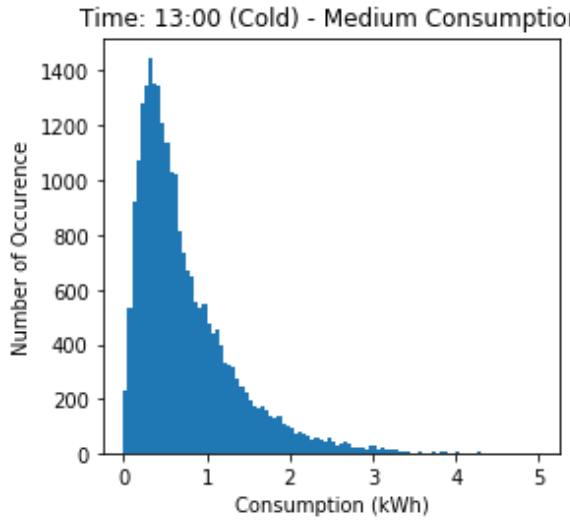
Time: 13:00 (Cold) - Medium Consumption



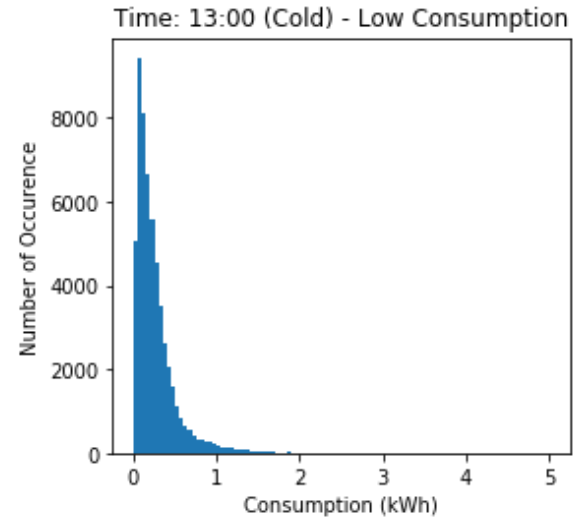
Time: 13:00 (Cold) - Low Consumption



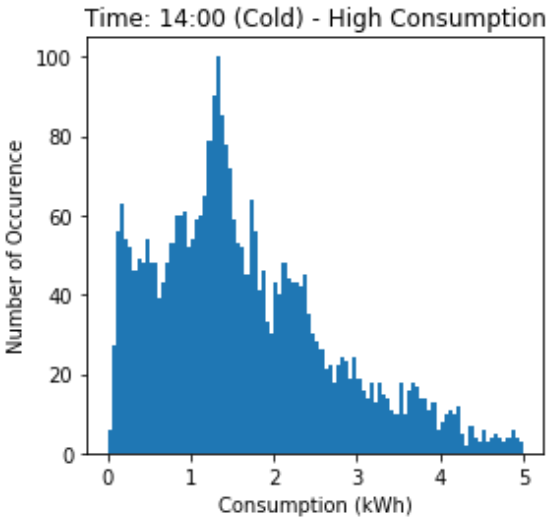
Time: 14:00 (Cold) - High Consumption



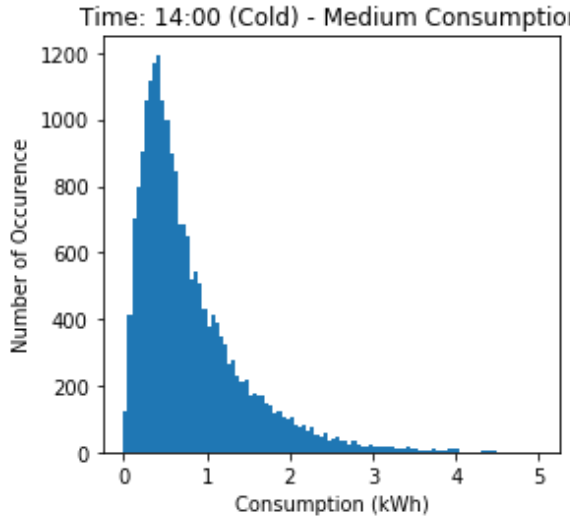
Time: 14:00 (Cold) - Medium Consumption



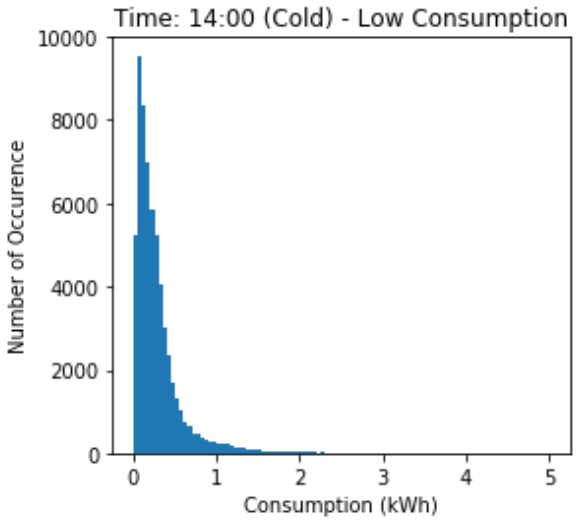
Time: 14:00 (Cold) - Low Consumption



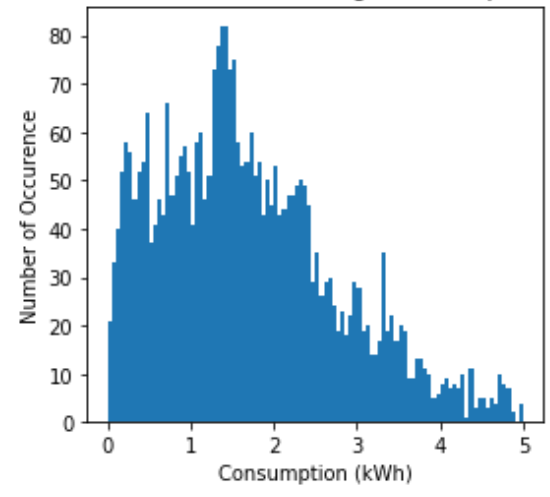
Time: 15:00 (Cold) - High Consumption



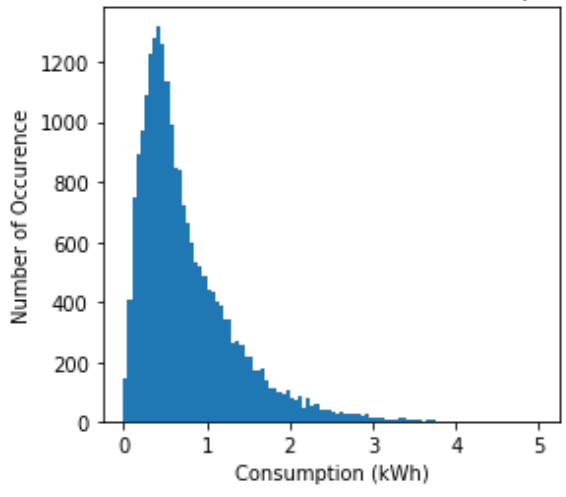
Time: 15:00 (Cold) - Medium Consumption



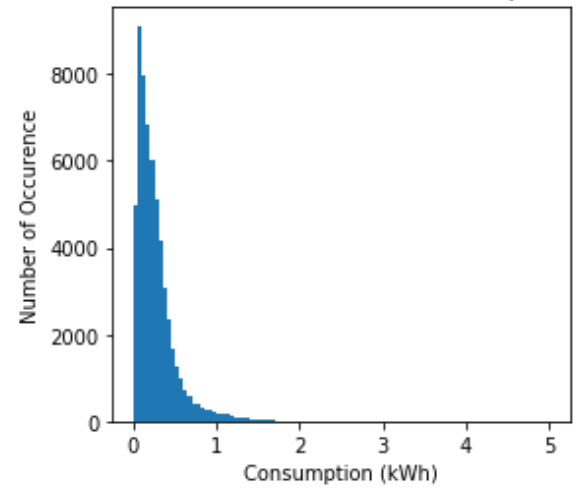
Time: 15:00 (Cold) - Low Consumption



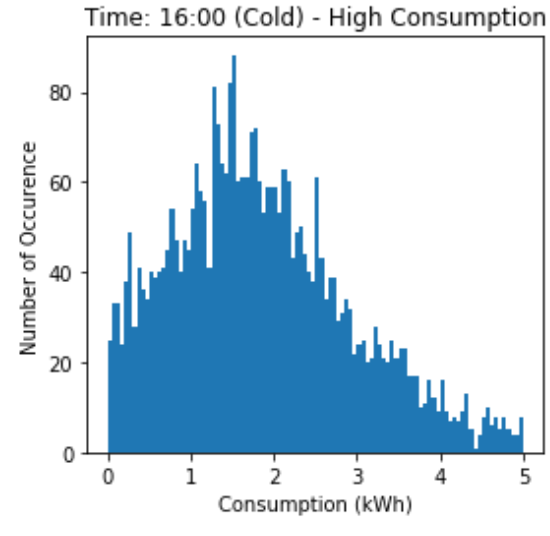
Time: 16:00 (Cold) - High Consumption



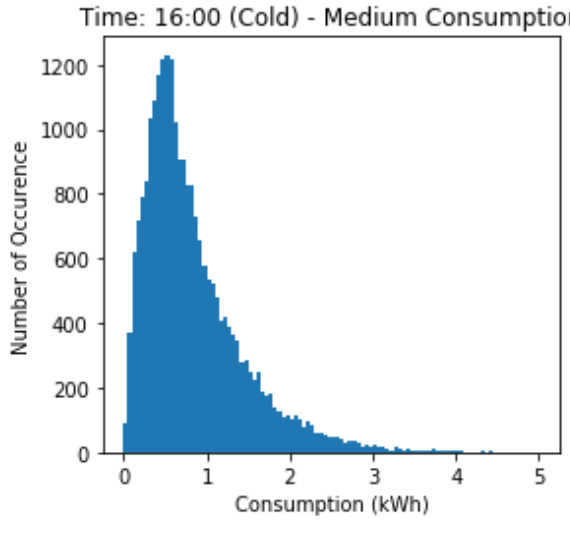
Time: 16:00 (Cold) - Medium Consumption



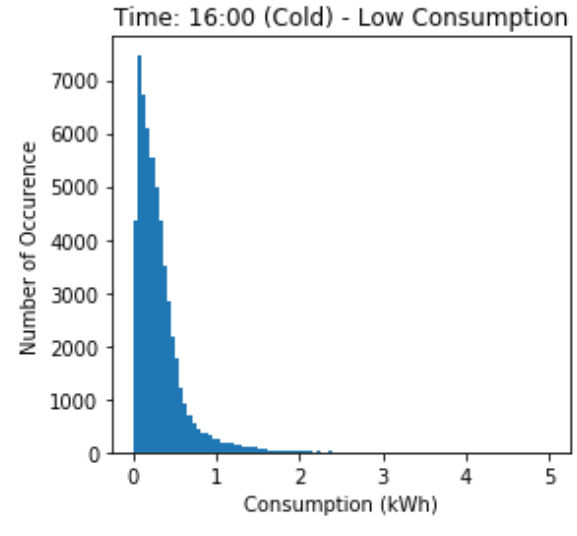
Time: 16:00 (Cold) - Low Consumption



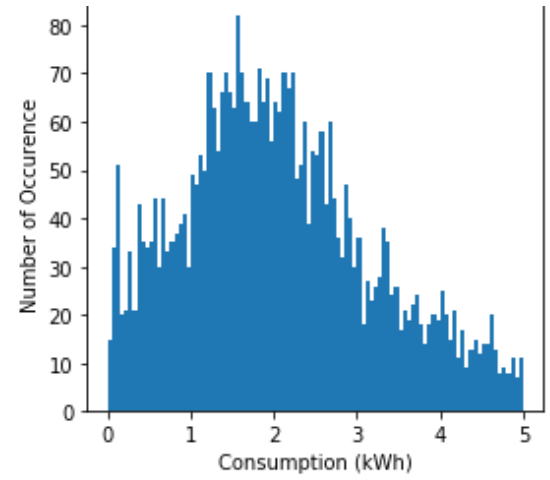
Time: 17:00 (Cold) - High Consumption

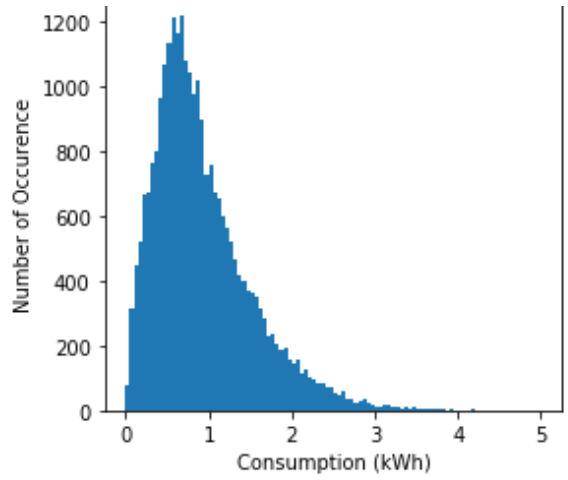


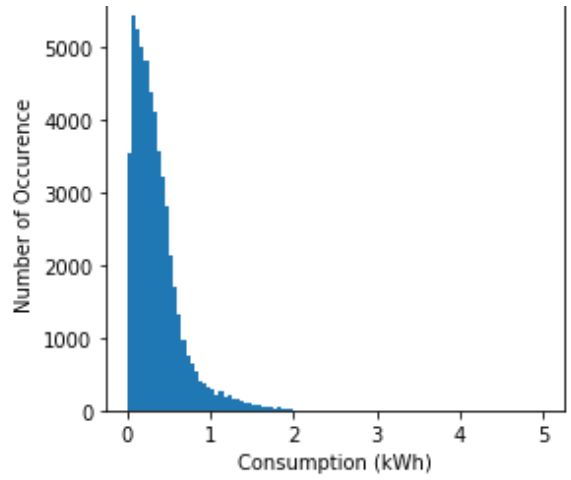
Time: 17:00 (Cold) - Medium Consumption

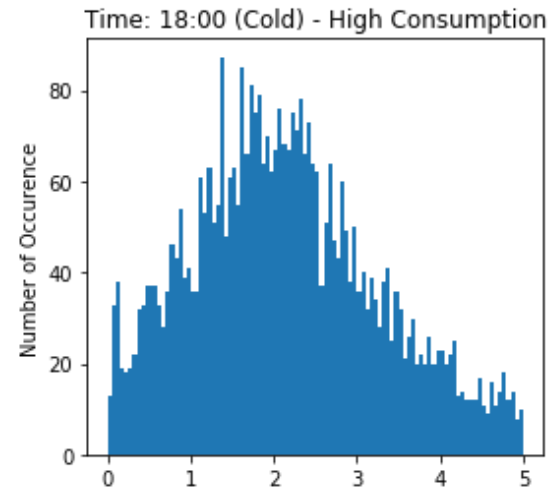


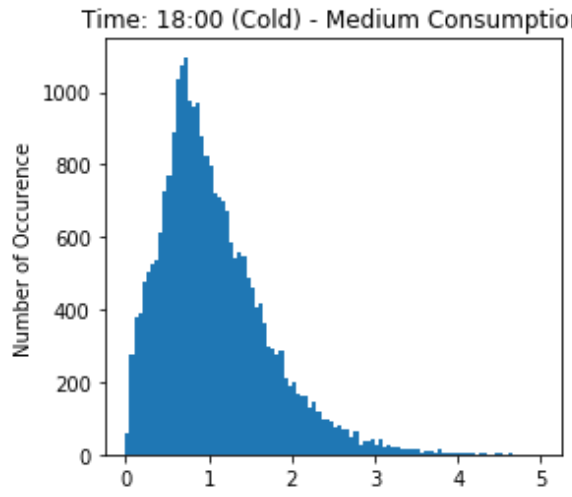
Time: 17:00 (Cold) - Low Consumption

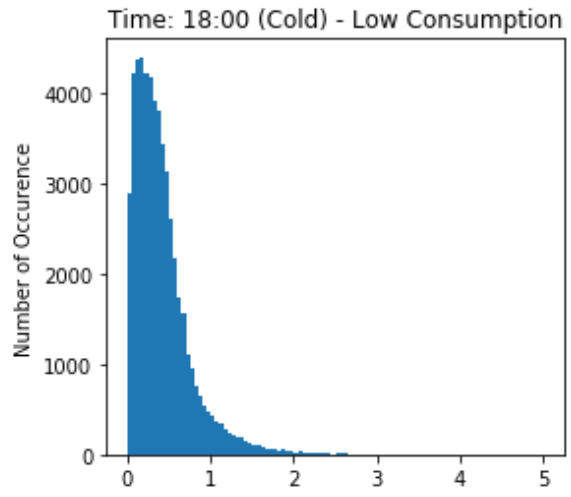

Number of Occurence
Consumption (kWh)

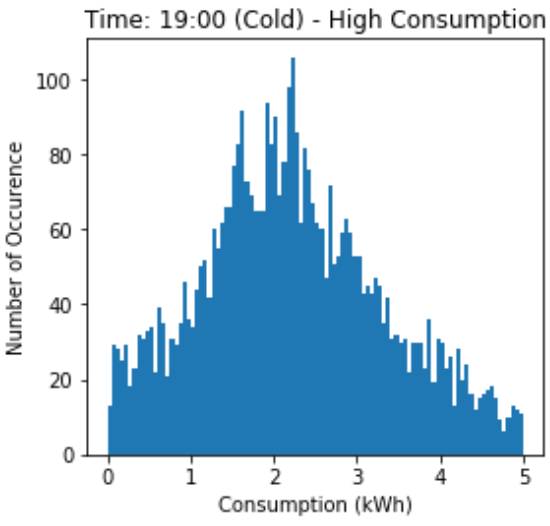

Number of Occurence
Consumption (kWh)

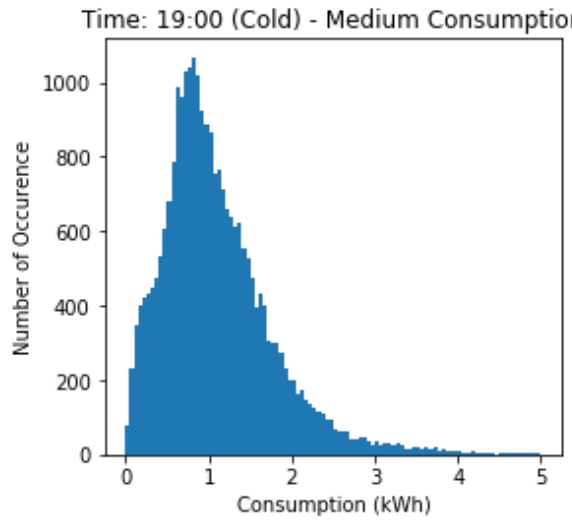

Number of Occurence
Consumption (kWh)

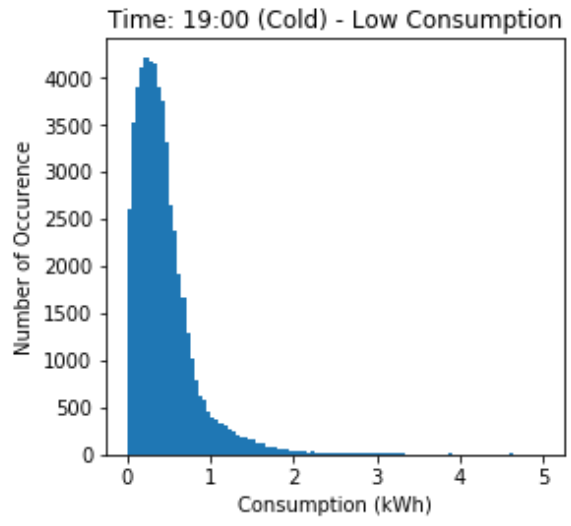

Time: 18:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

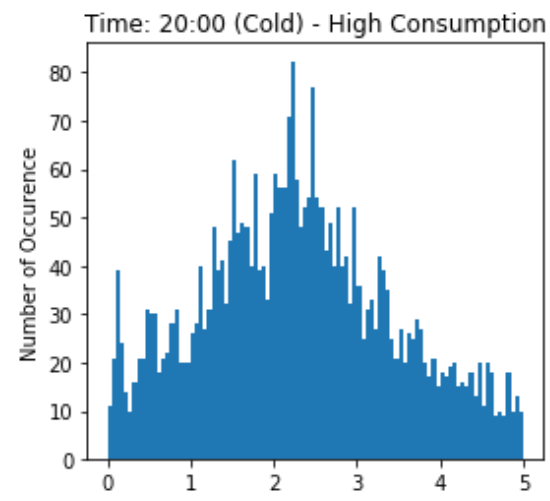

Time: 18:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

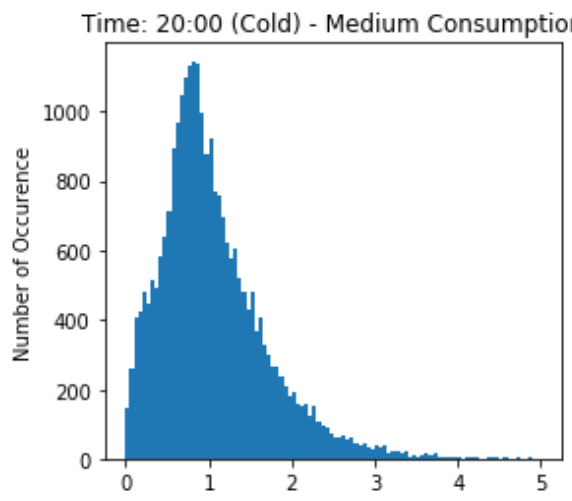

Time: 18:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

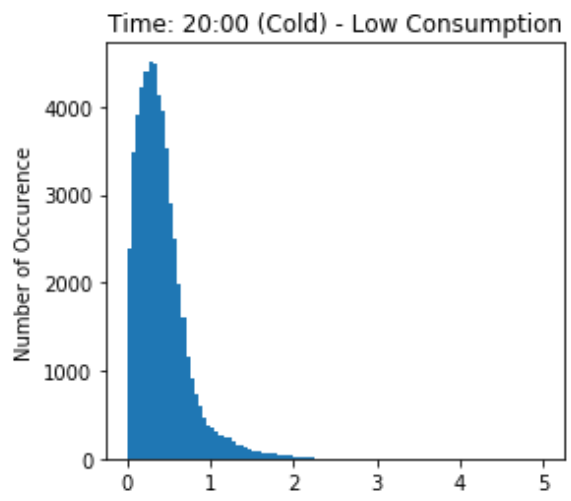

Time: 19:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

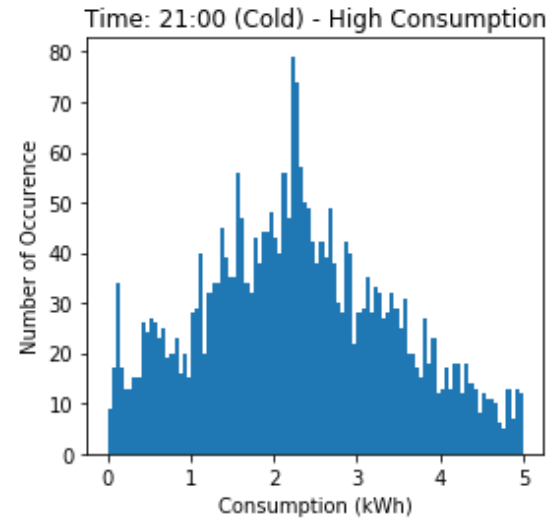

Time: 19:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

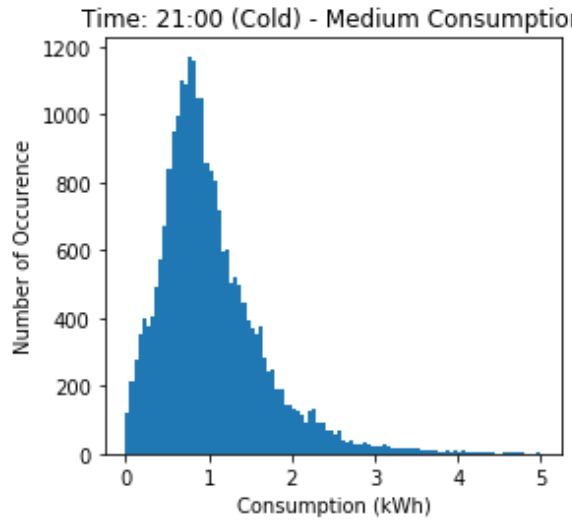

Time: 19:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

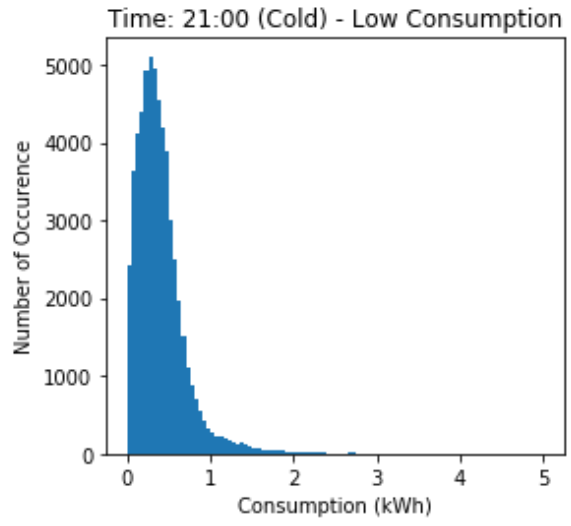

Time: 20:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

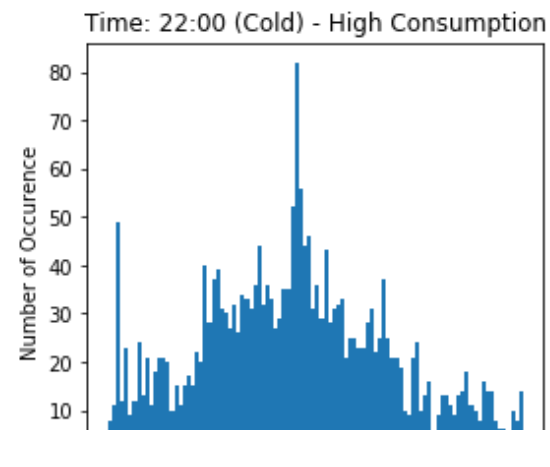

Time: 20:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)

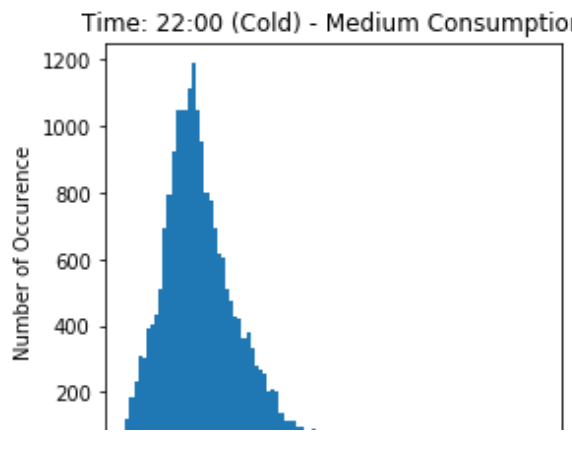

Time: 20:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)

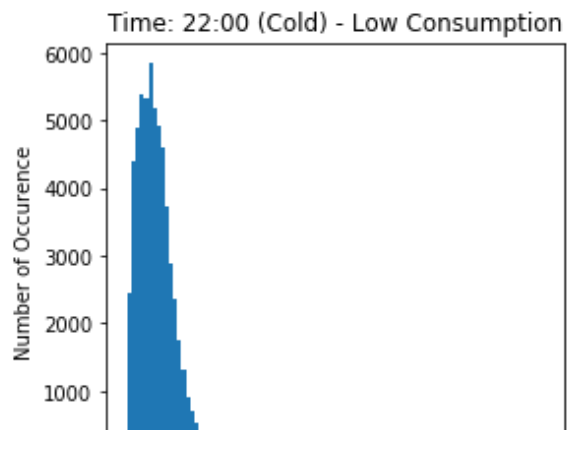

Time: 21:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)

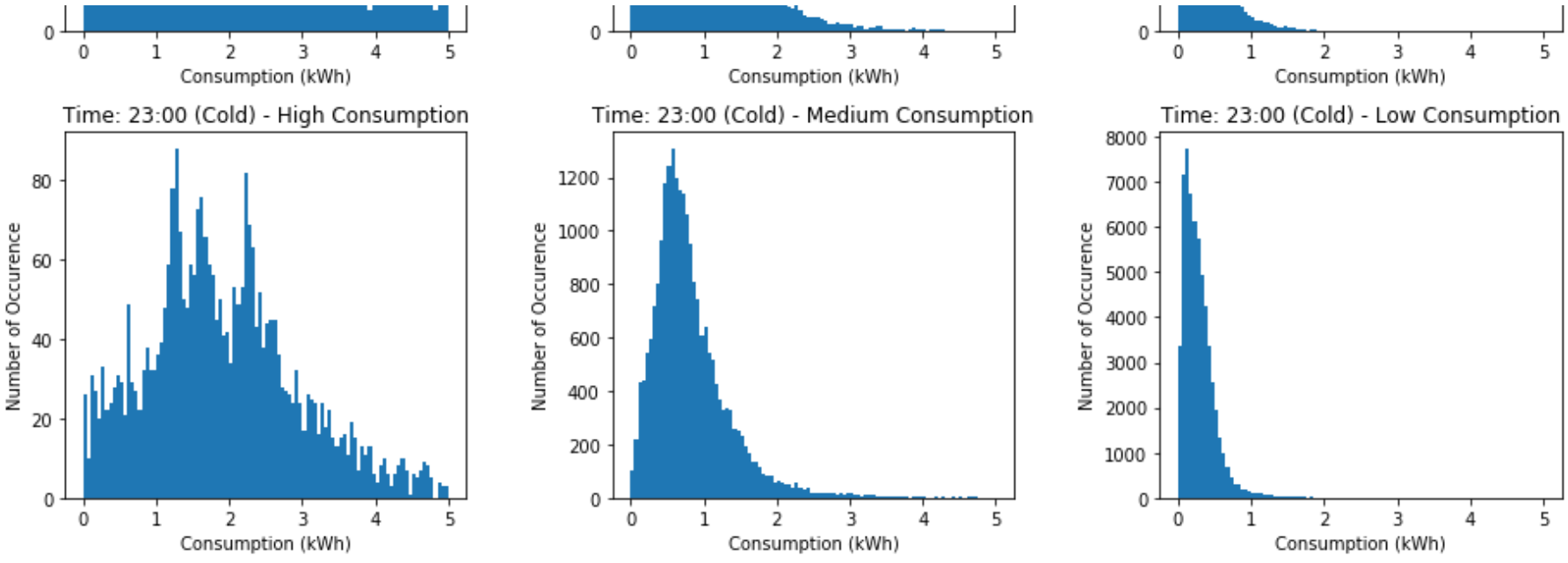

Time: 21:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)


Time: 21:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)


Time: 22:00 (Cold) - High Consumption
Number of Occurence

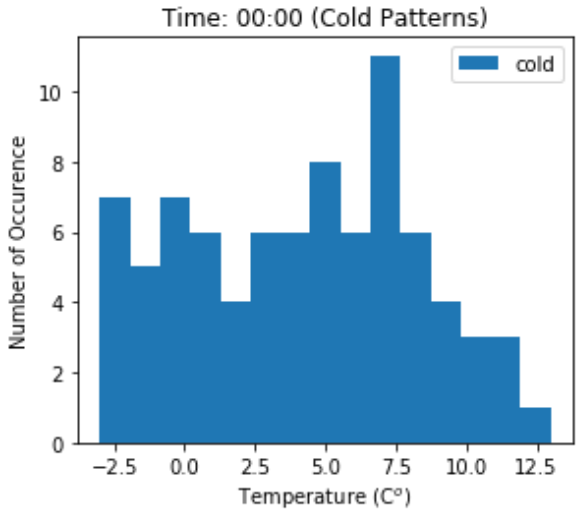

Time: 22:00 (Cold) - Medium Consumption
Number of Occurence

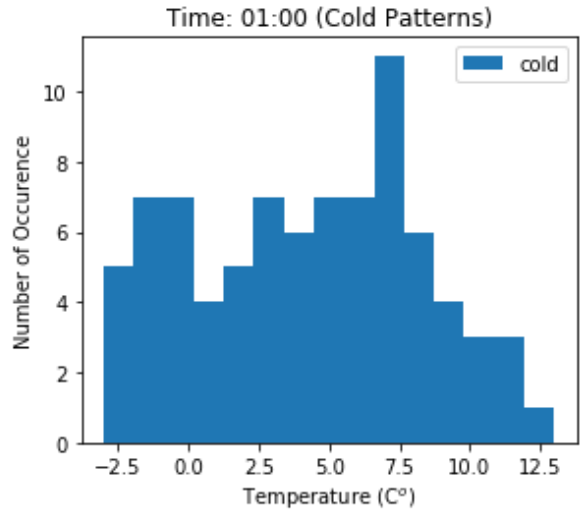

Time: 22:00 (Cold) - Low Consumption
Number of Occurence

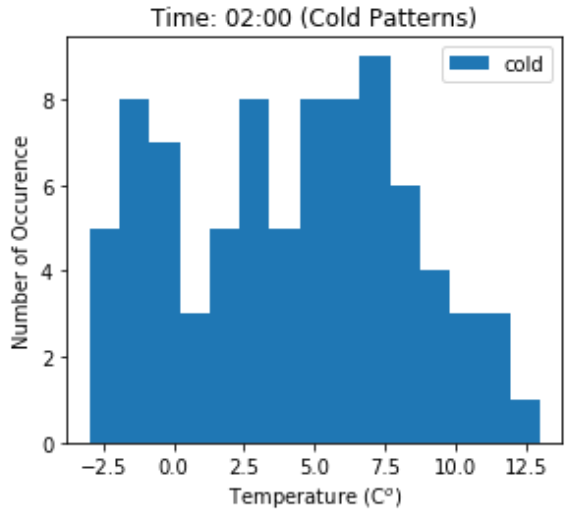


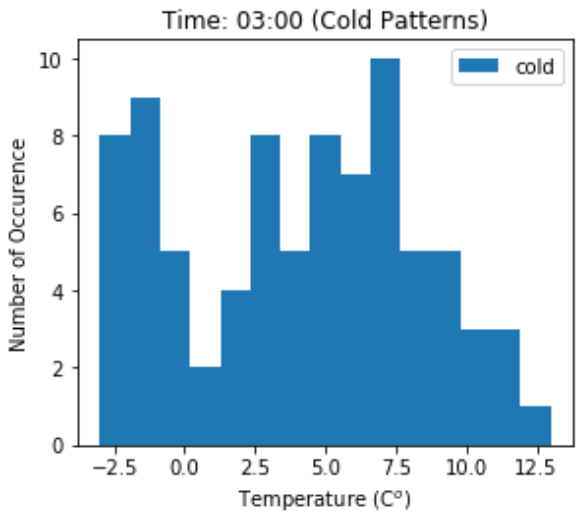
General days of a week distribution property of each cluster at each hour

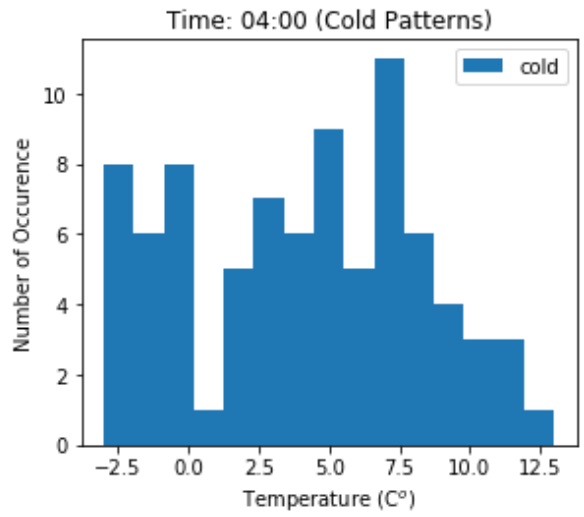
In [317]:
```
# temperature distribution of different hours of a day in a year
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1)
    if i <= 9:
        ax.hist(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]['TempC'], bins=1
5, label = 'cold')
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Number of Occurence')
    else:
        ax.hist(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['TempC'], bins=15, lab
el = 'cold')
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.legend()
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Number of Occurence')
plt.tight_layout()
```

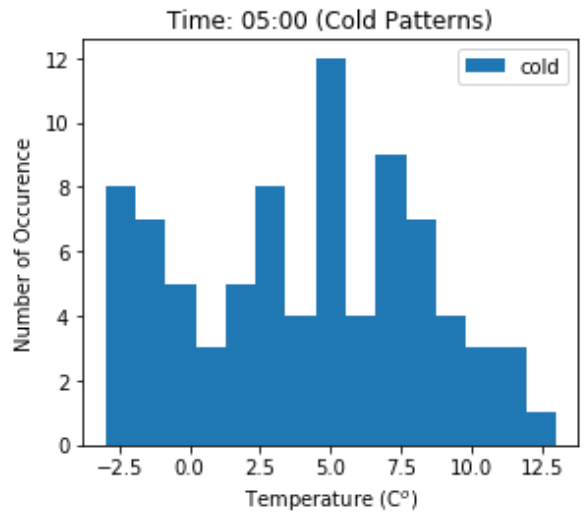

Time: 00:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

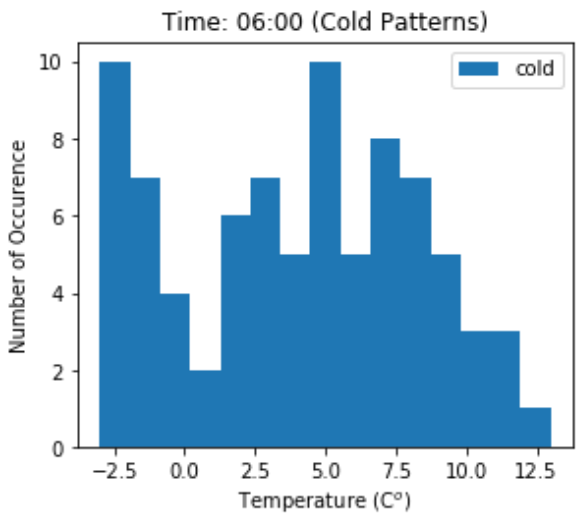

Time: 01:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

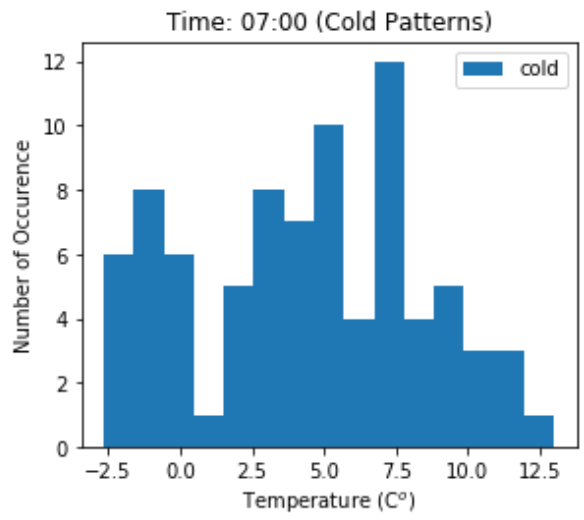

Time: 02:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

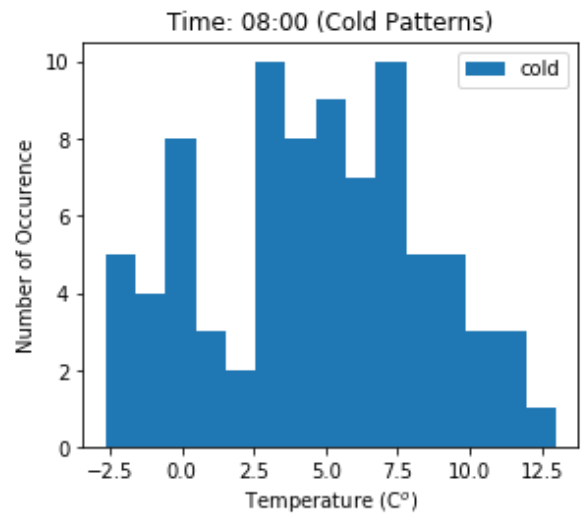

Time: 03:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

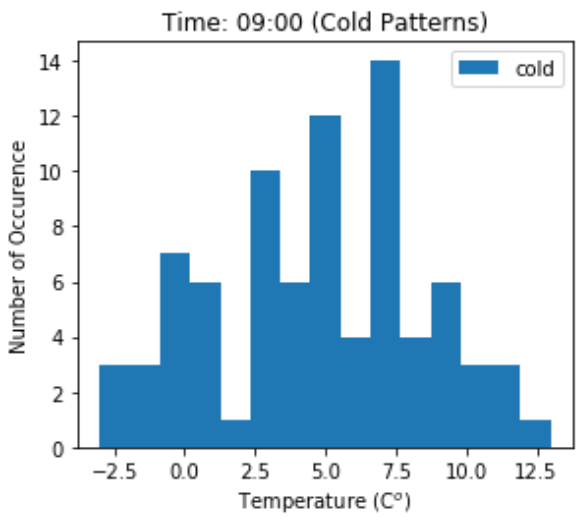

Time: 04:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

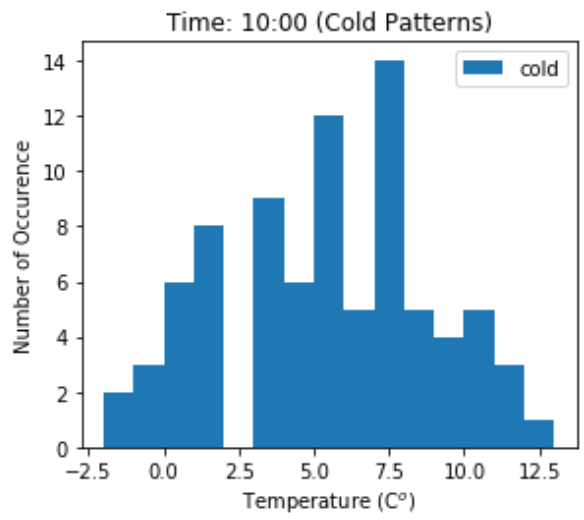

Time: 05:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

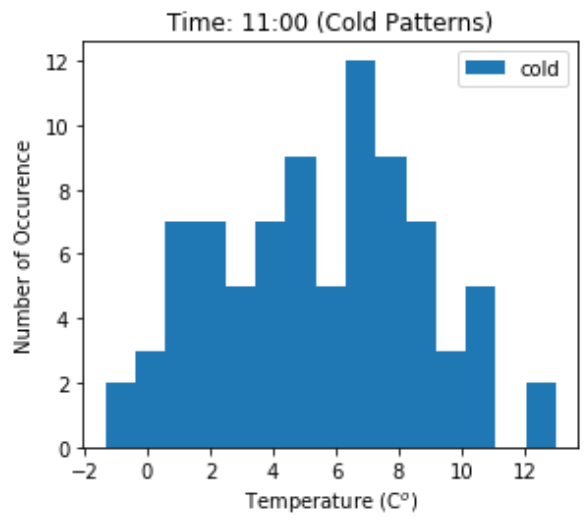

Time: 06:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

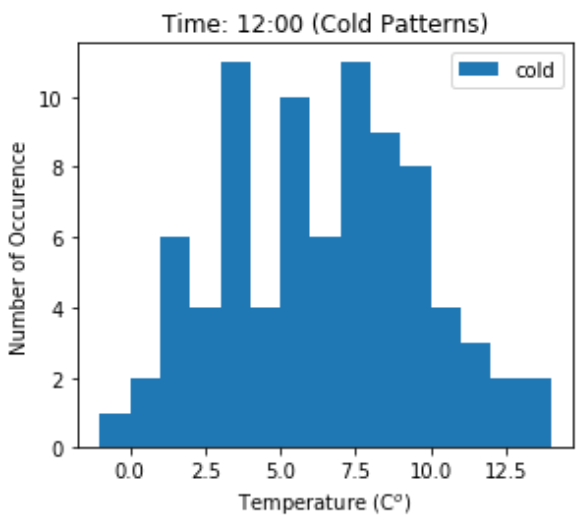

Time: 07:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

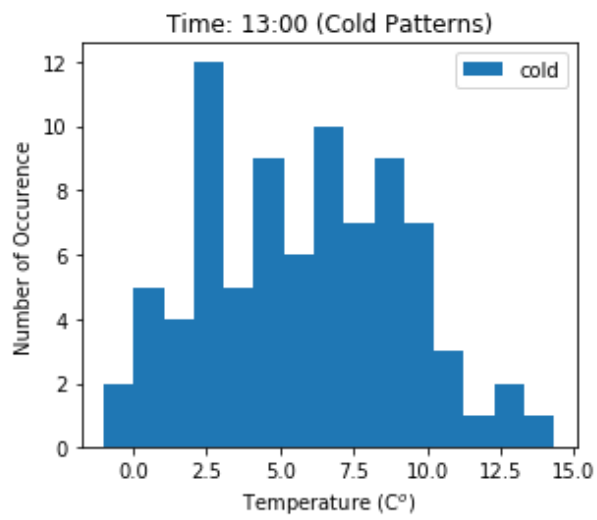

Time: 08:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

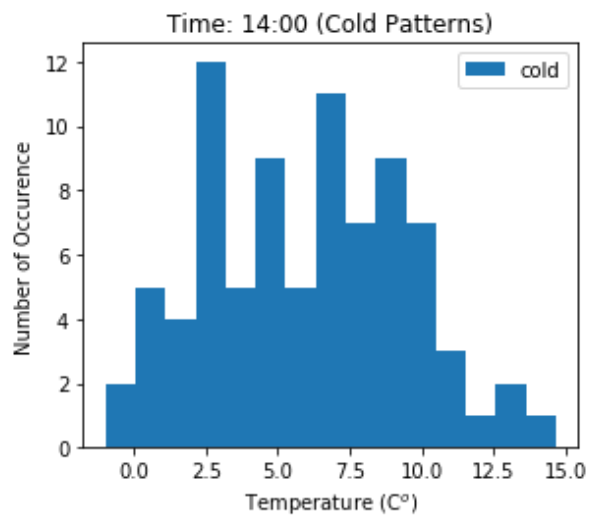

Time: 09:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

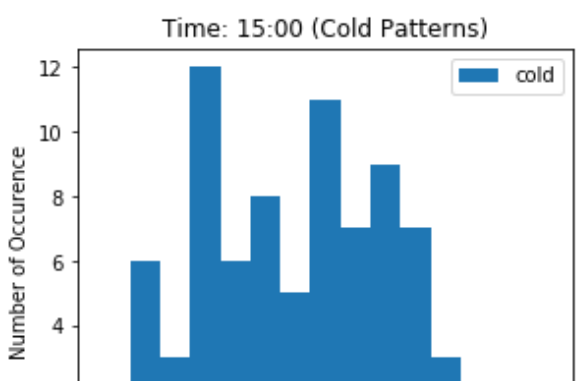

Time: 10:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

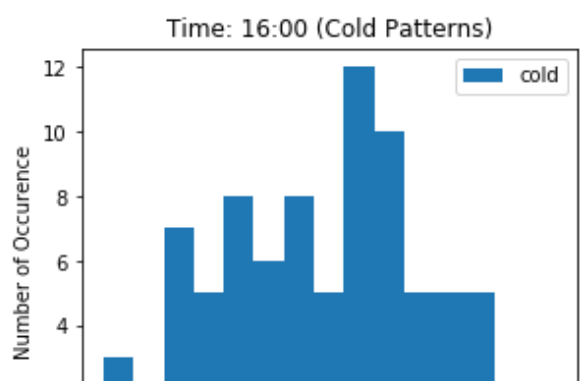

Time: 11:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

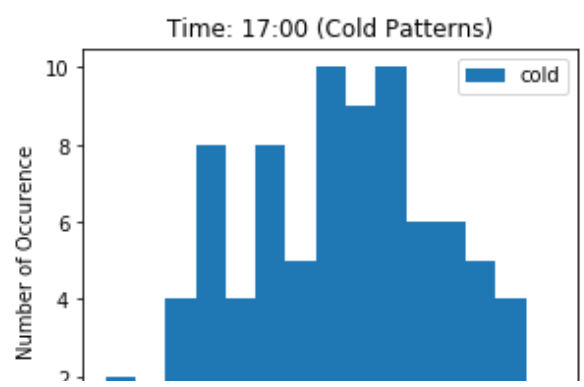

Time: 12:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)

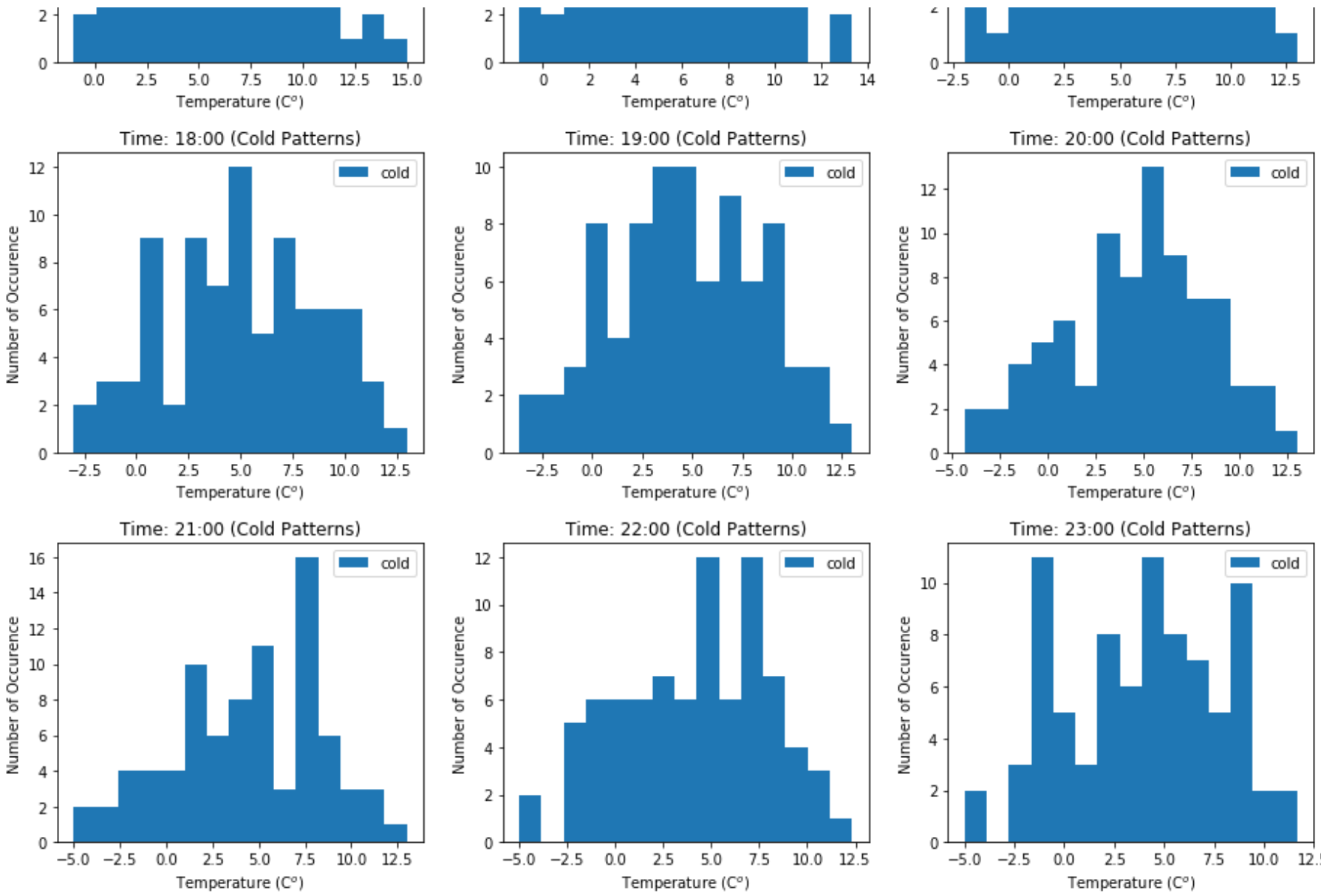

Time: 13:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)


Time: 14:00 (Cold Patterns)
cold
Number of Occurence
Temperature (C°)


Time: 15:00 (Cold Patterns)
cold
Number of Occurence


Time: 16:00 (Cold Patterns)
cold
Number of Occurence


Time: 17:00 (Cold Patterns)
cold
Number of Occurence

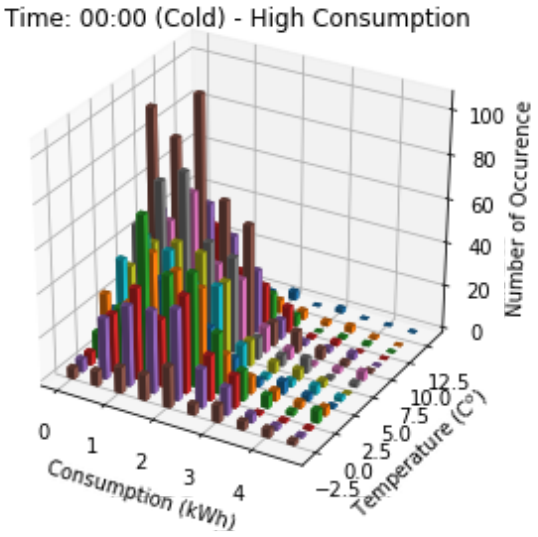


The distribution of days of a week (since it's non-event days, so it's worth checking this property)

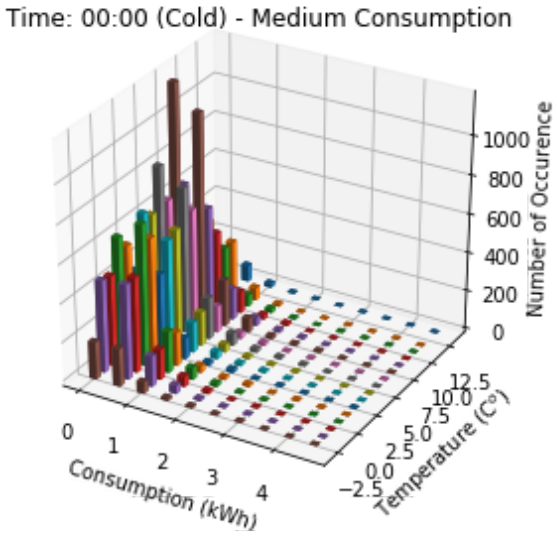
In [390]:
```
# temperature distribution of different hours of a day in a year
fig_all = plt.figure(figsize = (6,4))
ax = fig_all.add_subplot(1, 1, 1)
ax.hist([pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('00:00:00')]['GMT']).dt.dayofweek
+ 1,
         pd.to_datetime(df_wea1h_nf_warm[df_tou1h_nf_warm.GMT.str.contains('00:00:00')]['GMT']).dt.dayofweek
+ 1], bins=13, alpha = 0.5, color = ['blue', 'red'], label = ['cold', 'warm'])
ax.set_title('Distribution of Days' + ' (Cold Patterns)')
ax.legend()
ax.set_xlabel('Days of a Week')
ax.set_ylabel('Number of Occurence')
plt.show()
```

In [18]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
cons_type = ['High Consumption', 'Medium Consumption', 'Low Consumption']
fig_all = plt.figure(figsize = (13,90))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1, projection = '3d')
            t2 = t[t.columns[kmeans.labels_ == sorted_label[k][0]]].copy() # select all data from the n-th cl
uster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            temp_list = list(set(t2['TempC']))
            temp_list.sort(reverse = True)
            for temp in temp_list:
                t3 = t2[t2['TempC'] == temp] # select the data with a specific temperature
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=10, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(10) * temp
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(10) * 0.15
                dy = np.ones(10) * 0.5
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 3 + k + 1
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel(r'Temperature (C$^o$)')
            ax.set_zlabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1, projection = '3d')
            t2 = t[t.columns[kmeans.labels_ == sorted_label[k][0]]].copy() # select all data from the n-th cl
uster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            temp_list = list(set(t2['TempC']))
            temp_list.sort(reverse = True)
            for temp in temp_list:
                t3 = t2[t2['TempC'] == temp] # select the data with a specific temperature
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=10, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(10) * temp
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(10) * 0.15
                dy = np.ones(10) * 0.5
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 3 + k + 1
            ax.set_title('Time: ' +str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel(r'Temperature (C$^o$)')
            ax.set_zlabel('Number of Occurence')
plt.tight_layout()
```

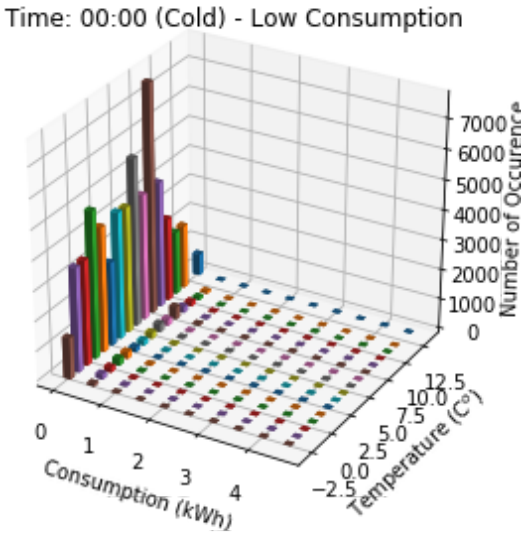
Time: 00:00 (Cold) - High Consumption



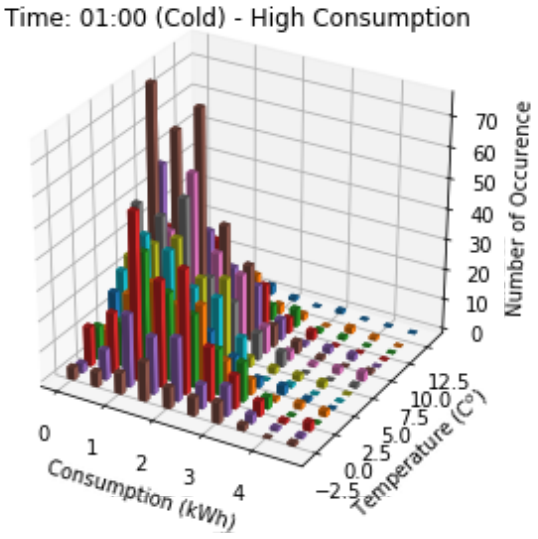
Time: 00:00 (Cold) - Medium Consumption



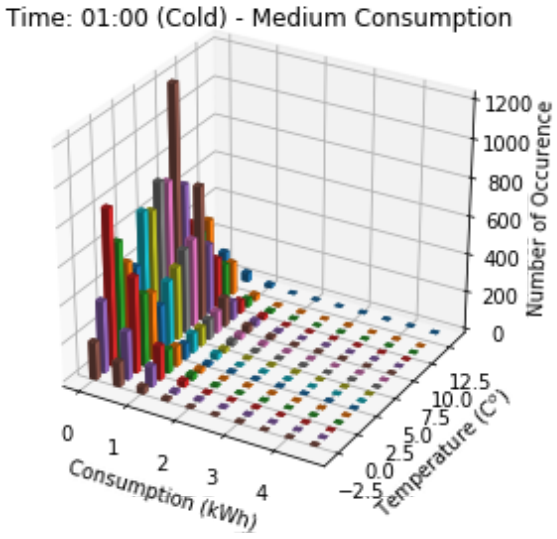
Time: 00:00 (Cold) - Low Consumption



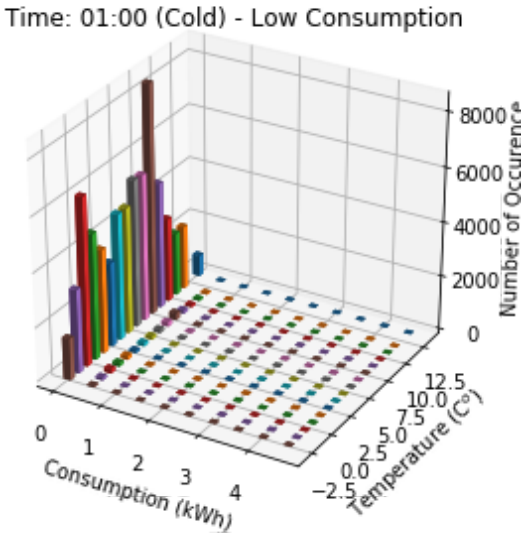
Time: 01:00 (Cold) - High Consumption



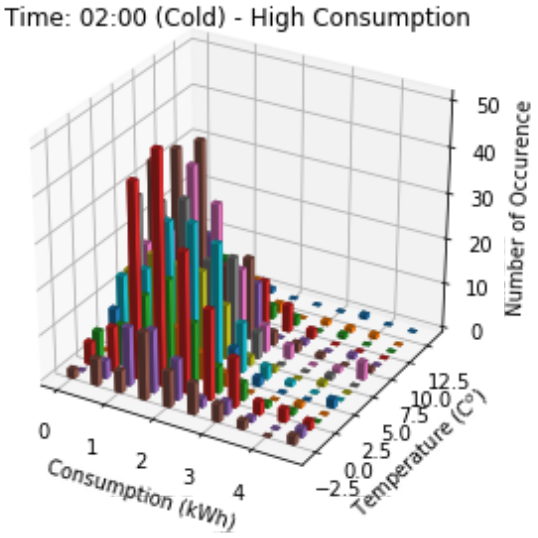
Time: 01:00 (Cold) - Medium Consumption



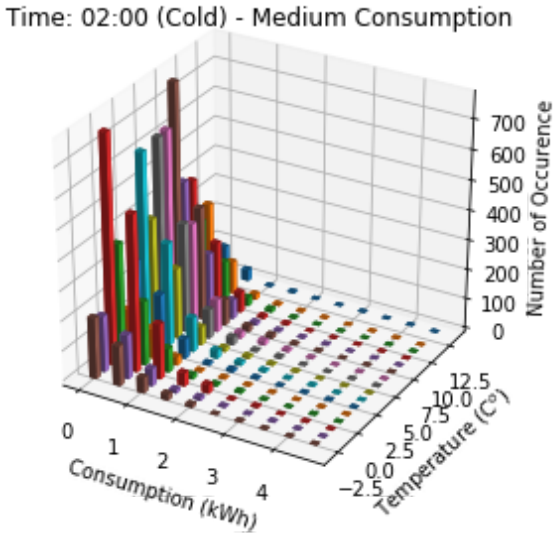
Time: 01:00 (Cold) - Low Consumption



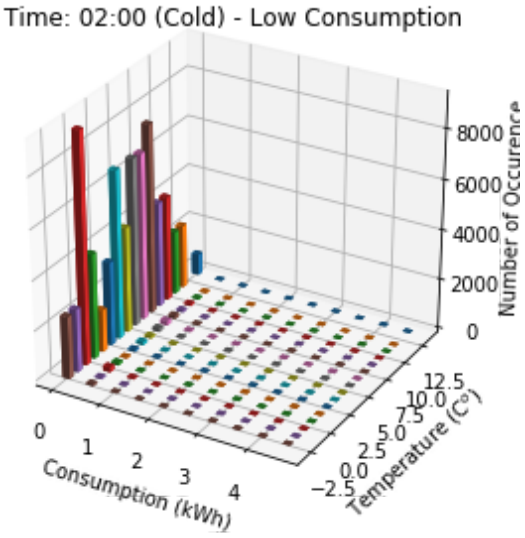
Time: 02:00 (Cold) - High Consumption



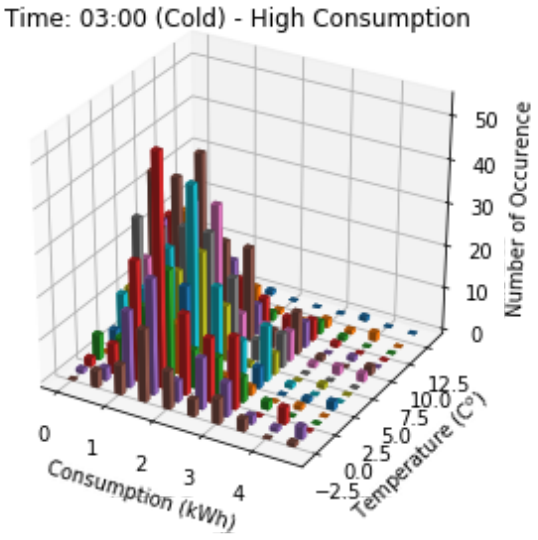
Time: 02:00 (Cold) - Medium Consumption



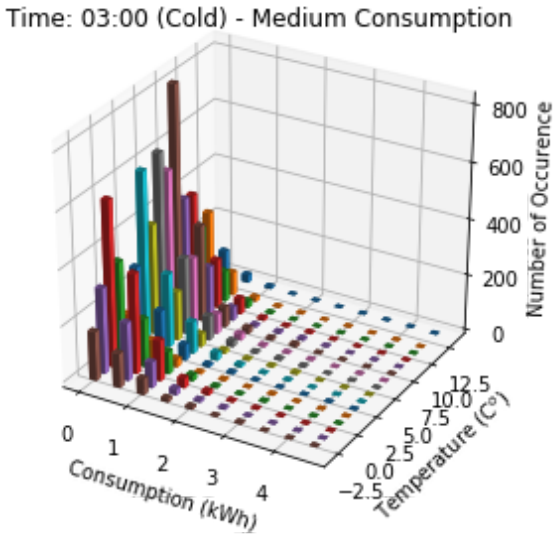
Time: 02:00 (Cold) - Low Consumption



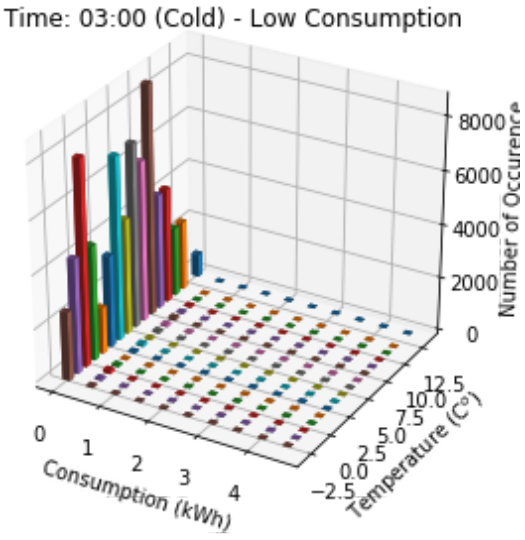
Time: 03:00 (Cold) - High Consumption



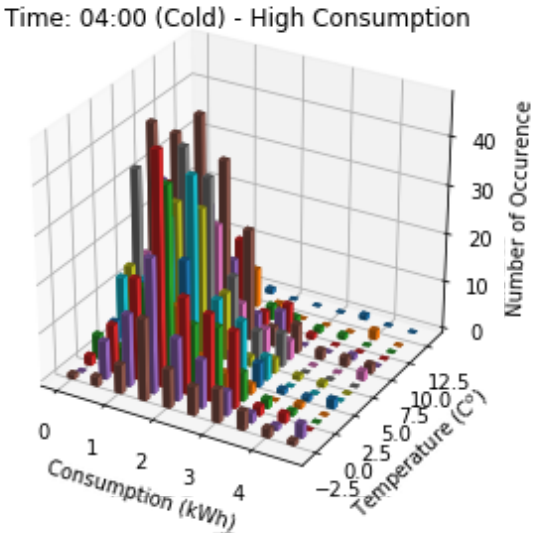
Time: 03:00 (Cold) - Medium Consumption



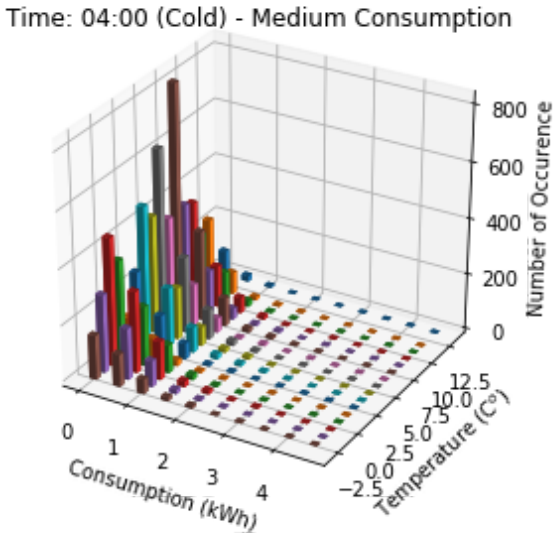
Time: 03:00 (Cold) - Low Consumption



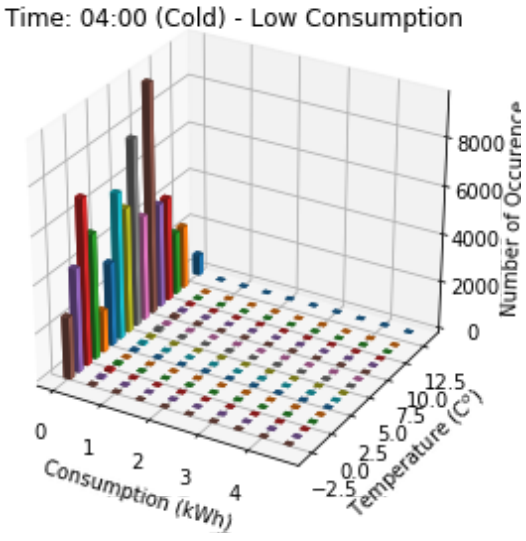
Time: 04:00 (Cold) - High Consumption



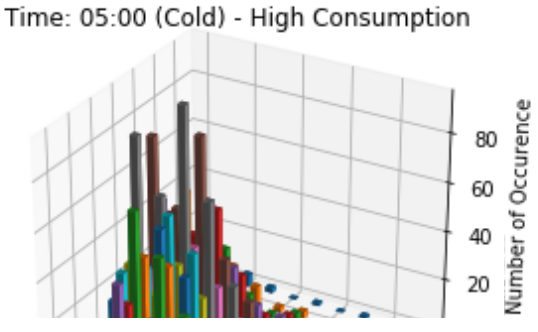
Time: 04:00 (Cold) - Medium Consumption



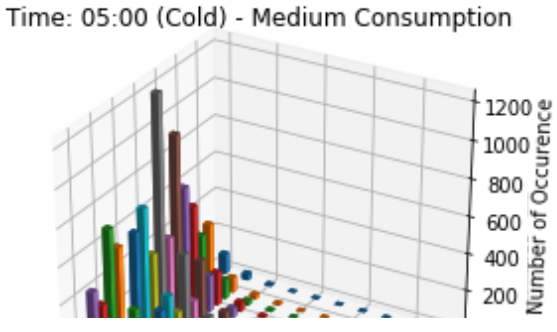
Time: 04:00 (Cold) - Low Consumption



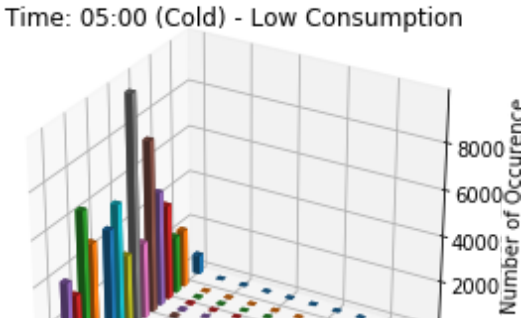
Time: 05:00 (Cold) - High Consumption

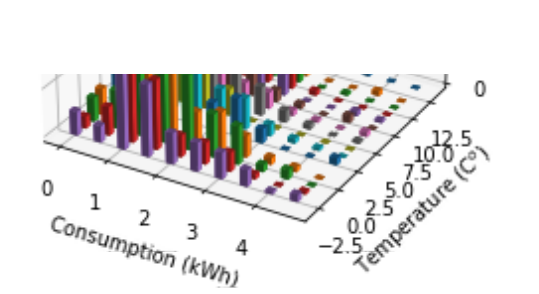


Time: 05:00 (Cold) - Medium Consumption

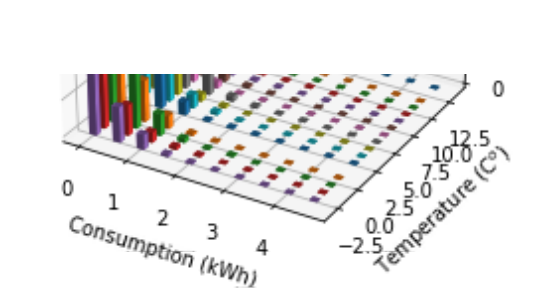


Time: 05:00 (Cold) - Low Consumption

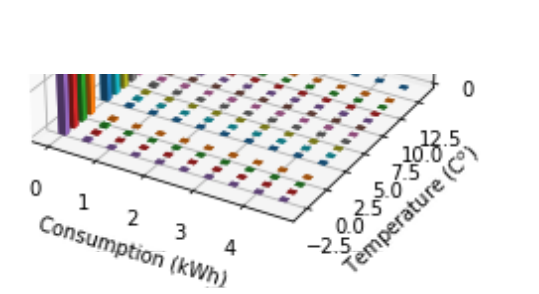




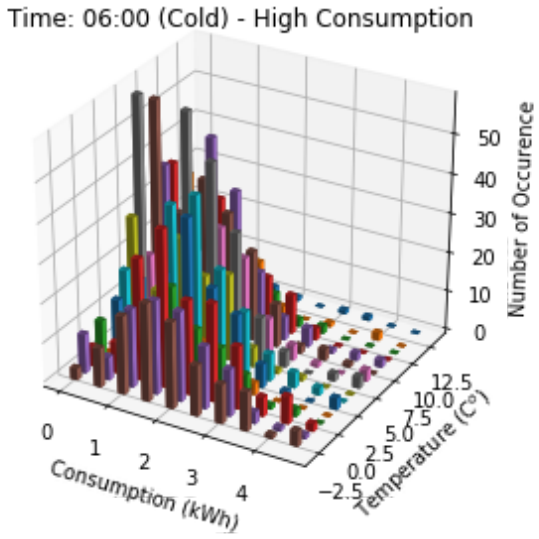
Time: 06:00 (Cold) - High Consumption



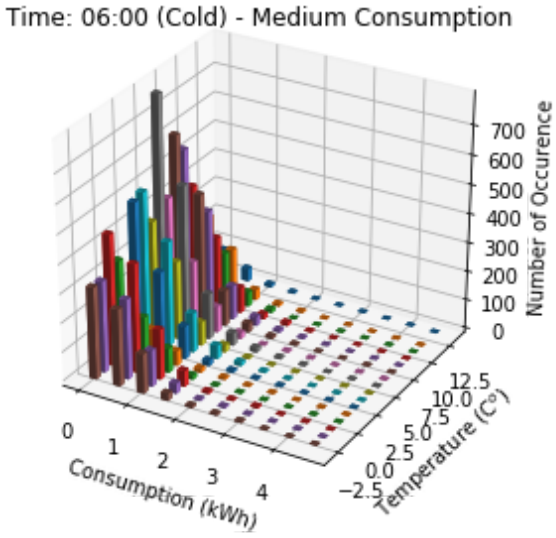
Time: 06:00 (Cold) - Medium Consumption



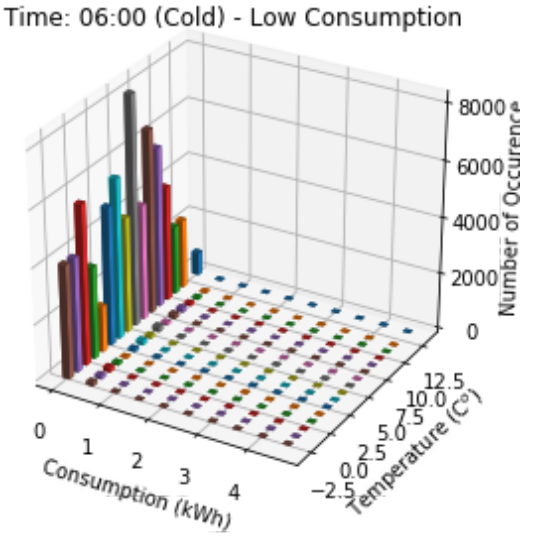
Time: 06:00 (Cold) - Low Consumption



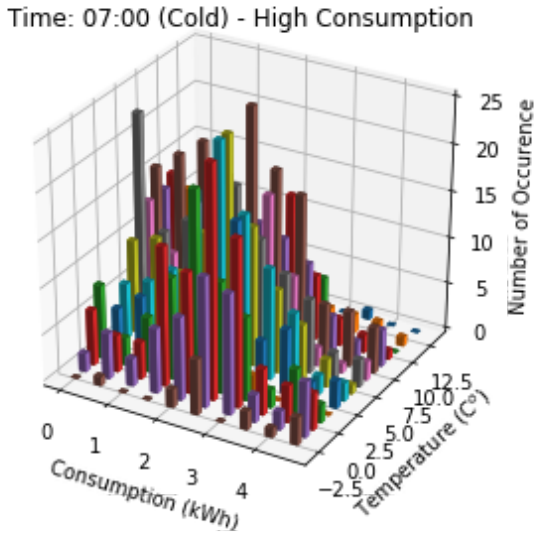
Time: 07:00 (Cold) - High Consumption



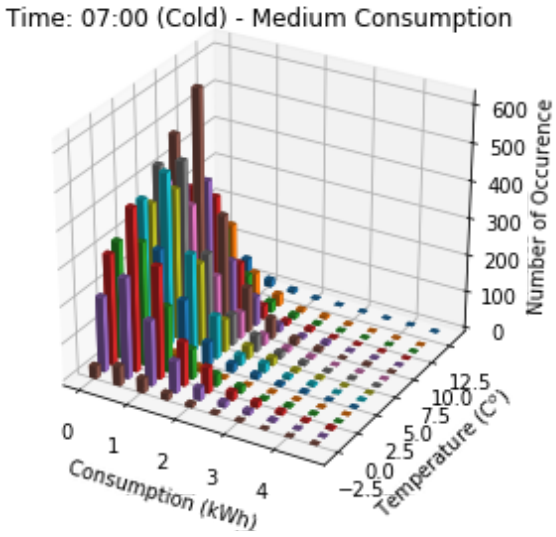
Time: 07:00 (Cold) - Medium Consumption



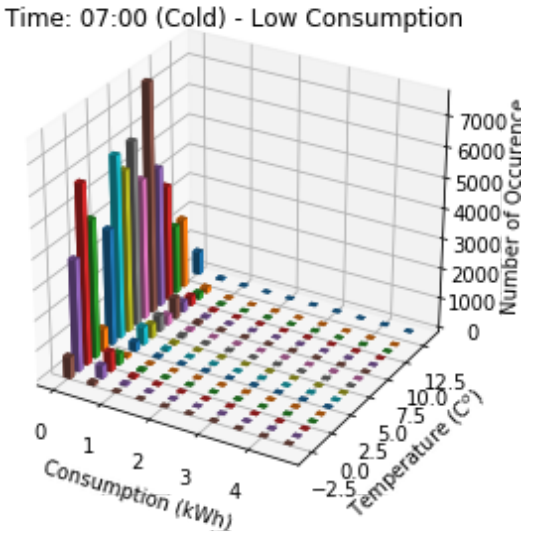
Time: 07:00 (Cold) - Low Consumption



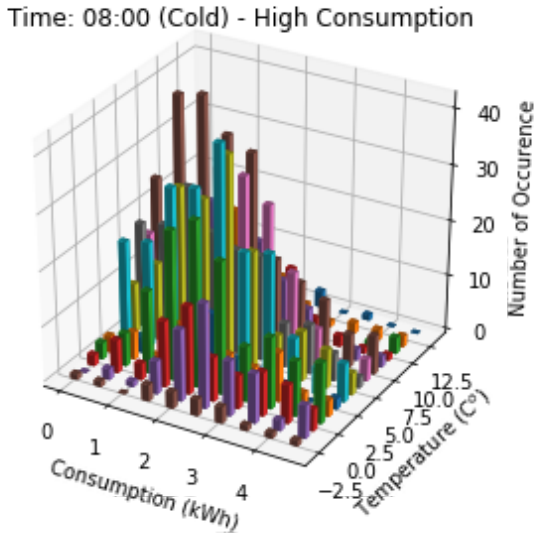
Time: 08:00 (Cold) - High Consumption



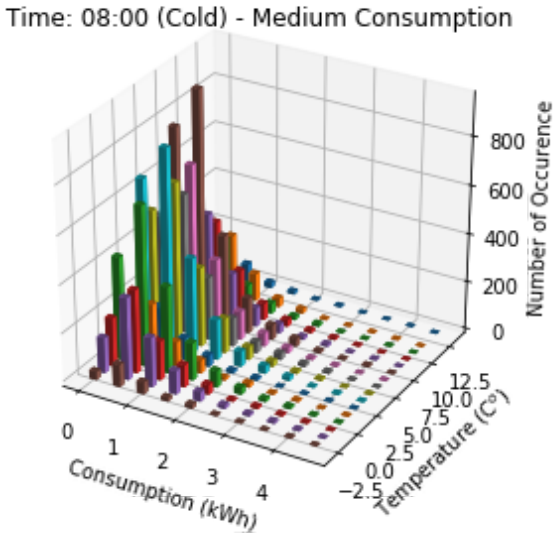
Time: 08:00 (Cold) - Medium Consumption



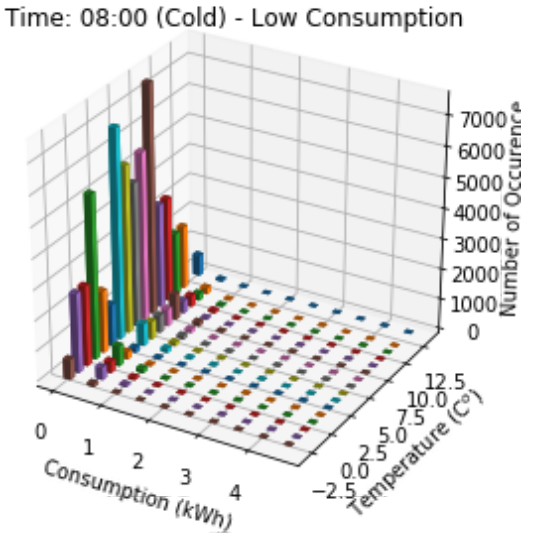
Time: 08:00 (Cold) - Low Consumption



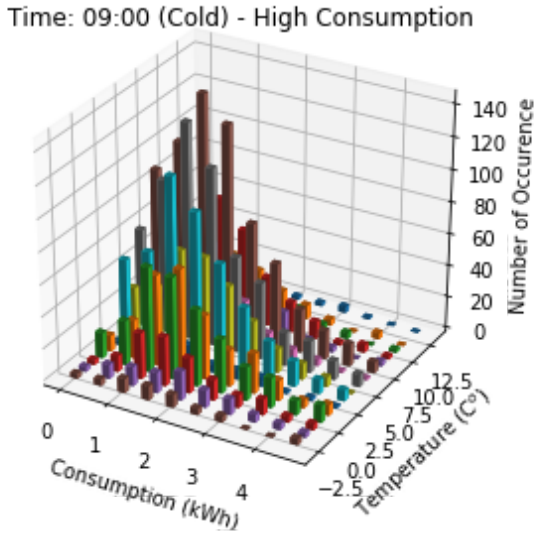
Time: 09:00 (Cold) - High Consumption



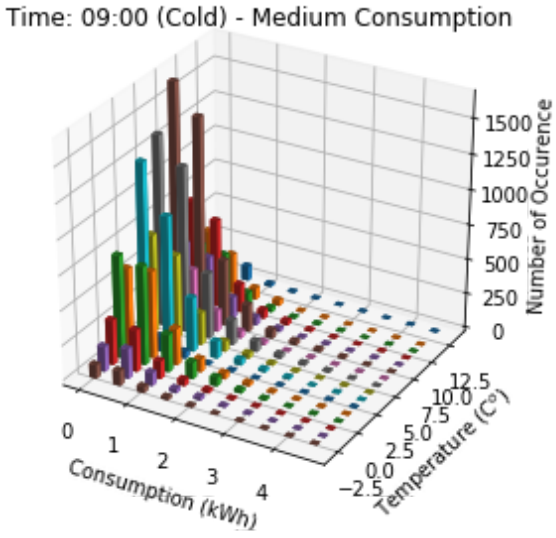
Time: 09:00 (Cold) - Medium Consumption



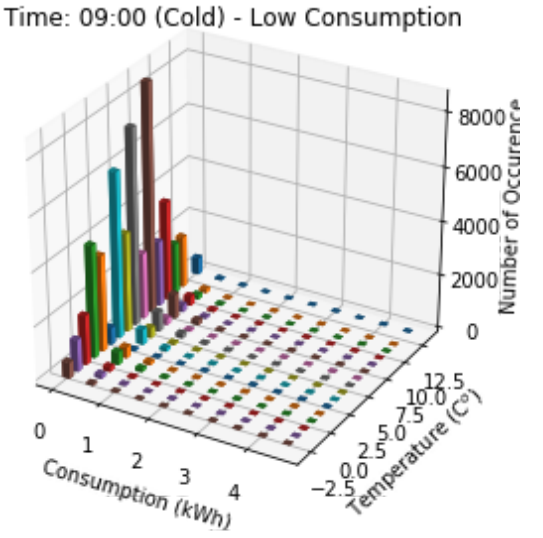
Time: 09:00 (Cold) - Low Consumption



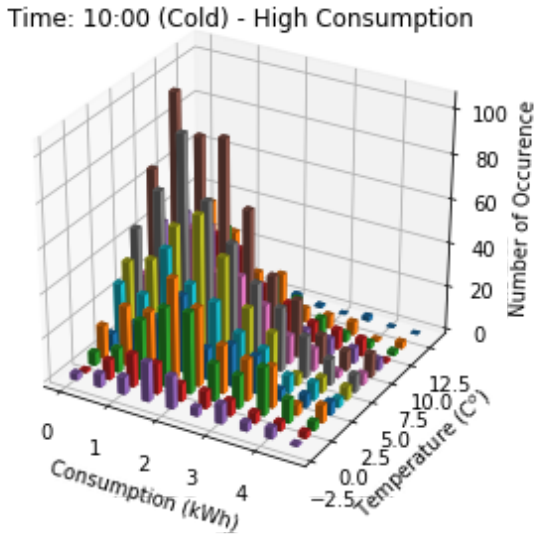
Time: 10:00 (Cold) - High Consumption



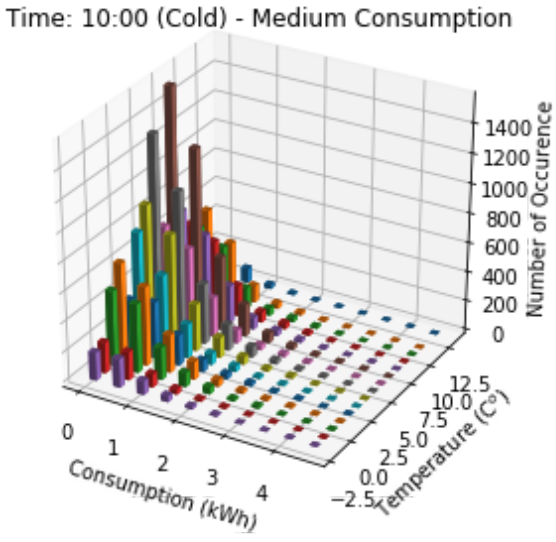
Time: 10:00 (Cold) - Medium Consumption



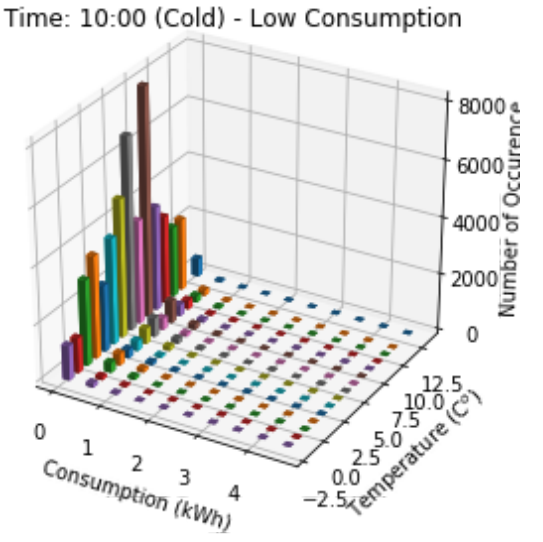
Time: 10:00 (Cold) - Low Consumption



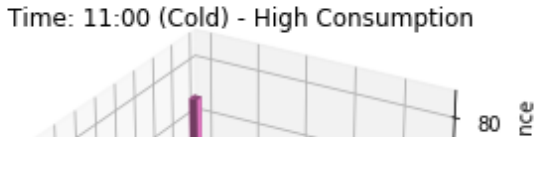
Time: 11:00 (Cold) - High Consumption

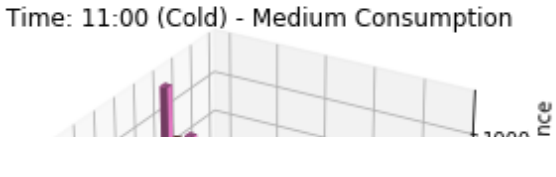


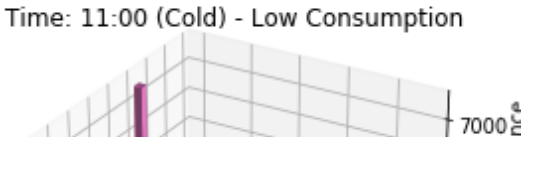
Time: 11:00 (Cold) - Medium Consumption

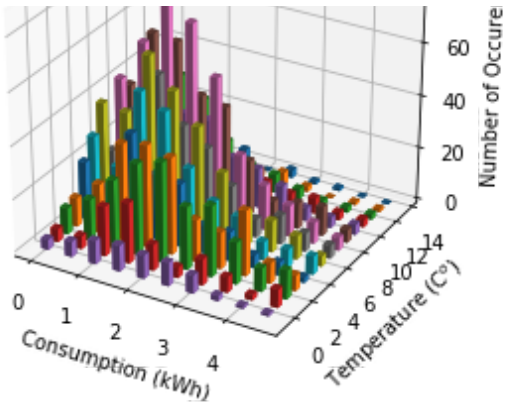


Time: 11:00 (Cold) - Low Consumption

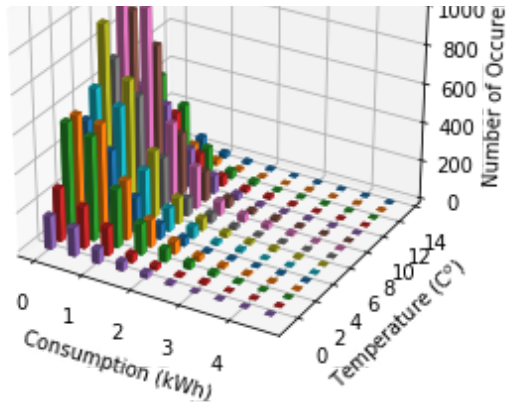




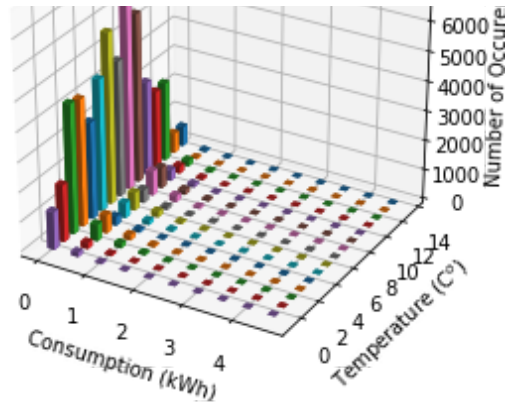




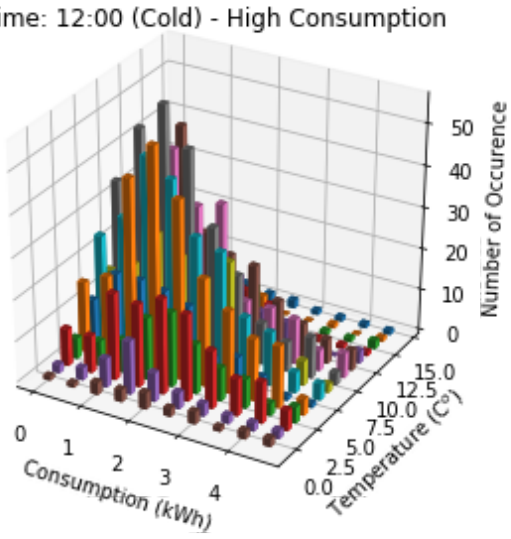
Time: 12:00 (Cold) - High Consumption



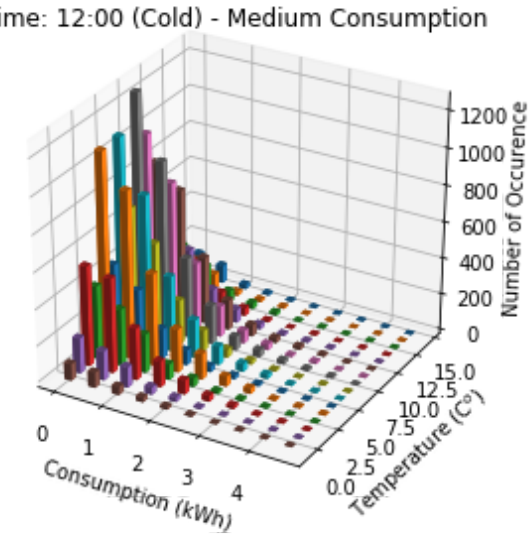
Time: 12:00 (Cold) - Medium Consumption



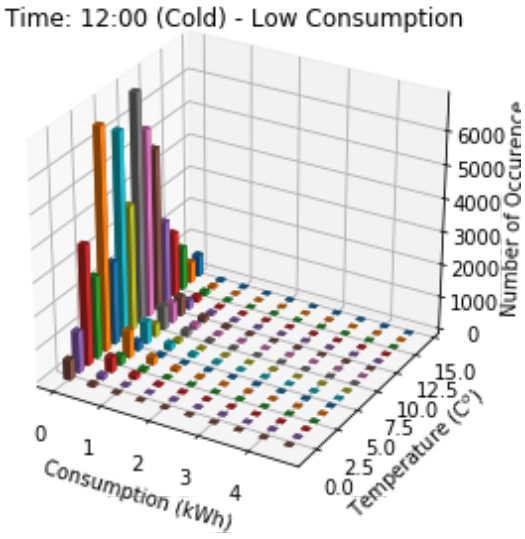
Time: 12:00 (Cold) - Low Consumption



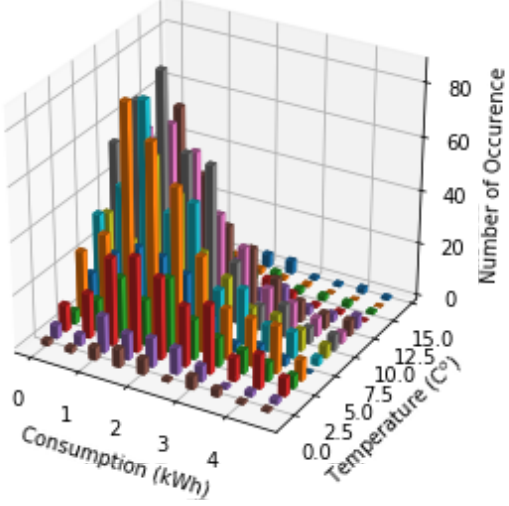
Time: 13:00 (Cold) - High Consumption



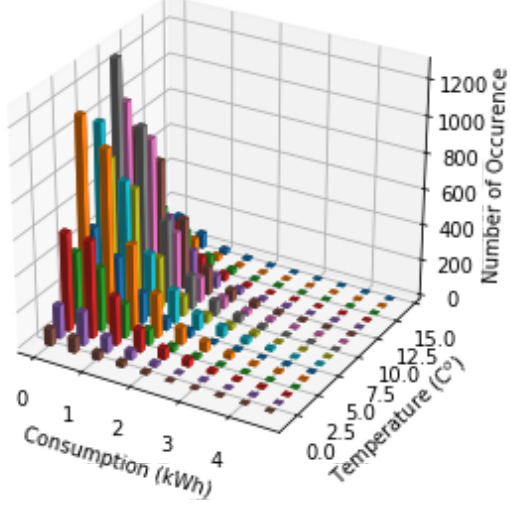
Time: 13:00 (Cold) - Medium Consumption



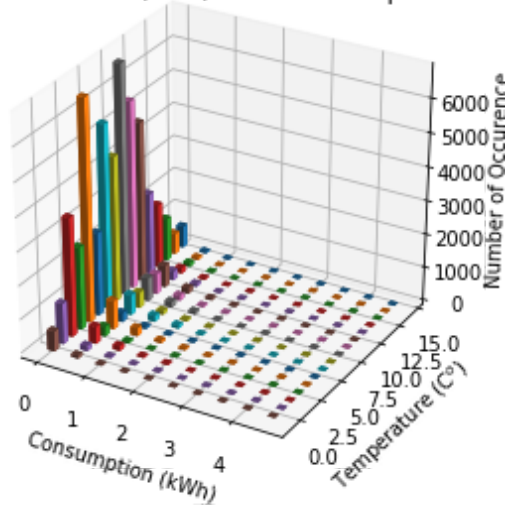
Time: 13:00 (Cold) - Low Consumption



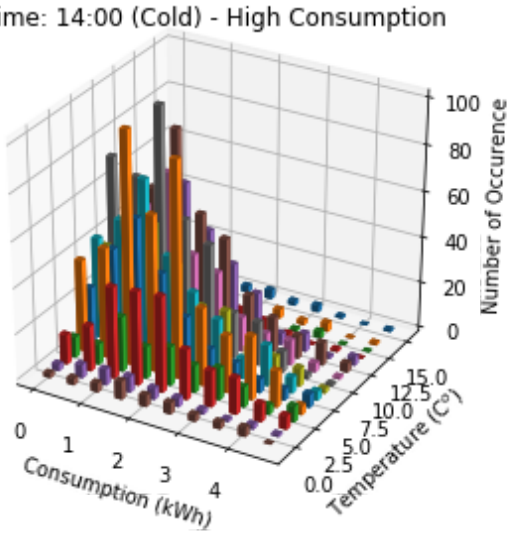
Time: 14:00 (Cold) - High Consumption



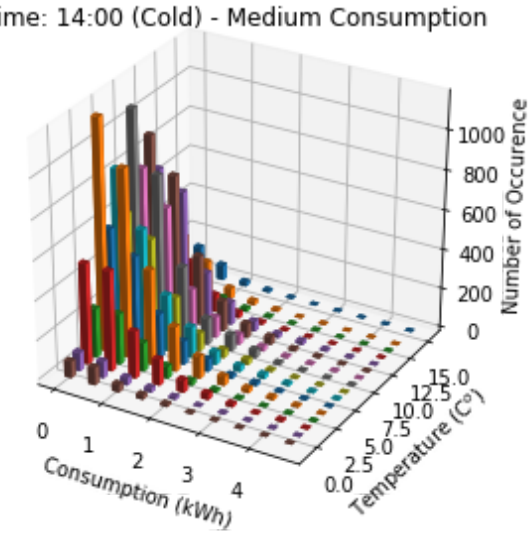
Time: 14:00 (Cold) - Medium Consumption



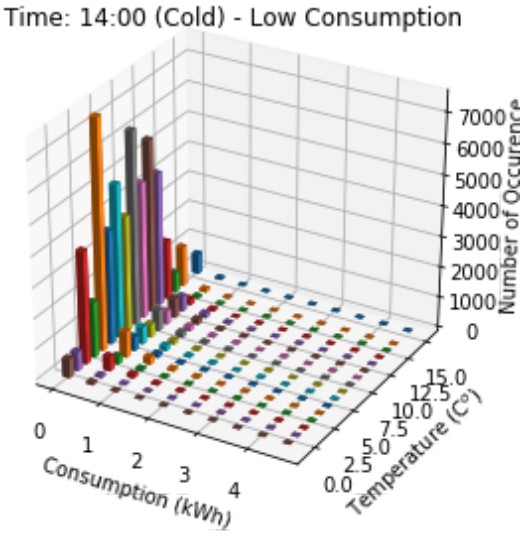
Time: 14:00 (Cold) - Low Consumption



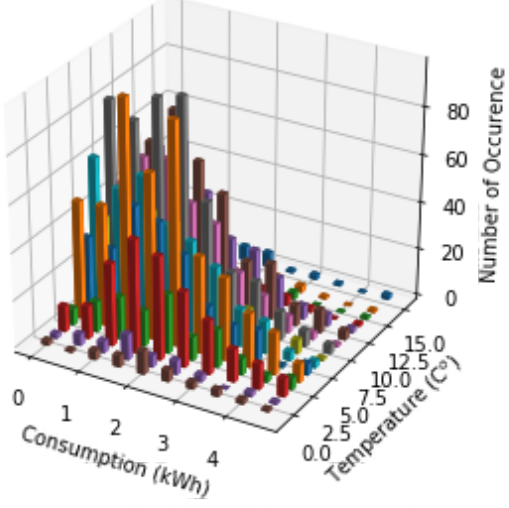
Time: 15:00 (Cold) - High Consumption



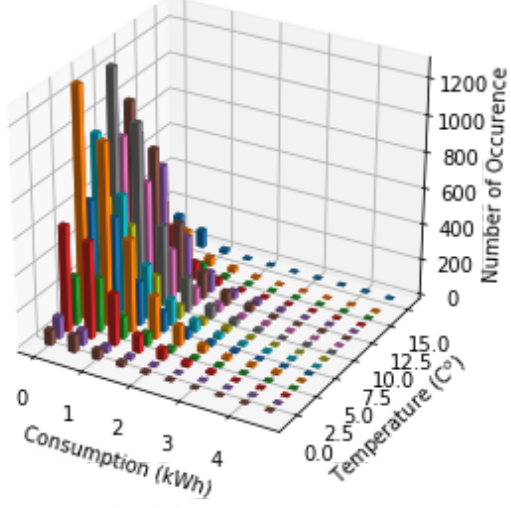
Time: 15:00 (Cold) - Medium Consumption



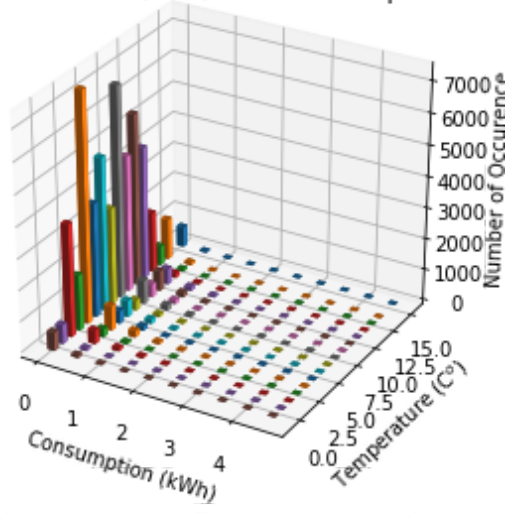
Time: 15:00 (Cold) - Low Consumption



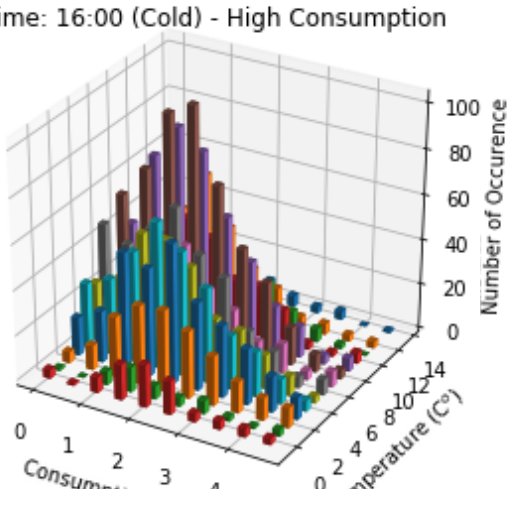
Time: 16:00 (Cold) - High Consumption

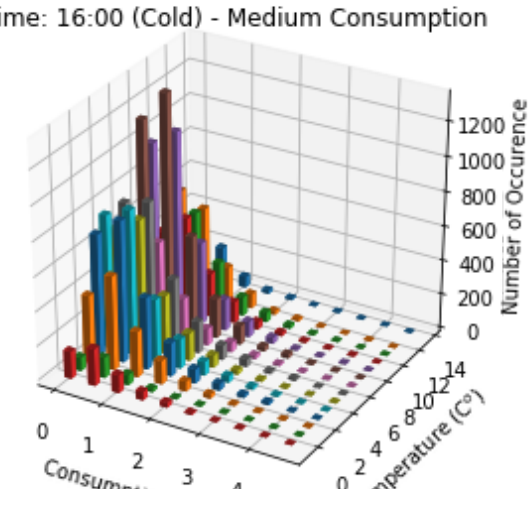


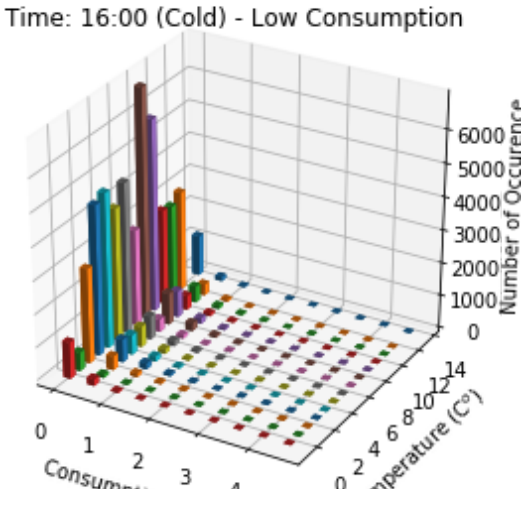
Time: 16:00 (Cold) - Medium Consumption



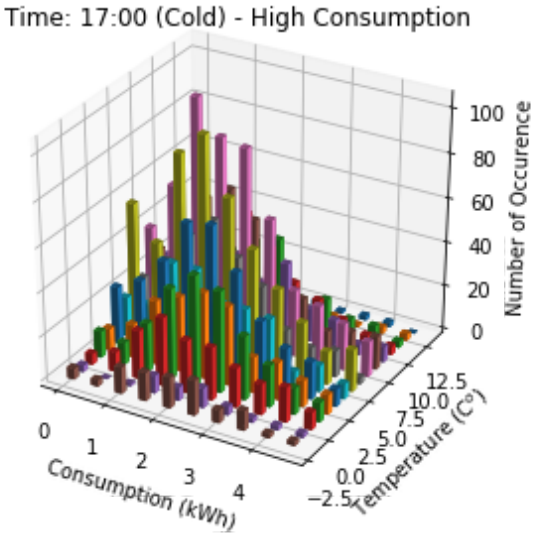
Time: 16:00 (Cold) - Low Consumption



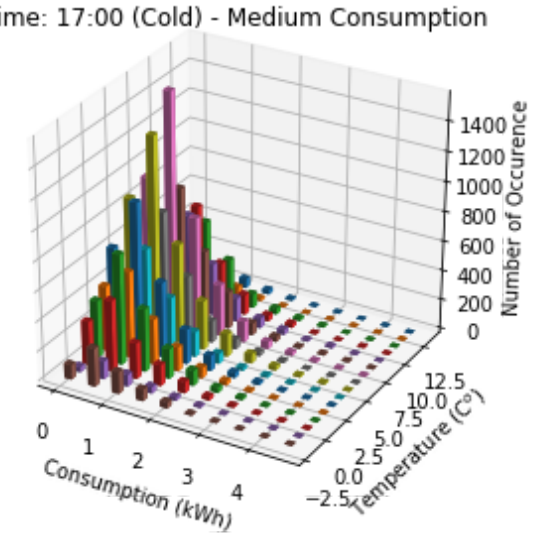




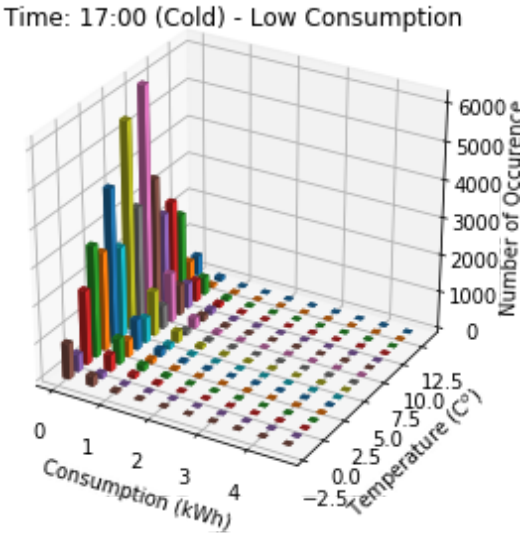
Time: 17:00 (Cold) - High Consumption



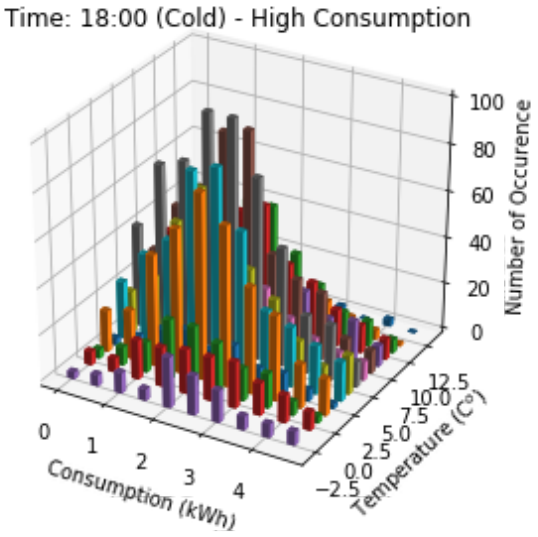
Time: 17:00 (Cold) - Medium Consumption



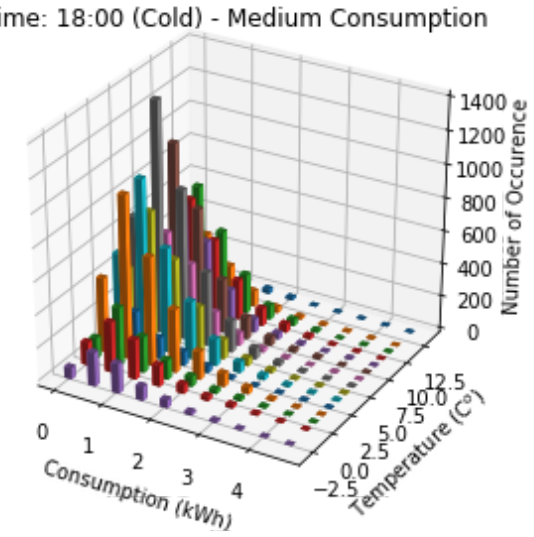
Time: 17:00 (Cold) - Low Consumption



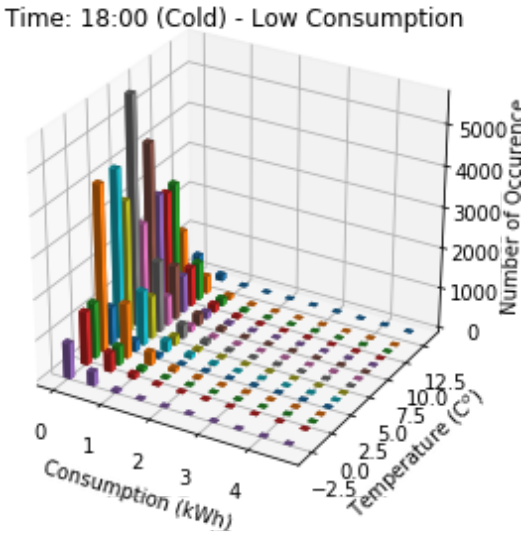
Time: 18:00 (Cold) - High Consumption



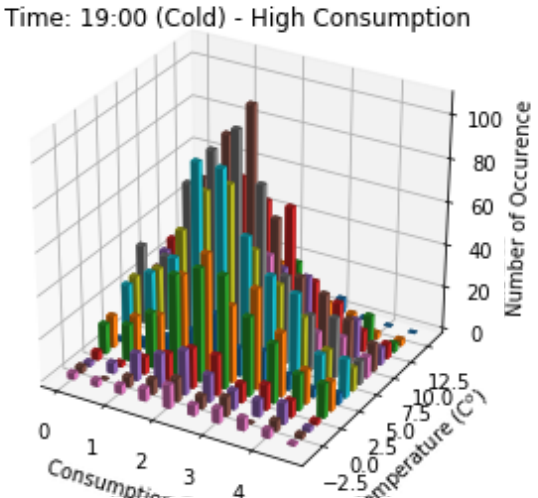
Time: 18:00 (Cold) - Medium Consumption



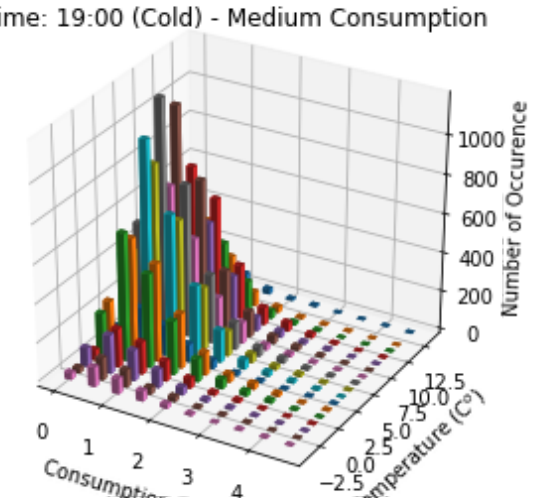
Time: 18:00 (Cold) - Low Consumption



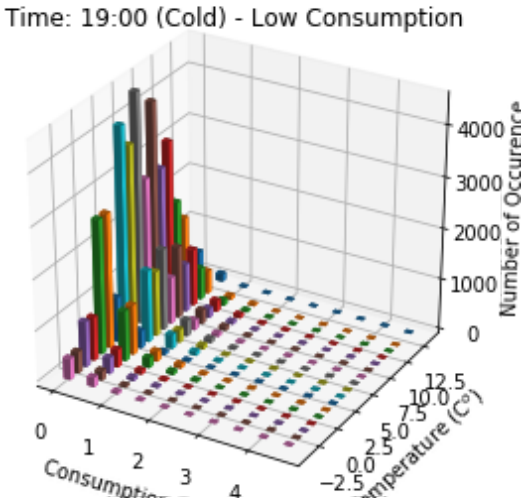
Time: 19:00 (Cold) - High Consumption



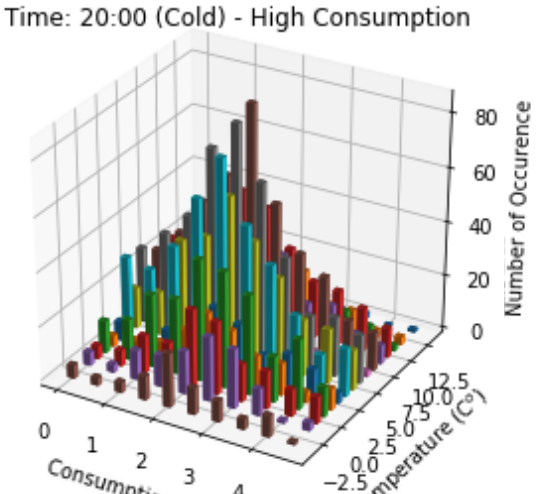
Time: 19:00 (Cold) - Medium Consumption



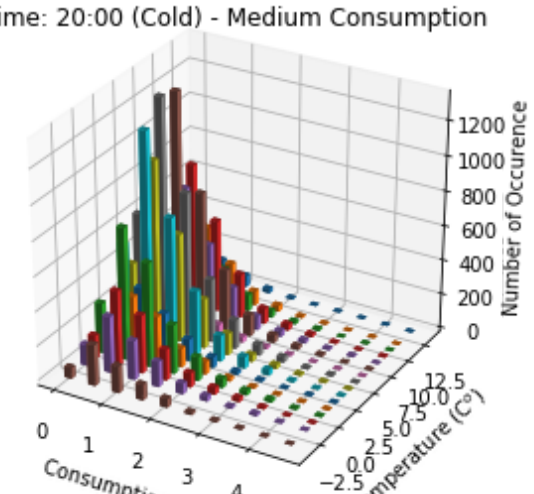
Time: 19:00 (Cold) - Low Consumption



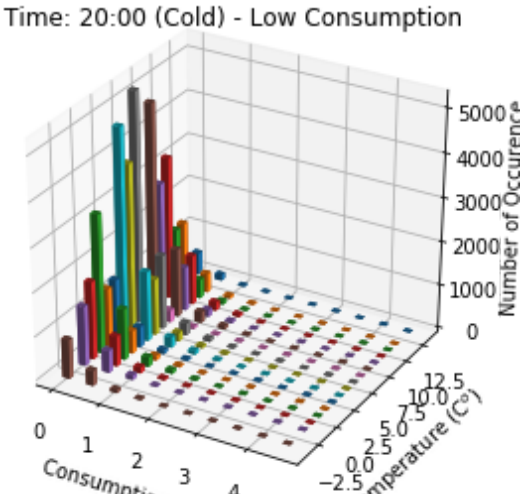
Time: 20:00 (Cold) - High Consumption



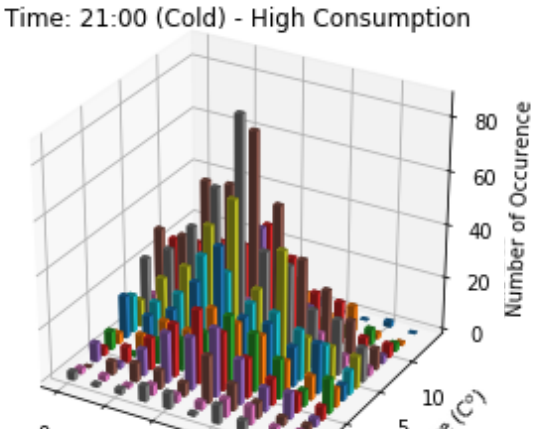
Time: 20:00 (Cold) - Medium Consumption



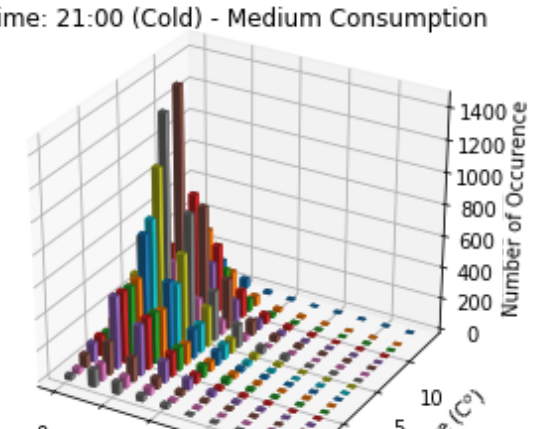
Time: 20:00 (Cold) - Low Consumption



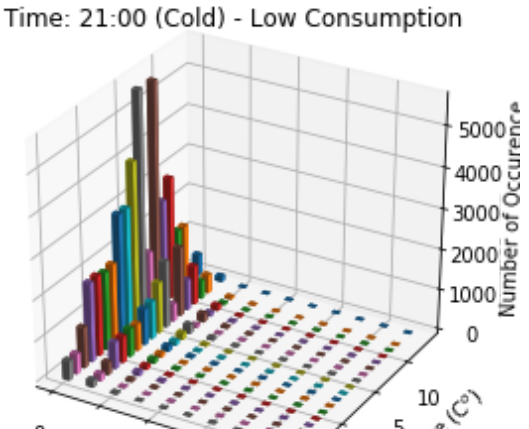
Time: 21:00 (Cold) - High Consumption



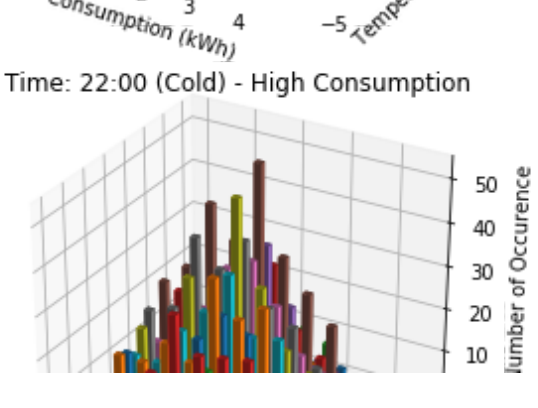
Time: 21:00 (Cold) - Medium Consumption



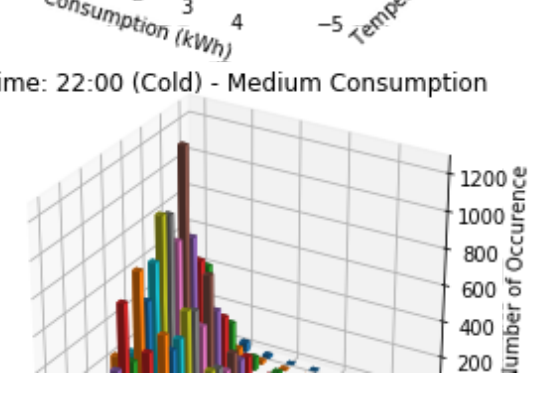
Time: 21:00 (Cold) - Low Consumption



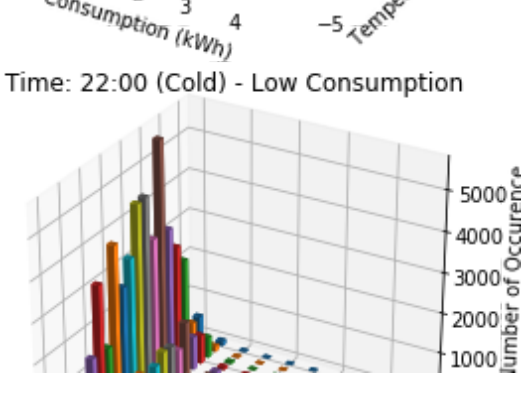
Time: 22:00 (Cold) - High Consumption

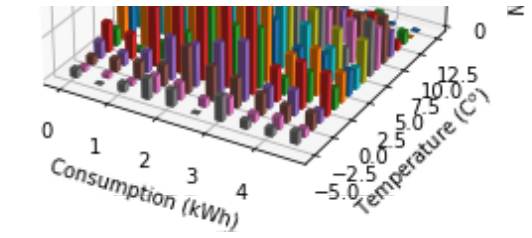


Time: 22:00 (Cold) - Medium Consumption

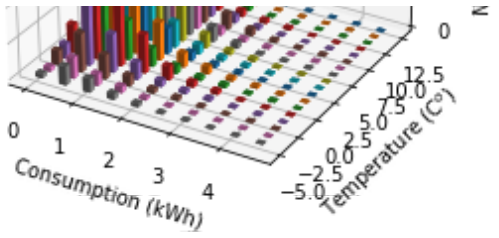


Time: 22:00 (Cold) - Low Consumption

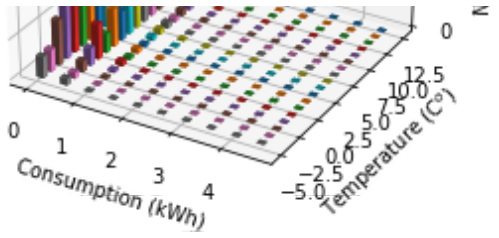




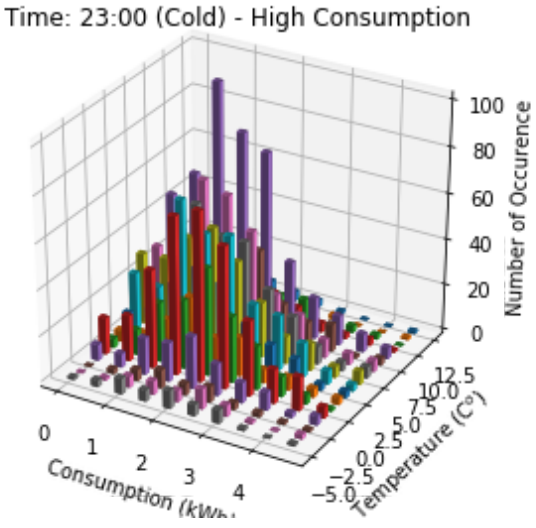
Time: 23:00 (Cold) - High Consumption

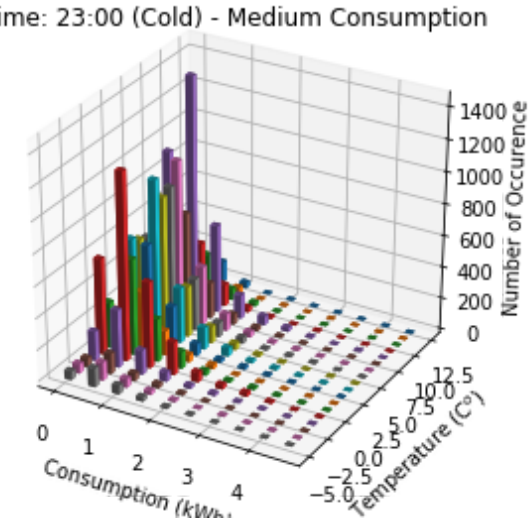


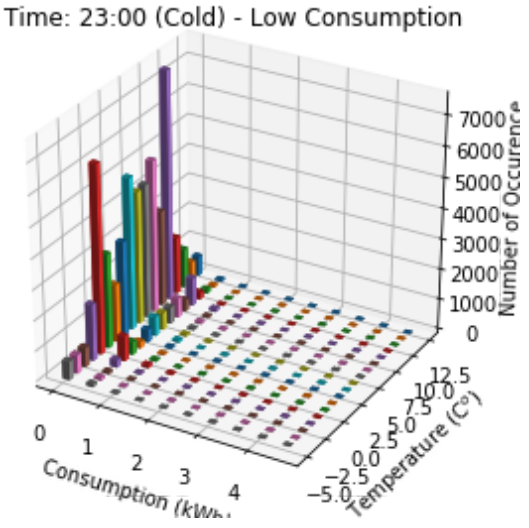
Time: 23:00 (Cold) - Medium Consumption



Time: 23:00 (Cold) - Low Consumption







Most of the above plots show that the highest demand of each group happens around 8-10 celcius degree, except for 2 pm and 3 pm, which have the highest load around 6 celcius degree

In [40]:
```
t
```

Out[40]:

	D0000	D0001	D0002	D0003	D0004	D0005	D0006	D0007	D0008	D0009	...	D1015	D1016	D1017	D1018	D1019	D1020
GMT																	
2013-01-01 00:00:00	1.447	0.429000	0.451	0.155	0.397	0.106	0.307	0.268	0.390	0.123	...	0.048	0.033	0.502	0.117	0.378	1.143
2013-01-02 00:00:00	0.358	0.153000	0.463	0.169	0.356	0.255	0.436	0.364	0.167	0.104	...	0.061	0.027	0.605	0.673	0.348	0.574
2013-01-03 00:00:00	0.142	0.385000	0.463	0.154	0.476	0.425	0.829	0.349	0.111	0.435	...	0.080	0.088	0.272	0.093	0.551	0.335
2013-01-05 00:00:00	0.230	0.137000	0.421	0.156	0.062	0.323	0.498	0.234	0.501	0.202	...	0.045	0.010	0.317	0.096	0.232	0.525
2013-01-06 00:00:00	0.303	0.229000	0.468	0.158	0.075	0.275	0.467	0.313	0.522	0.403	...	0.185	0.010	0.268	0.105	0.425	0.321
2013-01-09 00:00:00	0.086	0.330000	0.250	0.156	0.064	0.397	0.418	0.295	0.512	0.221	...	0.067	0.010	0.454	0.091	0.209	0.481
2013-01-12 00:00:00	0.218	0.317000	0.410	0.149	0.055	0.451	0.410	0.234	0.428	0.285	...	0.083	0.010	0.369	0.089	0.569	0.519
2013-01-14 00:00:00	0.220	0.120000	0.288	0.147	0.055	0.378	0.540	0.185	0.545	0.297	...	0.066	0.611	0.459	0.079	1.062	0.483
2013-01-15 00:00:00	0.218	0.222000	0.292	0.149	0.055	0.385	0.336	0.205	0.573	0.239	...	0.030	0.074	0.394	0.102	1.051	0.356
2013-01-18 00:00:00	0.095	0.193000	0.504	0.144	0.175	0.352	0.447	0.183	0.674	0.185	...	0.042	0.412	0.584	0.100	0.940	0.252
2013-01-22 00:00:00	0.099	0.155000	0.404	0.142	0.057	0.463	0.068	0.201	0.479	0.270	...	0.067	0.327	0.486	0.113	0.880	0.206
2013-01-23 00:00:00	0.100	0.106000	0.380	0.141	0.058	0.394	0.257	0.304	0.439	0.256	...	0.071	0.024	0.383	0.109	1.011	0.239
2013-01-24 00:00:00	0.103	0.215000	0.407	0.144	0.122	0.386	0.155	0.302	0.376	0.378	...	0.040	0.704	0.430	0.150	0.690	0.229
2013-01-26 00:00:00	0.152	0.268000	1.102	0.148	0.059	0.323	0.403	0.169	0.841	0.317	...	0.058	0.019	0.433	0.144	0.956	0.699
2013-01-27 00:00:00	0.076	0.449000	0.291	0.151	0.208	0.307	0.253	0.256	0.538	0.466	...	0.054	0.031	0.269	0.119	0.660	0.403
2013-01-31 00:00:00	0.098	0.163000	0.266	0.152	0.237	0.434	0.125	0.192	0.124	0.224	...	0.029	0.032	0.461	0.179	0.970	0.383
2013-02-01 00:00:00	0.088	0.115000	0.264	0.154	0.291	0.267	0.295	0.150	0.167	0.266	...	0.045	0.537	0.431	0.873	0.611	0.230
2013-02-02 00:00:00	0.227	0.371000	0.327	0.160	0.057	0.369	0.263	0.134	0.420	0.404	...	0.049	0.055	0.240	0.114	0.910	0.361
2013-02-04 00:00:00	0.105	0.099000	0.327	0.160	0.052	0.182	0.816	0.234	0.584	0.247	...	0.044	0.054	0.429	0.107	0.734	0.507
2013-02-06 00:00:00	0.100	0.088000	0.285	0.155	0.062	0.406	0.472	0.263	0.428	0.172	...	0.029	0.034	0.208	0.106	0.491	0.255

	D0000	D0001	D0002	D0003	D0004	D0005	D0006	D0007	D0008	D0009	...	D1015	D1016	D1017	D1018	D1019	D1020
GMT																	
2013-02-12 00:00:00	0.107	0.109000	0.354	0.149	0.056	0.337	0.743	0.238	0.368	0.423	...	0.058	0.021	0.269	0.127	0.804	0.165
2013-02-13 00:00:00	0.226	0.167000	0.406	0.151	0.070	0.400	0.682	0.291	0.501	0.246	...	0.036	0.043	0.646	0.327	0.827	0.249
2013-02-14 00:00:00	0.102	0.246000	0.350	0.155	0.056	0.375	0.410	0.252	0.106	0.256	...	0.061	0.515	0.488	0.103	0.638	0.198
2013-02-16 00:00:00	0.117	0.168000	0.411	0.157	0.249	0.350	0.155	0.278	0.407	0.306	...	0.082	0.370	0.245	0.115	0.680	0.279
2013-02-19 00:00:00	0.159	0.255000	0.481	0.154	0.057	0.453	0.410	0.245	0.428	0.267	...	0.065	0.065	0.376	0.114	0.802	0.158
2013-02-23 00:00:00	0.135	0.113000	0.402	0.145	0.346	0.456	0.667	0.294	0.419	0.338	...	0.037	0.018	0.473	0.139	0.772	0.497
2013-02-24 00:00:00	0.193	0.157000	0.404	0.196	0.100	0.478	0.595	0.273	0.396	0.475	...	0.063	0.059	0.450	0.104	0.663	0.500
2013-02-25 00:00:00	0.111	0.192000	1.046	0.149	0.073	0.161	0.885	0.258	0.536	0.221	...	0.060	0.046	0.246	0.112	1.015	0.211
2013-03-03 00:00:00	0.145	0.293000	0.277	0.201	0.076	0.432	1.003	0.179	0.389	0.223	...	0.129	0.027	0.223	1.074	0.142	0.363
2013-03-04 00:00:00	0.165	0.141000	0.283	0.153	0.058	0.328	0.524	0.209	0.383	0.214	...	0.077	0.014	0.228	0.679	0.540	0.285
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2013-11-12 00:00:00	0.204	0.109000	1.029	0.099	0.028	0.322	0.161	0.241	0.584	0.326	...	0.088	0.006	0.190	0.042	0.423	0.304
2013-11-13 00:00:00	0.121	0.136000	0.243	0.100	0.038	0.394	0.164	0.196	0.603	0.340	...	0.068	0.031	0.196	0.120	0.742	0.462
2013-11-15 00:00:00	0.091	0.206000	0.300	0.097	0.040	0.363	0.155	0.232	0.108	0.294	...	0.054	0.028	0.182	0.746	0.518	0.371
2013-11-16 00:00:00	0.183	0.194000	0.258	0.098	0.119	0.314	0.138	0.161	0.110	0.295	...	0.084	0.050	0.205	0.206	0.598	0.138
2013-11-17 00:00:00	0.089	0.323000	0.174	0.102	0.070	0.396	0.189	0.285	0.664	0.364	...	0.074	0.035	0.265	0.117	0.745	0.534
2013-11-18 00:00:00	0.091	0.223000	0.213	0.101	0.063	0.424	0.380	0.255	0.111	0.451	...	0.058	0.005	0.251	0.118	0.081	0.439
2013-11-19 00:00:00	0.120	0.090000	0.214	0.102	0.079	0.250	0.108	0.292	0.316	0.356	...	0.058	0.049	0.189	0.117	0.139	0.252
2013-11-20 00:00:00	0.181	0.134000	0.286	0.098	0.040	0.445	0.097	0.134	0.825	0.256	...	0.068	0.035	0.241	0.124	0.361	0.156
2013-11-22 00:00:00	0.190	0.245000	0.283	0.098	0.103	0.503	0.871	0.181	0.255	0.261	...	0.078	0.033	0.207	0.109	0.873	0.603
2013-11-23 00:00:00	0.510	0.286000	0.582	0.099	0.067	0.332	0.227	0.138	0.403	0.343	...	0.035	0.034	0.205	0.113	0.483	0.119

	D0000	D0001	D0002	D0003	D0004	D0005	D0006	D0007	D0008	D0009	...	D1015	D1016	D1017	D1018	D1019	D1020
GMT																	
2013-11-24 00:00:00	0.469	0.294000	0.243	0.100	0.025	0.434	0.086	0.039	0.600	0.806	...	0.059	0.046	0.211	0.116	0.640	1.369
2013-11-25 00:00:00	0.083	0.168000	0.538	0.100	0.013	0.185	0.110	0.043	0.392	0.454	...	0.065	0.001	0.196	0.125	0.721	0.546
2013-12-02 00:00:00	0.106	0.358000	0.427	0.100	0.085	0.228	0.384	0.245	0.312	0.293	...	0.048	0.045	0.212	0.203	0.324	0.350
2013-12-03 00:00:00	0.095	0.152000	0.726	0.100	0.017	0.371	0.331	0.217	0.368	0.268	...	0.039	0.019	0.199	0.154	0.417	0.546
2013-12-05 00:00:00	0.103	0.160000	1.086	0.099	0.024	0.398	0.595	0.305	0.491	0.268	...	0.063	0.062	0.285	0.306	0.392	0.602
2013-12-06 00:00:00	0.094	0.163000	0.324	0.098	0.082	0.125	0.470	0.082	0.518	0.267	...	0.053	0.011	0.227	0.137	0.687	0.617
2013-12-09 00:00:00	0.092	0.193829	0.692	0.101	0.056	0.402	0.446	0.199	0.372	0.245	...	0.055	0.000	0.206	0.111	0.266	0.439
2013-12-10 00:00:00	0.465	0.148000	0.240	0.101	0.013	0.349	0.485	0.272	0.469	0.324	...	0.061	0.016	0.290	0.115	0.607	0.502
2013-12-11 00:00:00	0.116	0.212000	0.324	0.097	0.016	0.480	0.429	0.199	0.651	0.271	...	0.082	0.033	0.203	0.150	0.972	0.255
2013-12-14 00:00:00	0.477	0.177000	0.216	0.099	0.035	0.557	0.265	0.312	1.049	0.255	...	0.080	0.035	0.203	0.108	0.242	0.237
2013-12-17 00:00:00	0.172	0.215000	0.206	0.104	0.022	0.424	0.625	0.258	0.845	0.274	...	0.055	0.028	0.228	0.125	0.509	0.300
2013-12-18 00:00:00	0.091	0.292000	0.252	0.167	0.022	0.147	0.208	0.213	0.900	0.264	...	0.069	0.000	0.259	0.125	0.462	0.495
2013-12-21 00:00:00	0.292	0.302000	0.169	0.100	0.035	0.464	0.428	0.241	0.742	0.254	...	0.073	0.015	0.230	0.270	0.330	0.400
2013-12-22 00:00:00	0.289	0.212000	0.455	0.099	0.124	0.285	0.270	0.259	0.461	0.360	...	0.070	0.001	0.319	0.107	0.300	0.614
2013-12-23 00:00:00	0.473	0.080000	0.782	0.100	0.103	0.453	0.173	0.137	0.242	0.647	...	0.055	0.006	0.179	0.787	0.449	0.480
2013-12-24 00:00:00	0.326	0.076000	0.302	0.099	0.017	0.507	0.257	0.315	0.857	0.260	...	0.054	0.000	0.244	0.114	0.769	0.591
2013-12-25 00:00:00	0.632	0.371000	0.636	0.099	0.283	0.265	0.434	0.216	0.472	0.678	...	0.034	0.020	0.259	0.113	0.601	0.535
2013-12-27 00:00:00	0.318	0.397000	0.279	0.099	0.050	0.342	0.550	0.244	0.473	0.103	...	0.050	0.035	0.247	0.122	0.441	0.647
2013-12-28 00:00:00	0.298	0.110000	0.284	0.099	0.150	0.359	0.793	0.242	0.647	1.235	...	0.053	0.026	0.183	0.235	0.451	2.069
2013-12-31 00:00:00	0.335	0.107000	0.415	0.101	0.295	0.338	0.624	0.139	0.410	0.550	...	0.078	0.033	0.194	0.120	0.742	0.341

83 rows × 1025 columns

In [44]:
```
i
```

Out[44]:
```
23
```

In [45]:
```
pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('23:00:00')]['GMT']).dt.dayofweek
```

Out[45]:
```
23      1
47      2
71      3
119     5
143     6
215     2
287     5
335     0
359     1
431     4
527     1
551     2
575     3
623     5
647     6
743     3
767     4
791     5
839     0
887     2
1031    1
1055    2
1079    3
1127    5
1199    1
1295    5
1319    6
1343    0
1487    6
1511    0
       ..
7583    1
7607    2
7655    4
7679    5
7703    6
7727    0
7751    1
7775    2
7823    4
7847    5
7871    6
7895    0
8063    0
8087    1
8135    3
8159    4
8231    0
8255    1
8279    2
8351    5
8423    1
8447    2
8519    5
8543    6
8567    0
8591    1
8615    2
8663    4
8687    5
8759    1
Name: GMT, Length: 83, dtype: int64
```

In [47]:
```
t2['DayofWeek'] = np.array(pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('23:00:00')]['GM
T']).dt.dayofweek)
t2
```

Out[47]:

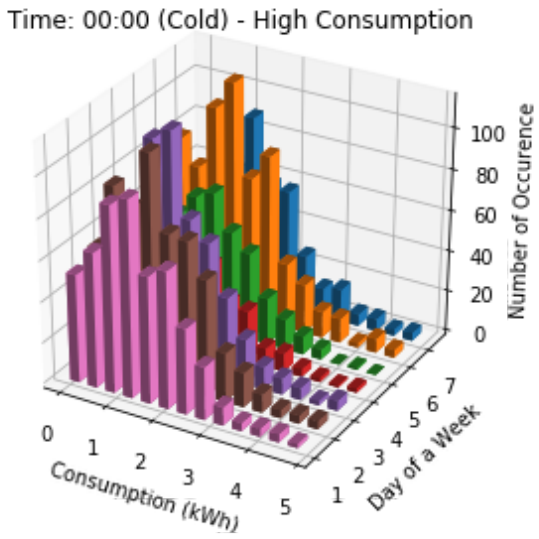
	GMT	D0000	D0001	D0003	D0004	D0005	D0007	D0008	D0009	D0010	...	D1015	D1016	D1017	D1018	D1020	D1021	D102
0	2013-01-01 23:00:00	0.323	0.269	0.241	0.349	0.293	0.495	0.533	0.110	0.019	...	0.031	0.121	0.656	0.294	0.715	0.102	0.337
1	2013-01-02 23:00:00	0.232	0.452	0.153	0.500	0.458	0.357	0.268	0.327	0.028	...	0.036	0.166	0.394	0.087	0.400	0.142	0.348
2	2013-01-03 23:00:00	0.315	0.375	0.156	0.502	0.352	0.402	0.457	0.655	0.028	...	0.047	0.072	0.420	0.085	1.110	0.128	0.373
3	2013-01-05 23:00:00	0.064	0.403	0.206	0.263	0.288	0.457	0.463	0.624	0.024	...	0.080	0.136	0.368	0.083	0.455	0.502	0.357
4	2013-01-06 23:00:00	0.297	0.333	0.157	0.233	0.495	0.194	0.376	0.592	0.024	...	0.072	0.204	0.463	0.083	0.352	0.152	0.365
5	2013-01-09 23:00:00	0.164	0.245	0.153	0.057	0.326	0.220	0.437	0.230	0.020	...	0.033	0.048	0.439	0.091	0.365	0.110	0.351
6	2013-01-12 23:00:00	0.262	0.440	0.168	0.255	0.157	0.259	0.424	0.405	0.027	...	0.045	0.521	0.482	0.081	0.685	0.127	0.363
7	2013-01-14 23:00:00	1.416	0.479	0.147	0.064	0.384	0.239	0.501	0.291	0.018	...	0.107	0.271	0.418	0.086	0.355	0.136	0.350
8	2013-01-15 23:00:00	0.098	0.305	0.145	0.056	0.407	0.229	0.396	0.218	0.025	...	0.082	0.074	0.598	0.080	0.554	0.289	0.357
9	2013-01-18 23:00:00	0.299	0.367	0.141	0.265	0.459	0.219	0.447	0.250	0.022	...	0.057	0.780	0.460	0.100	0.496	0.186	0.341
10	2013-01-22 23:00:00	0.103	0.398	0.141	0.229	0.491	0.381	0.617	0.265	0.026	...	0.056	0.742	0.419	0.171	0.381	0.135	0.418
11	2013-01-23 23:00:00	0.126	0.418	0.143	0.224	0.380	0.694	0.425	0.535	0.030	...	0.067	0.054	0.451	0.364	0.253	0.113	0.348
12	2013-01-24 23:00:00	0.103	0.458	0.152	0.114	0.399	0.249	0.536	0.330	0.015	...	0.046	0.702	0.514	0.116	0.320	0.163	0.423
13	2013-01-26 23:00:00	0.215	0.303	0.208	0.303	0.389	0.257	0.361	0.499	0.021	...	0.055	0.635	0.297	0.118	0.559	0.119	0.432
14	2013-01-27 23:00:00	0.156	0.257	0.144	0.128	0.168	0.145	0.447	0.508	0.023	...	0.031	0.053	0.453	1.516	0.573	0.124	0.355
15	2013-01-31 23:00:00	0.089	0.302	0.154	0.506	0.362	0.143	0.568	0.449	0.023	...	0.072	0.247	0.478	0.572	0.201	0.213	0.366
16	2013-02-01 23:00:00	0.268	0.397	0.160	0.186	0.347	0.213	0.434	0.395	0.031	...	0.063	0.660	0.414	0.120	0.380	0.166	0.375
17	2013-02-02 23:00:00	0.285	0.483	0.158	0.230	0.579	0.284	0.539	1.603	0.030	...	0.062	0.130	0.604	1.494	0.377	0.217	0.340
18	2013-02-04 23:00:00	0.166	0.242	0.156	0.252	0.392	0.290	0.472	0.291	0.020	...	0.050	0.058	0.568	0.116	0.437	0.169	0.366
19	2013-02-06 23:00:00	0.158	0.384	0.154	0.100	0.365	0.137	0.518	0.234	0.018	...	0.078	0.435	0.445	0.119	0.842	0.163	0.368
20	2013-02-12 23:00:00	0.206	0.320	0.150	0.070	0.424	0.258	0.387	0.943	0.018	...	0.091	0.666	0.636	0.833	0.575	0.139	0.365

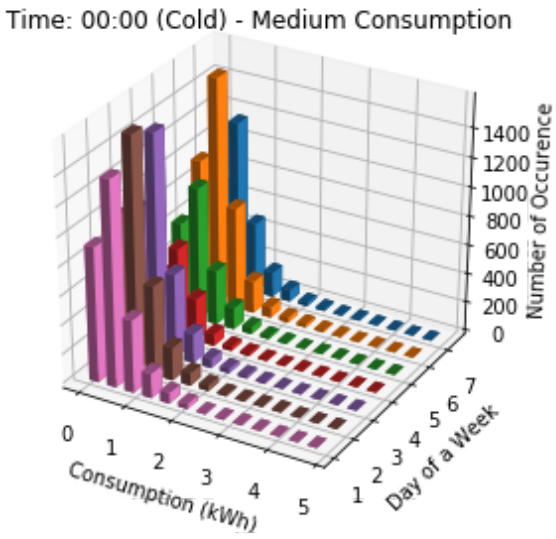
	GMT	D0000	D0001	D0003	D0004	D0005	D0007	D0008	D0009	D0010	...	D1015	D1016	D1017	D1018	D1020	D1021	D102
21	2013-02-13 23:00:00	0.119	0.196	0.154	0.115	0.354	0.292	0.238	0.284	0.025	...	0.030	0.728	0.479	0.117	0.183	0.214	0.355
22	2013-02-14 23:00:00	0.586	0.248	0.154	0.061	0.147	0.263	0.395	0.375	0.026	...	0.075	0.475	0.537	0.109	0.464	0.584	0.368
23	2013-02-16 23:00:00	0.261	0.077	0.245	0.218	0.144	0.326	0.368	0.421	0.022	...	0.068	0.529	0.413	0.110	0.633	0.425	0.353
24	2013-02-19 23:00:00	0.224	0.241	0.158	0.276	0.368	0.397	0.401	0.513	0.022	...	0.056	0.059	0.526	0.422	0.661	0.152	0.350
25	2013-02-23 23:00:00	0.330	0.298	0.236	0.355	0.470	0.305	0.408	0.499	0.019	...	0.060	0.060	0.579	0.088	0.698	0.122	0.329
26	2013-02-24 23:00:00	0.099	0.292	0.150	0.223	0.184	0.250	0.443	0.237	0.017	...	0.036	0.058	0.512	0.113	0.173	0.131	0.338
27	2013-02-25 23:00:00	0.292	0.293	0.148	0.082	0.448	0.270	0.458	0.349	0.016	...	0.056	0.058	0.401	0.114	0.389	0.123	0.444
28	2013-03-03 23:00:00	0.299	0.283	0.153	0.061	0.162	0.238	0.386	0.205	0.000	...	0.042	0.054	0.237	1.215	0.210	0.081	0.386
29	2013-03-04 23:00:00	0.149	0.245	0.154	0.151	0.392	0.239	0.387	0.188	0.002	...	0.042	0.059	0.210	1.040	0.265	0.110	0.293
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53	2013-11-12 23:00:00	0.189	0.231	0.098	0.135	0.428	0.261	0.249	0.840	0.020	...	0.060	0.048	0.249	0.116	0.500	0.163	0.321
54	2013-11-13 23:00:00	0.416	0.156	0.095	0.189	0.347	0.239	0.129	0.284	0.018	...	0.055	0.048	0.199	0.143	0.368	0.228	0.324
55	2013-11-15 23:00:00	0.215	0.281	0.099	0.039	0.361	0.158	0.337	0.287	0.022	...	0.055	0.145	0.289	1.459	0.161	0.800	0.357
56	2013-11-16 23:00:00	0.050	0.541	0.149	0.045	0.463	0.324	0.344	0.543	0.020	...	0.044	0.193	0.459	0.113	0.483	0.179	0.319
57	2013-11-17 23:00:00	0.255	0.422	0.100	0.168	0.450	0.245	0.160	0.230	0.022	...	0.095	0.220	0.244	0.118	0.489	0.488	0.340
58	2013-11-18 23:00:00	0.299	0.348	0.102	0.096	0.302	0.292	0.573	0.365	0.020	...	0.055	0.202	0.256	0.246	0.417	0.134	0.346
59	2013-11-19 23:00:00	0.405	0.131	0.150	0.137	0.441	0.212	0.908	0.254	0.014	...	0.035	0.047	0.258	0.127	0.150	0.105	0.326
60	2013-11-20 23:00:00	0.354	0.352	0.098	0.126	0.398	0.234	0.402	0.281	0.025	...	0.072	0.074	0.302	0.179	0.255	0.306	0.344
61	2013-11-22 23:00:00	1.767	0.256	0.100	0.095	0.354	0.161	0.438	0.690	0.014	...	0.088	0.149	0.170	0.111	0.217	0.240	0.344
62	2013-11-23 23:00:00	0.392	0.510	0.147	0.142	0.502	0.176	0.370	1.092	0.055	...	0.076	0.152	0.187	0.111	0.742	0.159	0.351
63	2013-11-24 23:00:00	0.121	0.380	0.101	0.119	0.267	0.069	0.451	0.482	0.022	...	0.061	0.079	0.248	0.114	0.452	0.161	0.352
64	2013-11-25 23:00:00	0.091	0.307	0.100	0.108	0.420	0.292	0.525	0.253	0.023	...	0.054	0.047	0.288	0.118	0.383	0.126	0.347

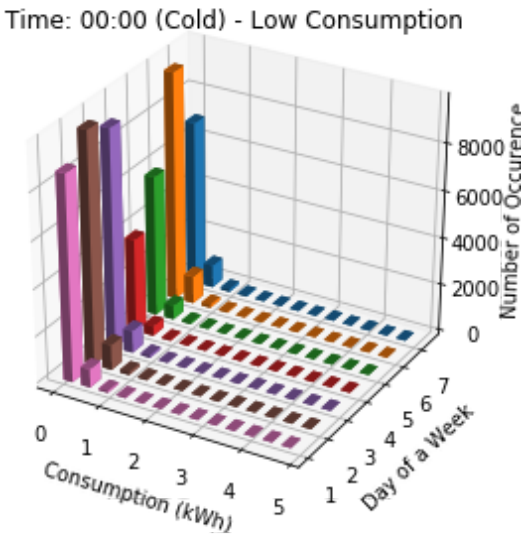
	GMT	D0000	D0001	D0003	D0004	D0005	D0007	D0008	D0009	D0010	...	D1015	D1016	D1017	D1018	D1020	D1021	D102
65	2013-12-02 23:00:00	0.152	0.629	0.099	0.163	0.411	0.233	0.687	0.256	0.020	...	0.082	0.048	0.272	0.189	0.519	0.073	0.329
66	2013-12-03 23:00:00	0.149	0.270	0.098	0.217	0.188	0.199	0.368	0.259	0.022	...	0.092	0.076	0.259	0.136	0.558	0.197	0.356
67	2013-12-05 23:00:00	0.177	0.304	0.099	0.147	0.206	0.155	0.586	0.291	0.021	...	0.088	0.076	0.298	0.263	0.831	0.153	0.384
68	2013-12-06 23:00:00	0.423	0.523	0.124	0.397	0.463	0.330	0.529	0.342	0.021	...	0.094	0.047	0.173	0.112	1.073	0.175	0.386
69	2013-12-09 23:00:00	0.294	0.369	0.100	0.041	0.439	0.314	0.410	0.337	0.020	...	0.069	0.077	0.238	0.392	0.489	0.174	0.377
70	2013-12-10 23:00:00	0.287	0.232	0.152	0.232	0.470	0.339	0.505	0.263	0.016	...	0.071	0.124	0.228	0.344	0.445	0.317	0.367
71	2013-12-11 23:00:00	0.289	0.255	0.103	0.079	0.531	0.396	0.841	0.274	0.015	...	0.038	0.076	0.207	0.827	0.632	0.244	0.346
72	2013-12-14 23:00:00	1.094	0.354	0.146	0.269	0.398	0.355	0.457	0.226	0.029	...	0.042	0.075	0.314	0.107	0.985	0.124	0.348
73	2013-12-17 23:00:00	0.272	0.182	0.186	0.040	0.174	0.290	0.489	0.234	0.026	...	0.062	0.170	0.239	0.126	0.498	0.108	0.358
74	2013-12-18 23:00:00	0.165	0.448	0.100	0.173	0.333	0.357	0.860	0.332	0.029	...	0.086	0.049	0.302	0.129	0.468	0.190	0.337
75	2013-12-21 23:00:00	0.294	0.378	0.149	0.219	0.457	0.262	0.430	0.454	0.020	...	0.059	0.051	0.362	0.107	0.458	0.114	0.354
76	2013-12-22 23:00:00	0.544	0.123	0.150	0.388	0.430	0.200	0.937	0.941	0.026	...	0.116	0.073	0.199	1.462	0.721	0.189	0.415
77	2013-12-23 23:00:00	0.359	0.154	0.127	0.225	0.550	0.379	0.451	0.307	0.026	...	0.086	0.073	0.277	0.118	0.658	0.249	0.140
78	2013-12-24 23:00:00	0.775	0.274	0.099	0.386	0.435	0.285	0.450	1.049	0.016	...	0.065	0.144	0.304	0.116	0.554	0.101	0.140
79	2013-12-25 23:00:00	0.813	0.313	0.100	0.313	0.166	0.329	0.447	0.124	0.027	...	0.065	0.047	0.319	0.111	0.405	0.133	0.141
80	2013-12-27 23:00:00	0.313	0.264	0.099	0.251	0.481	0.294	0.451	0.235	0.032	...	0.066	0.057	0.237	0.333	1.297	0.121	0.342
81	2013-12-28 23:00:00	0.295	0.248	0.151	0.211	0.348	0.189	0.435	0.189	0.022	...	0.039	0.078	0.511	0.109	0.266	0.151	0.373
82	2013-12-31 23:00:00	0.319	0.103	0.104	0.037	0.453	0.382	0.415	0.161	0.017	...	0.073	0.225	0.495	0.110	0.935	0.103	0.354

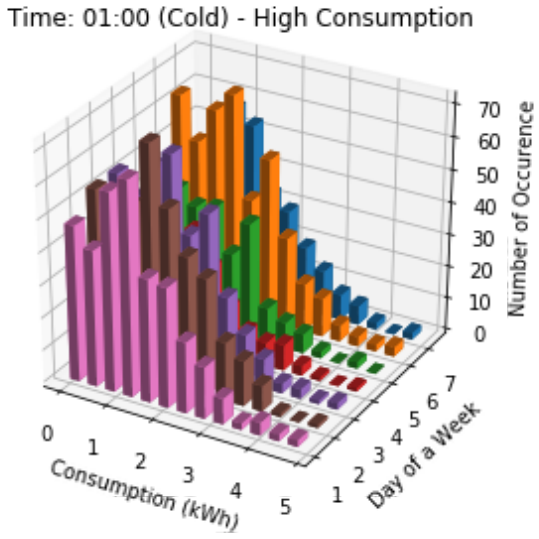
83 rows × 720 columns

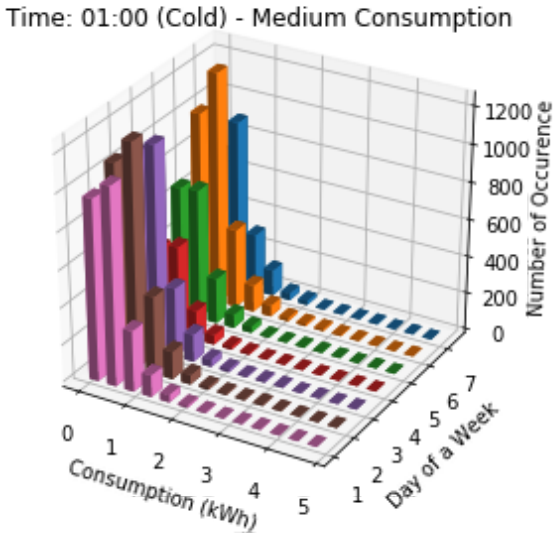
In [48]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
cons_type = ['High Consumption', 'Medium Consumption', 'Low Consumption']
fig_all = plt.figure(figsize = (13,90))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        t.reset_index(inplace = True)
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['DayofWeek'] = np.array(pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0'
+ str(i) + ':00:00')]['GMT']).dt.dayofweek) # add day of a week column to dataframe
            for day in reversed(range(7)):
                t3 = t2[t2['DayofWeek'] == day] # select the data with a specific day of a week
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=13, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(13) * (day+1)
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(13) * 0.2
                dy = np.ones(13) * 0.5
                # dz = t3[0] / ((kmeans.labels_ == sorted_label[k][0]).sum() * (t2['DayofWeek'] == day).sum
())
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 3 + k + 1
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Day of a Week')
            ax.set_zlabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        t.reset_index(inplace = True)
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 3, i * 3 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['DayofWeek'] = np.array(pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(
i) + ':00:00')]['GMT']).dt.dayofweek) # add day of weeks column to dataframe
            for day in reversed(range(7)):
                t3 = t2[t2['DayofWeek'] == day] # select the data with a specific day of a week
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=13, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(13) * (day+1)
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(13) * 0.2
                dy = np.ones(13) * 0.5
                # dz = t3[0] / ((kmeans.labels_ == sorted_label[k][0]).sum() * (t2['DayofWeek'] == day).sum
())
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 3 + k + 1
            ax.set_title('Time: ' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Day of a Week')
            ax.set_zlabel('Number of Occurence')
plt.tight_layout()
```

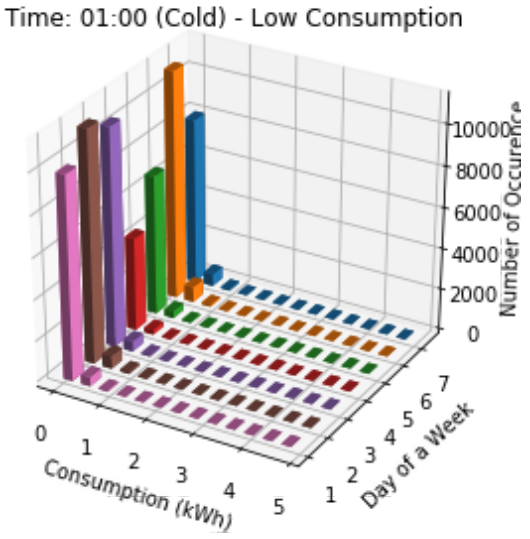

Time: 00:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

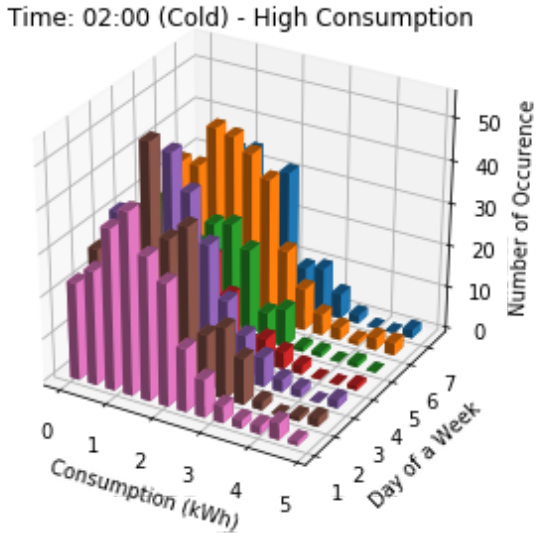

Time: 00:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

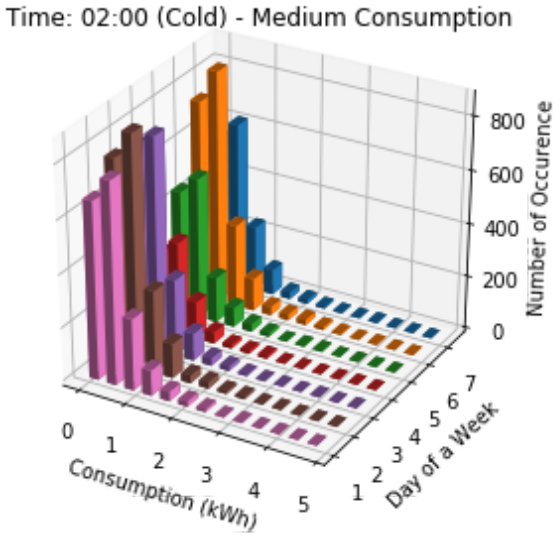

Time: 00:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

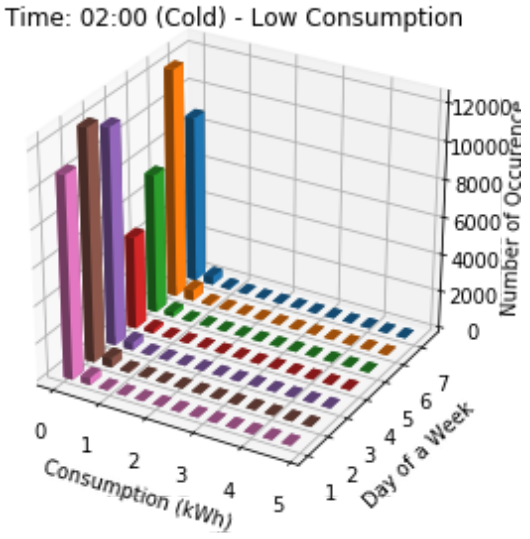

Time: 01:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

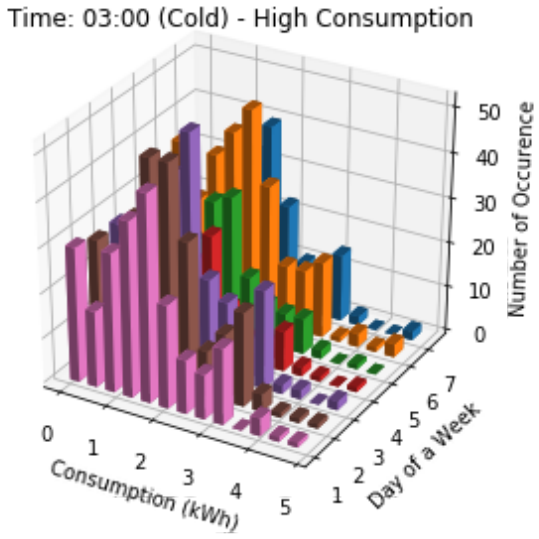

Time: 01:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

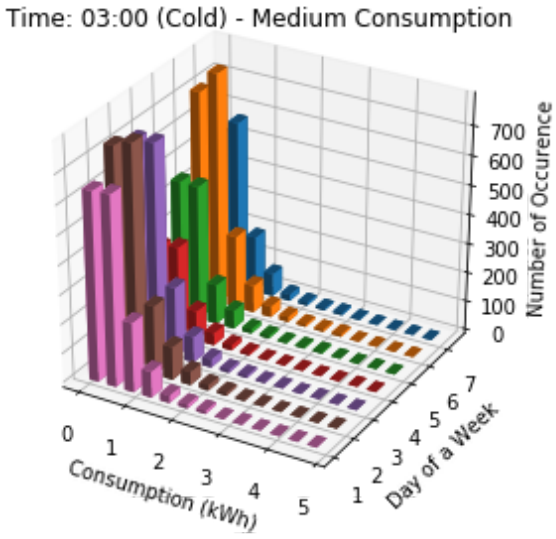

Time: 01:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

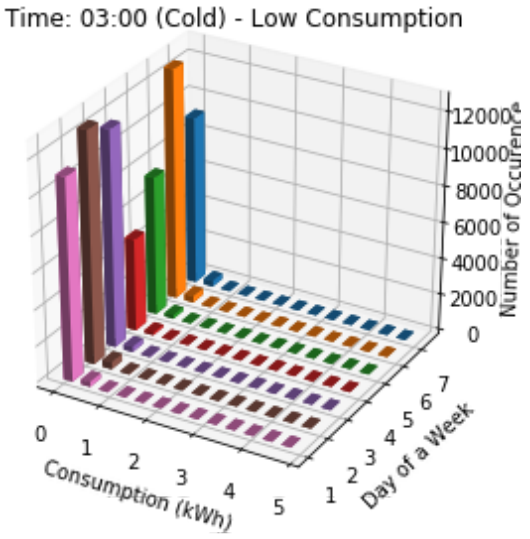

Time: 02:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

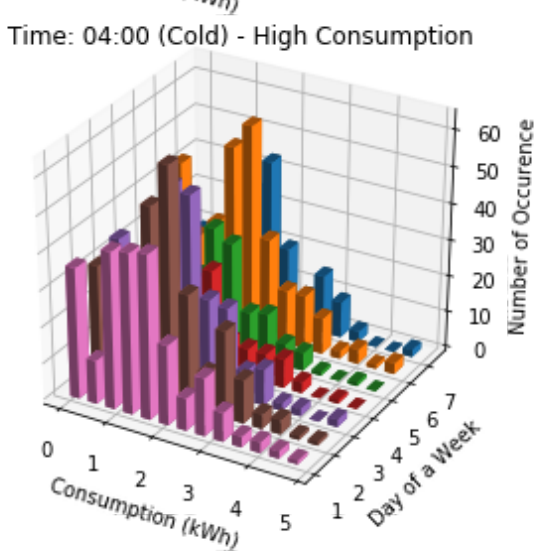

Time: 02:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

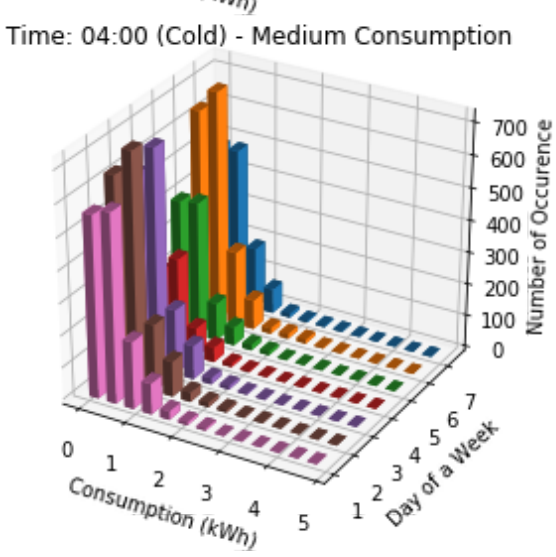

Time: 02:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

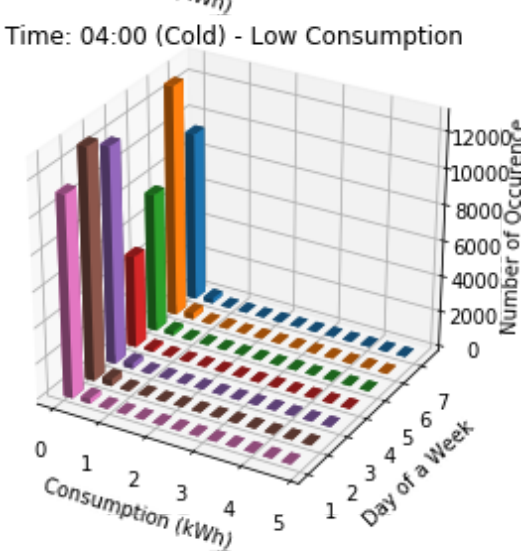

Time: 03:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

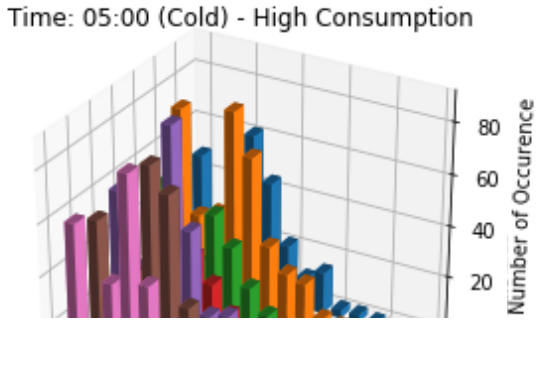

Time: 03:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

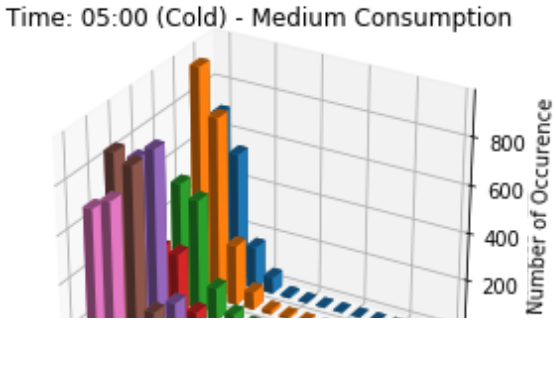

Time: 03:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

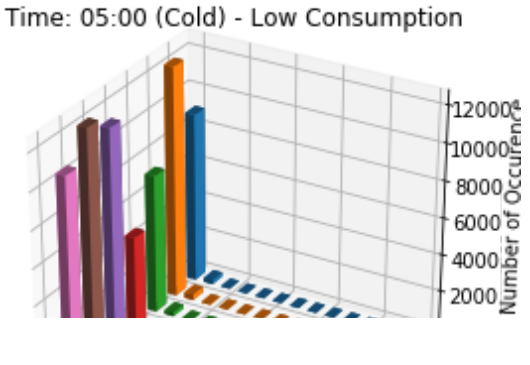

Time: 04:00 (Cold) - High Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

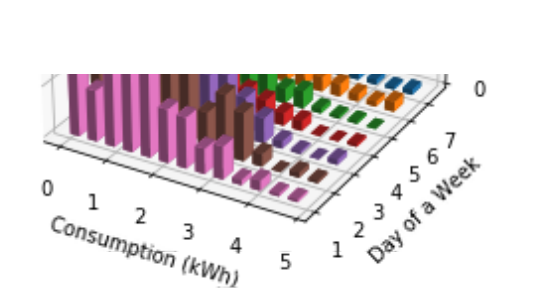

Time: 04:00 (Cold) - Medium Consumption
Number of Occurence
Consumption (kWh)
Day of a Week


Time: 04:00 (Cold) - Low Consumption
Number of Occurence
Consumption (kWh)
Day of a Week

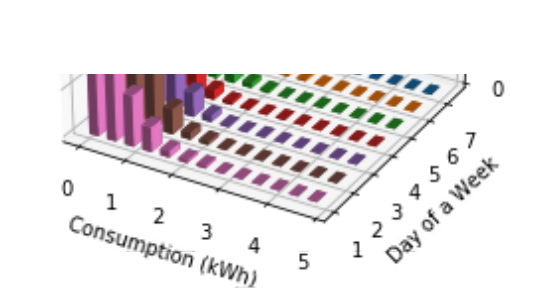

Time: 05:00 (Cold) - High Consumption
Number of Occurence


Time: 05:00 (Cold) - Medium Consumption
Number of Occurence

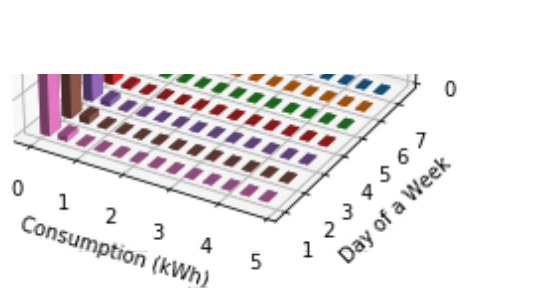

Time: 05:00 (Cold) - Low Consumption
Number of Occurence



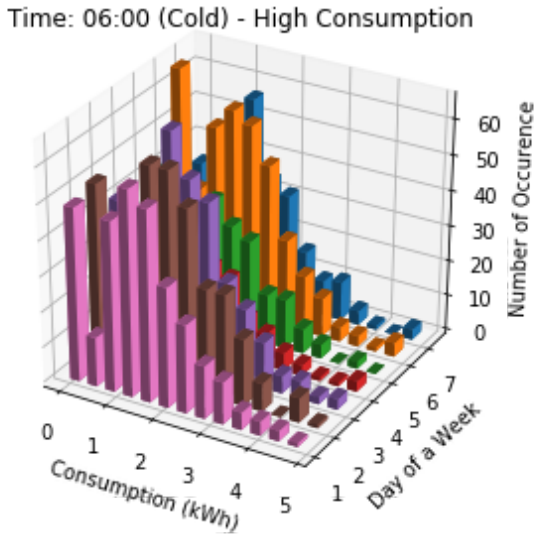
Time: 06:00 (Cold) - High Consumption



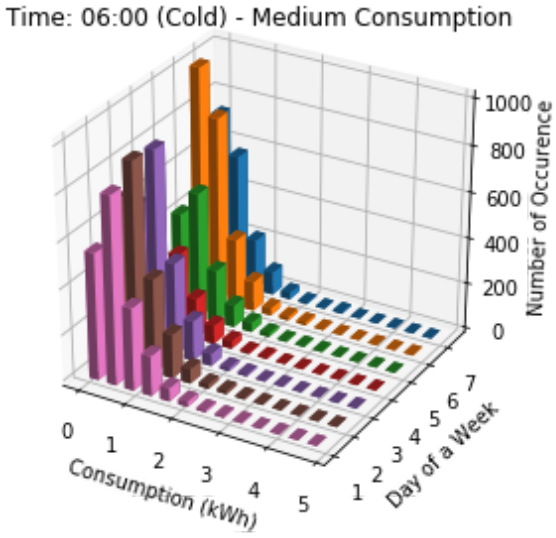
Time: 06:00 (Cold) - Medium Consumption



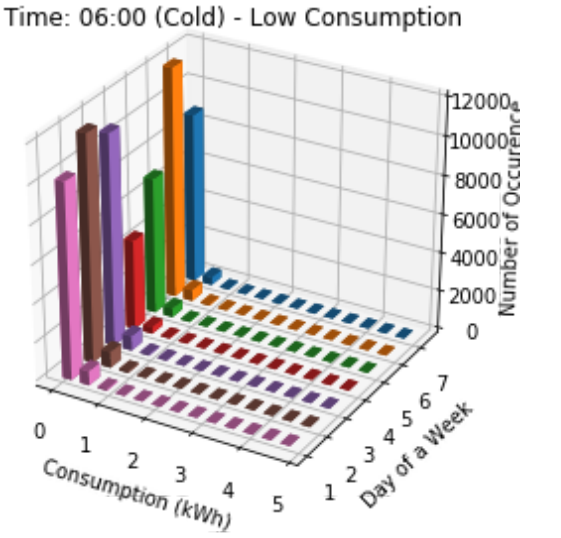
Time: 06:00 (Cold) - Low Consumption



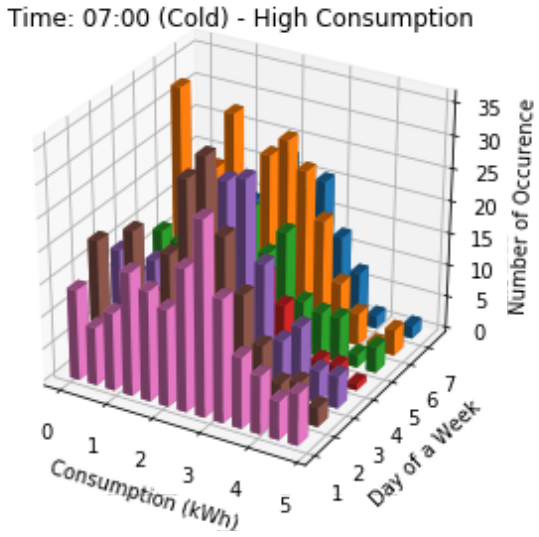
Time: 07:00 (Cold) - High Consumption



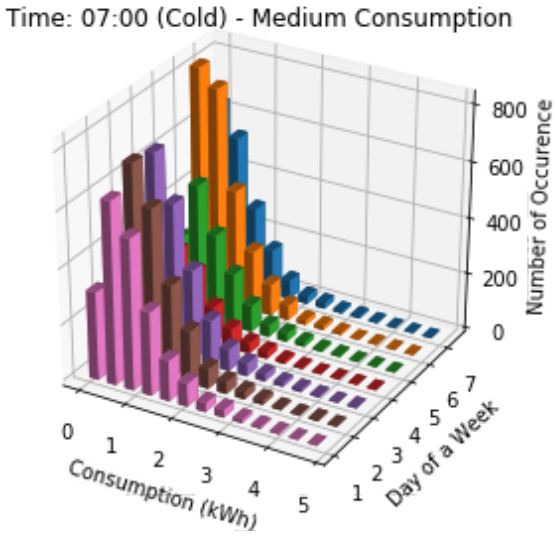
Time: 07:00 (Cold) - Medium Consumption



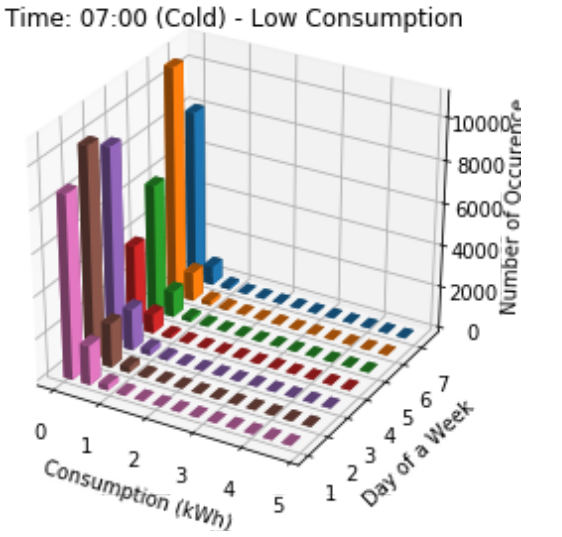
Time: 07:00 (Cold) - Low Consumption



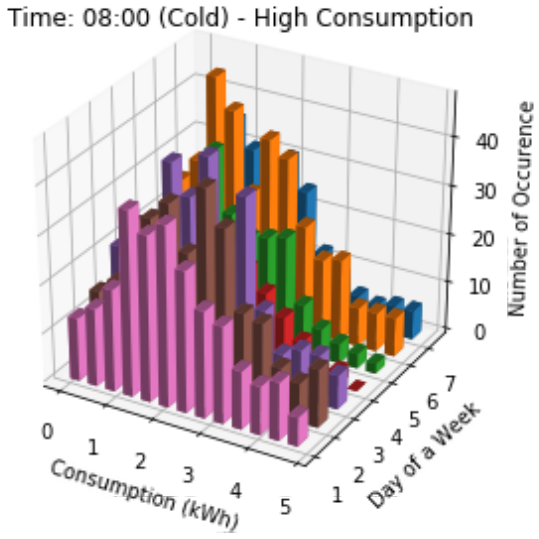
Time: 08:00 (Cold) - High Consumption



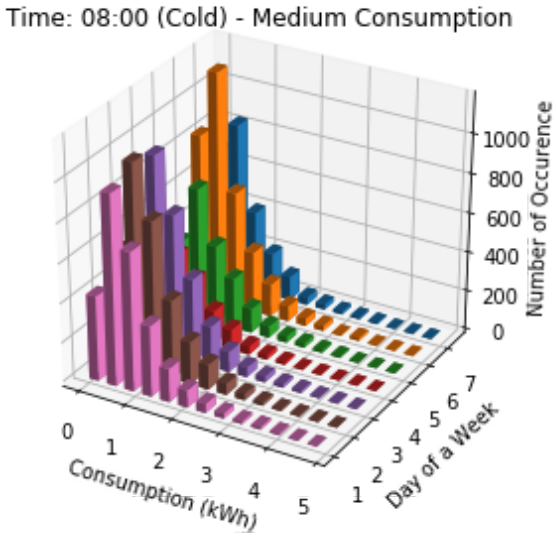
Time: 08:00 (Cold) - Medium Consumption



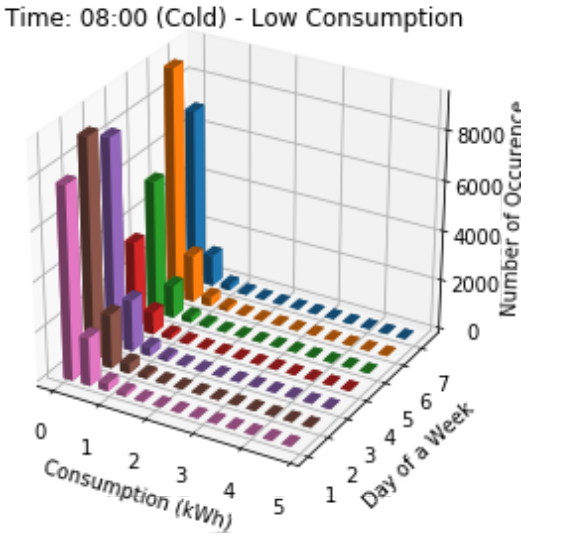
Time: 08:00 (Cold) - Low Consumption



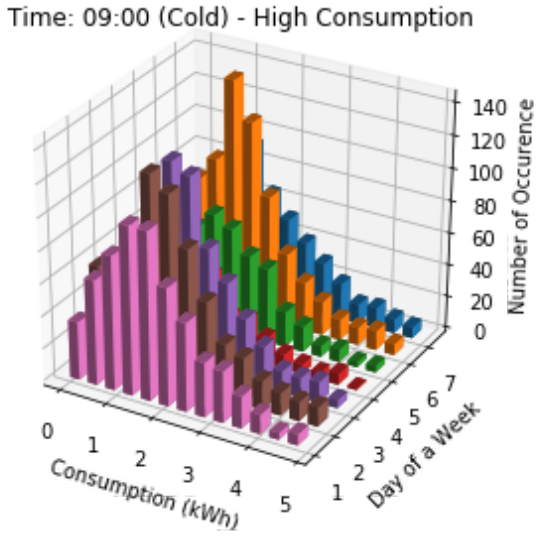
Time: 09:00 (Cold) - High Consumption



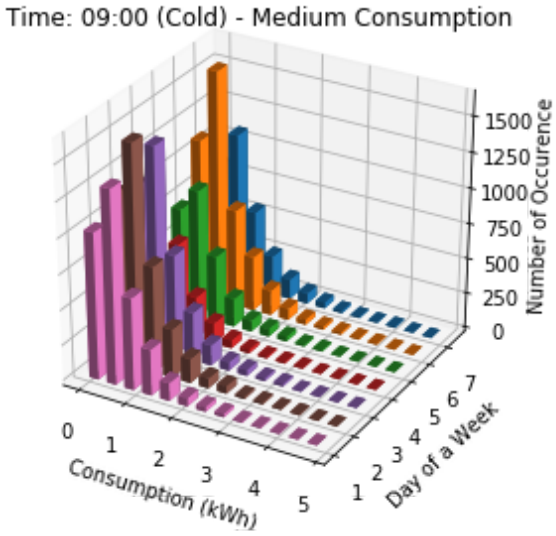
Time: 09:00 (Cold) - Medium Consumption



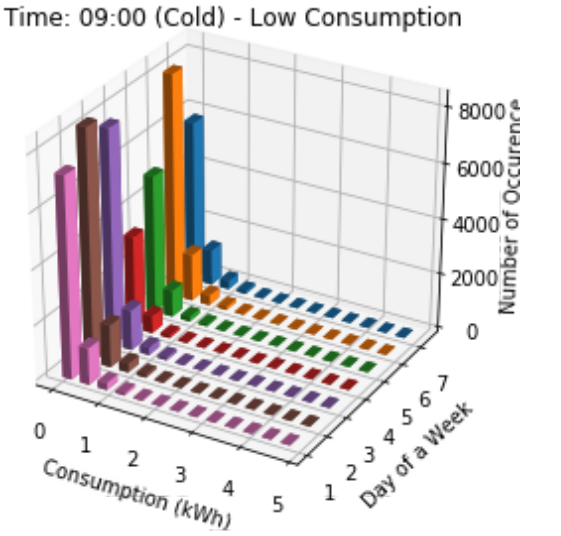
Time: 09:00 (Cold) - Low Consumption



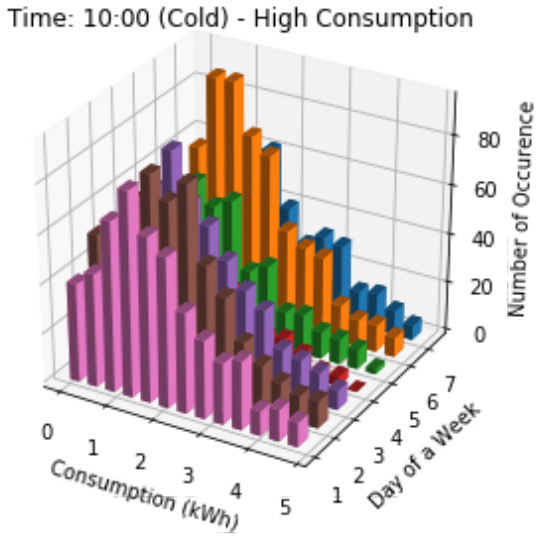
Time: 10:00 (Cold) - High Consumption



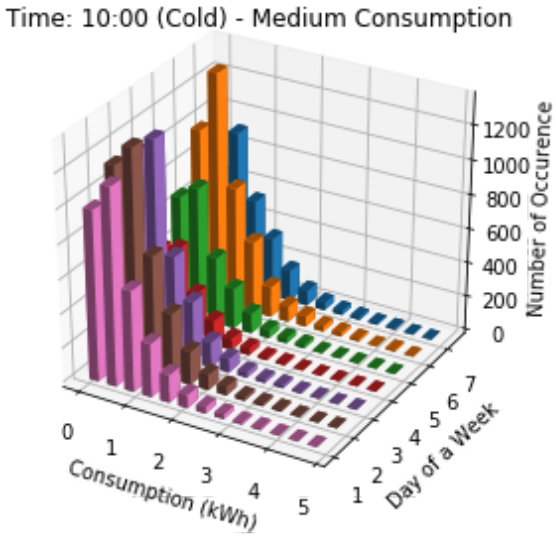
Time: 10:00 (Cold) - Medium Consumption



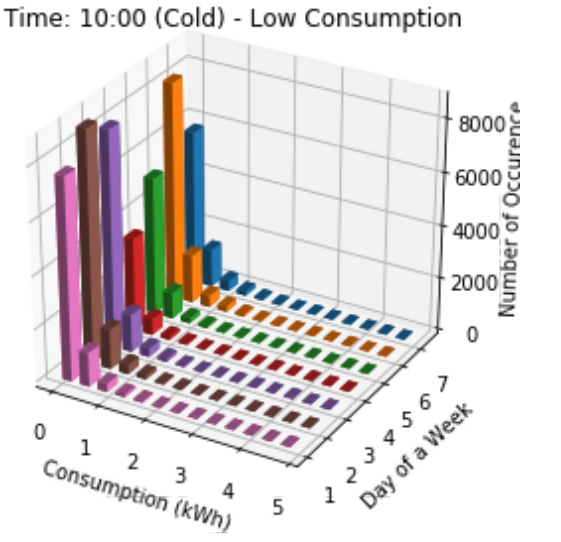
Time: 10:00 (Cold) - Low Consumption



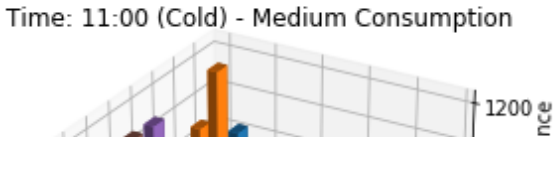
Time: 11:00 (Cold) - High Consumption

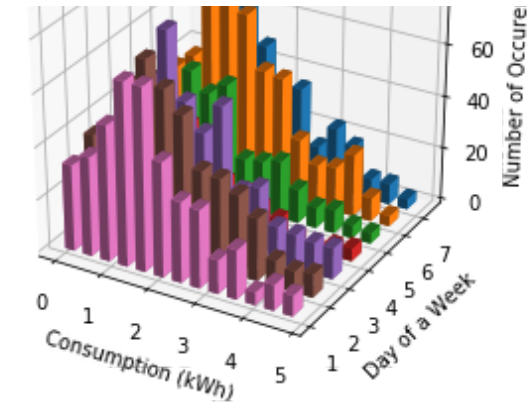


Time: 11:00 (Cold) - Medium Consumption

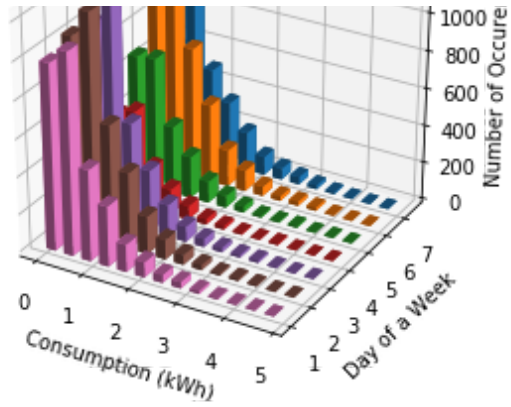


Time: 11:00 (Cold) - Low Consumption

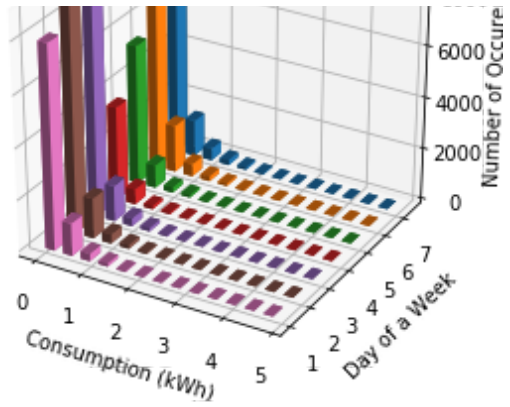




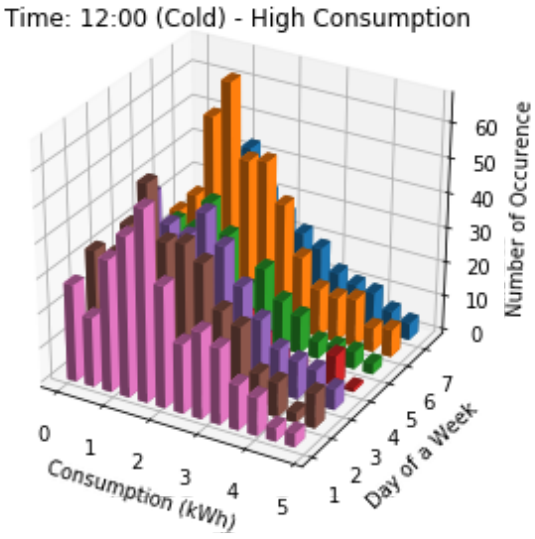
Time: 12:00 (Cold) - High Consumption



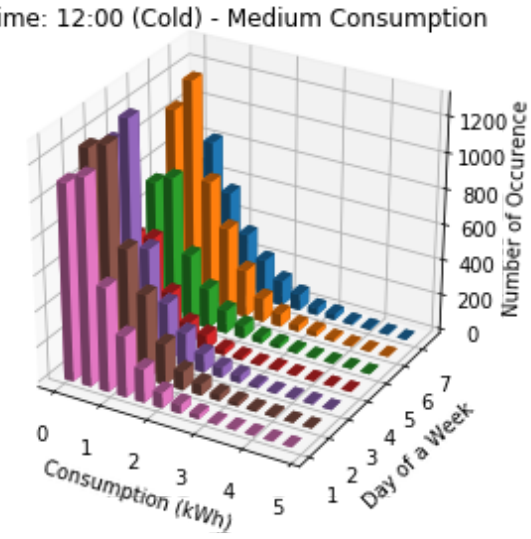
Time: 12:00 (Cold) - Medium Consumption



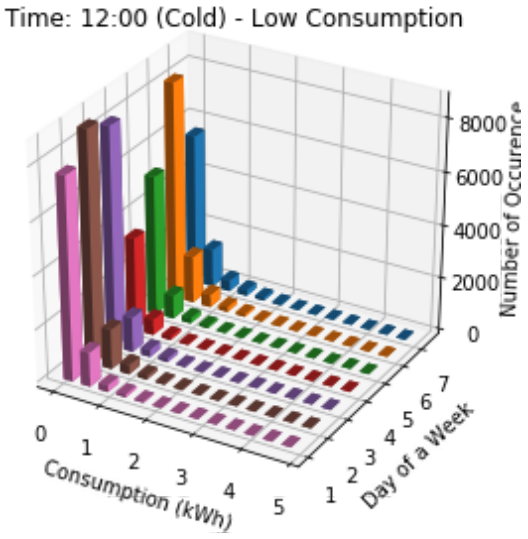
Time: 12:00 (Cold) - Low Consumption



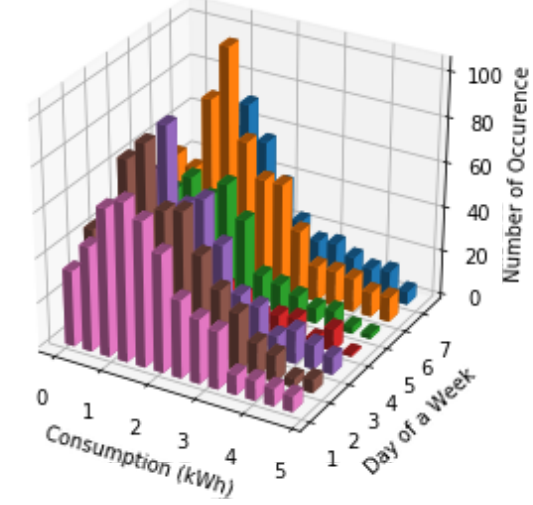
Time: 13:00 (Cold) - High Consumption



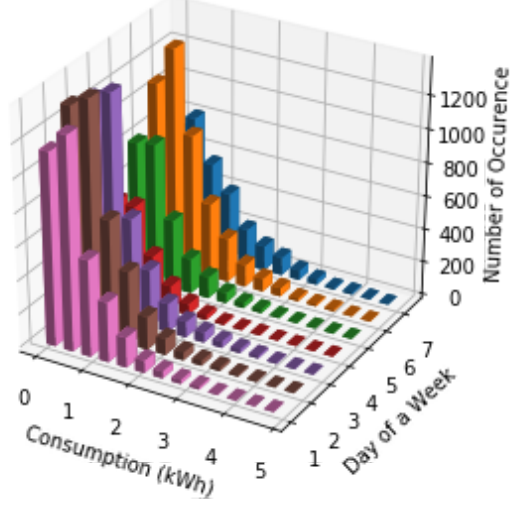
Time: 13:00 (Cold) - Medium Consumption



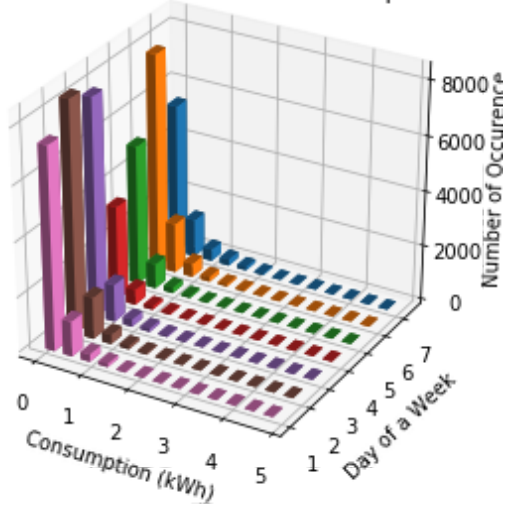
Time: 13:00 (Cold) - Low Consumption



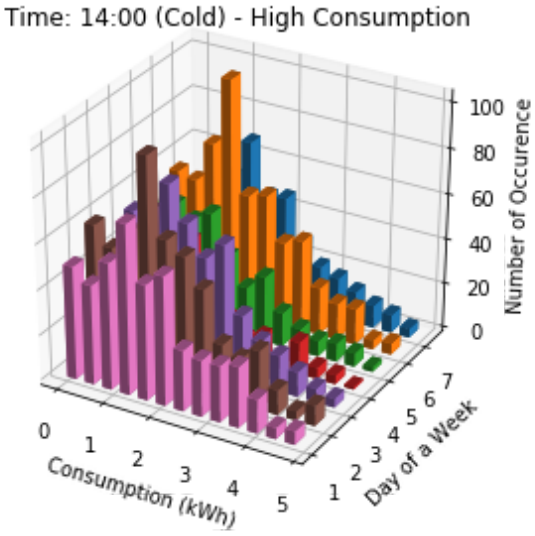
Time: 14:00 (Cold) - High Consumption



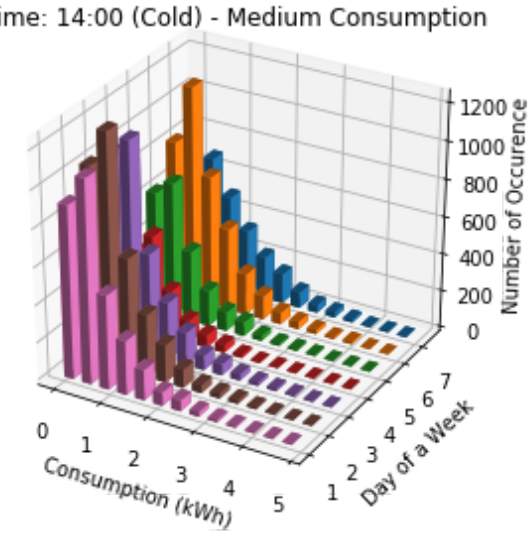
Time: 14:00 (Cold) - Medium Consumption



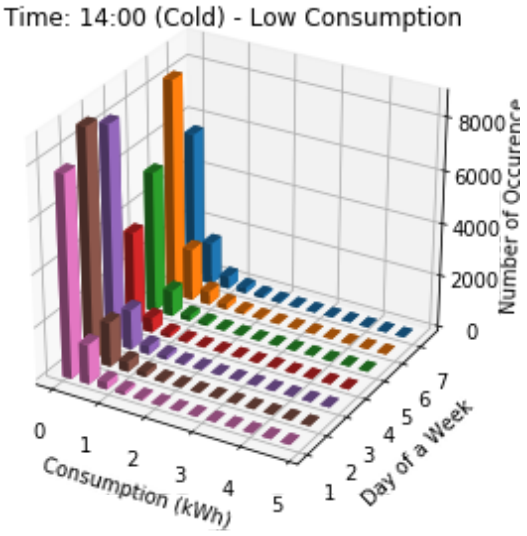
Time: 14:00 (Cold) - Low Consumption



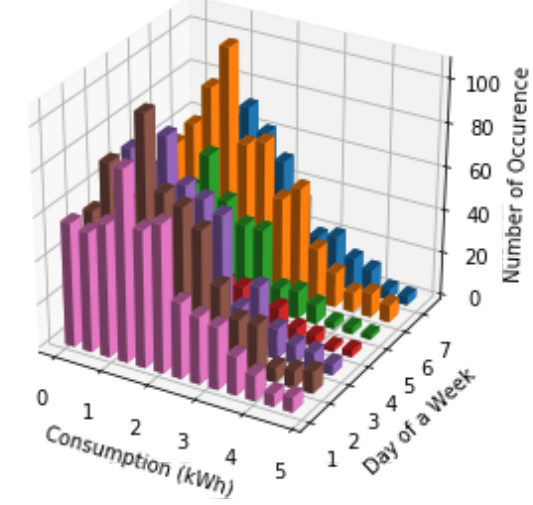
Time: 15:00 (Cold) - High Consumption



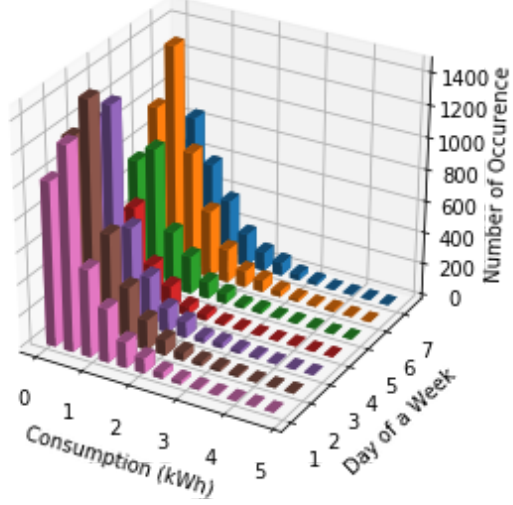
Time: 15:00 (Cold) - Medium Consumption



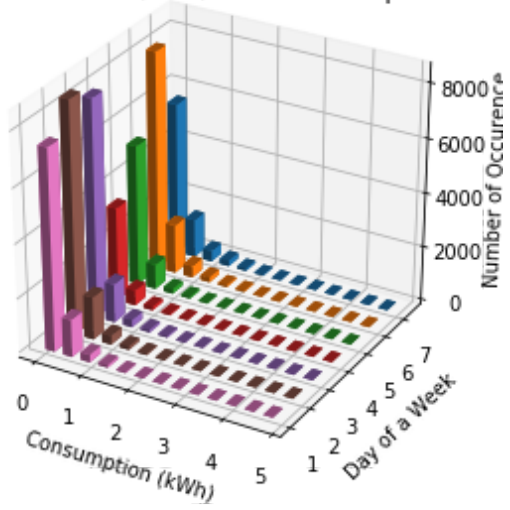
Time: 15:00 (Cold) - Low Consumption



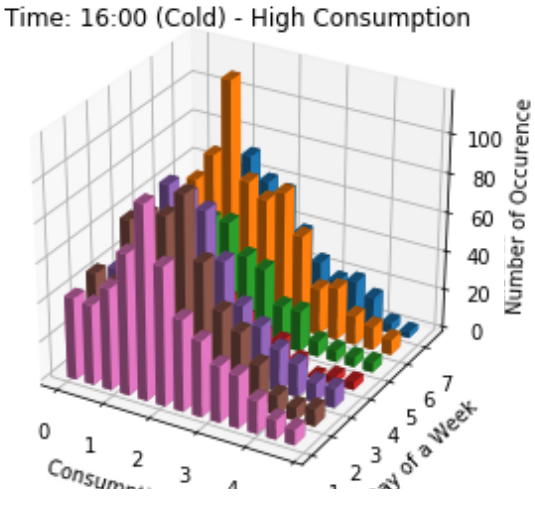
Time: 16:00 (Cold) - High Consumption

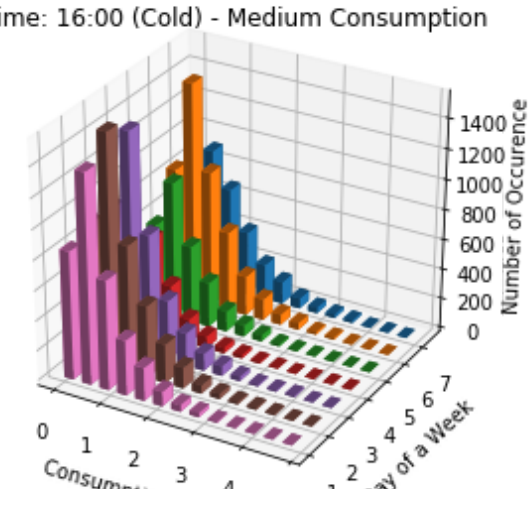


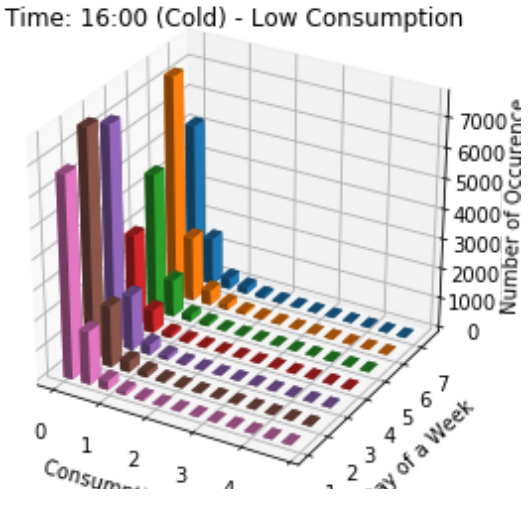
Time: 16:00 (Cold) - Medium Consumption

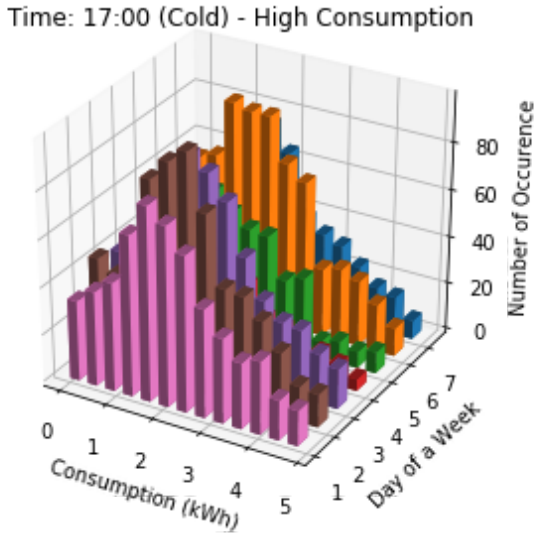


Time: 16:00 (Cold) - Low Consumption

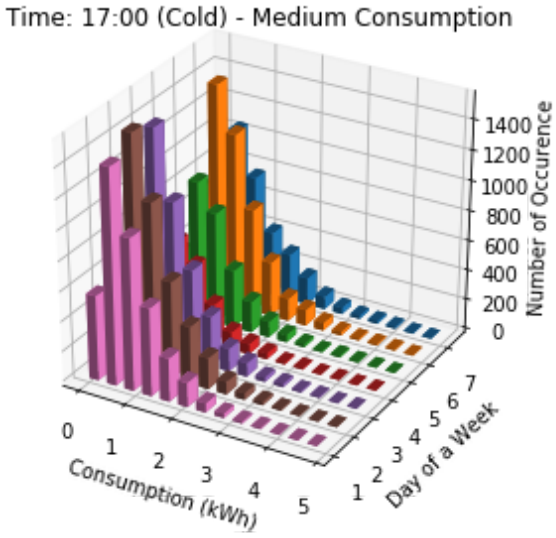




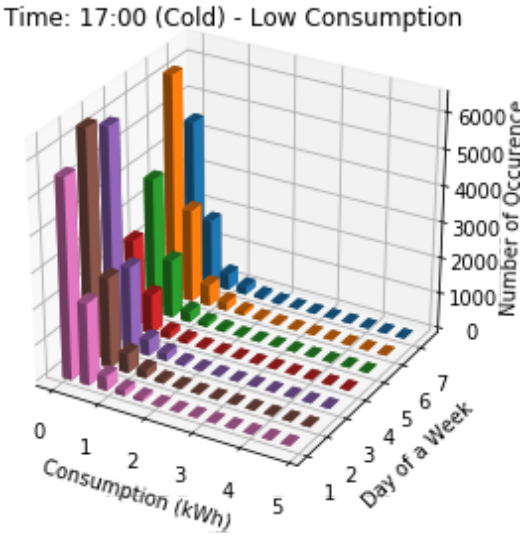




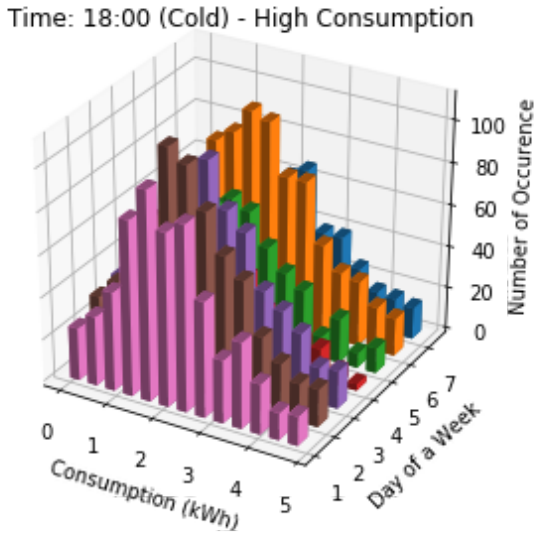
Time: 17:00 (Cold) - High Consumption



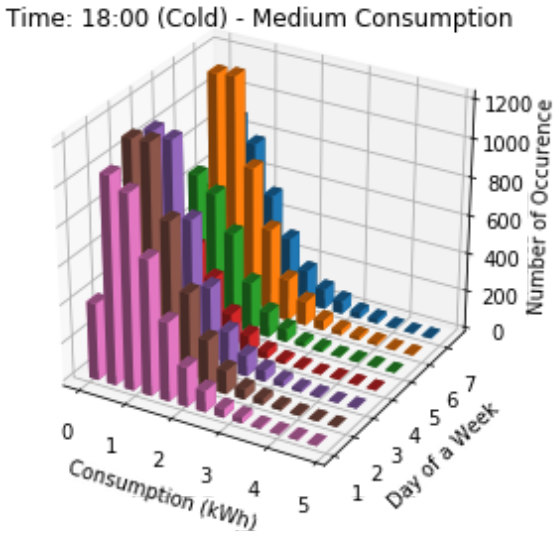
Time: 17:00 (Cold) - Medium Consumption



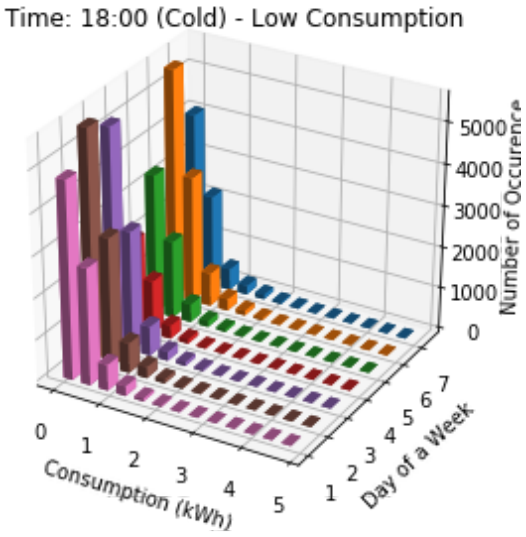
Time: 17:00 (Cold) - Low Consumption



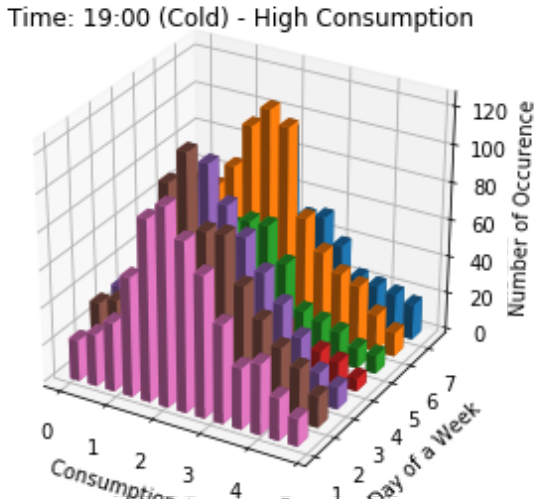
Time: 18:00 (Cold) - High Consumption



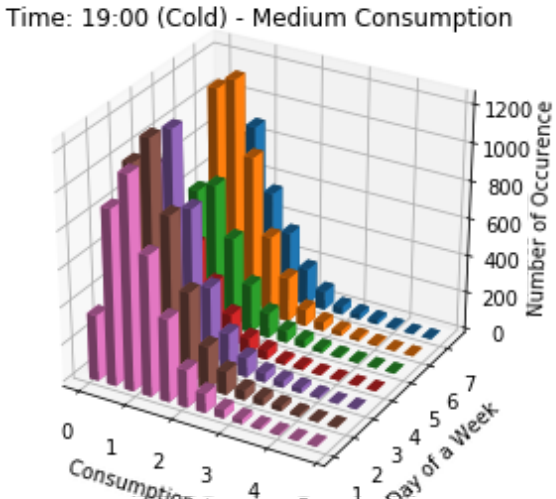
Time: 18:00 (Cold) - Medium Consumption



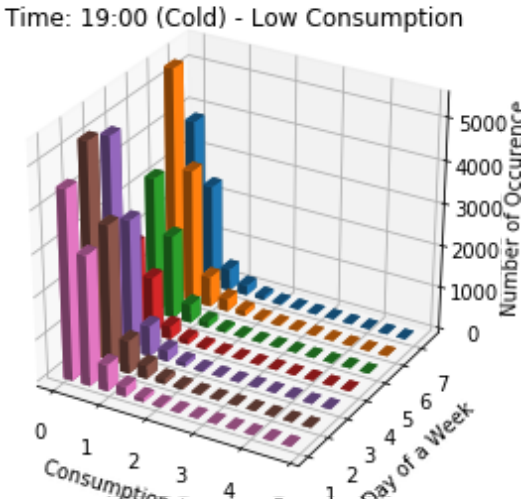
Time: 18:00 (Cold) - Low Consumption



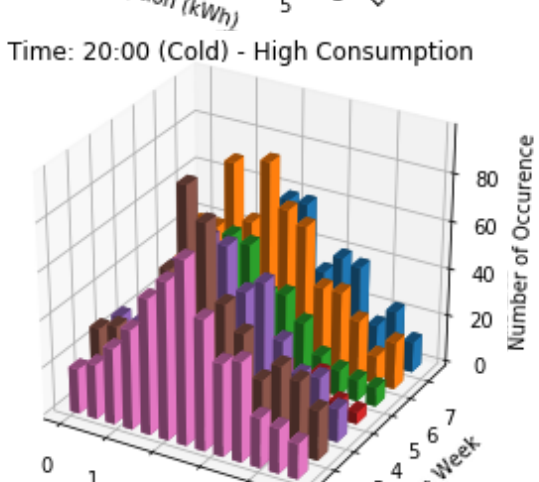
Time: 19:00 (Cold) - High Consumption



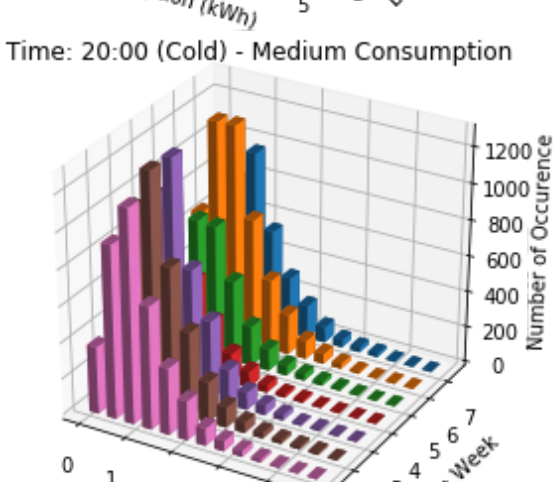
Time: 19:00 (Cold) - Medium Consumption



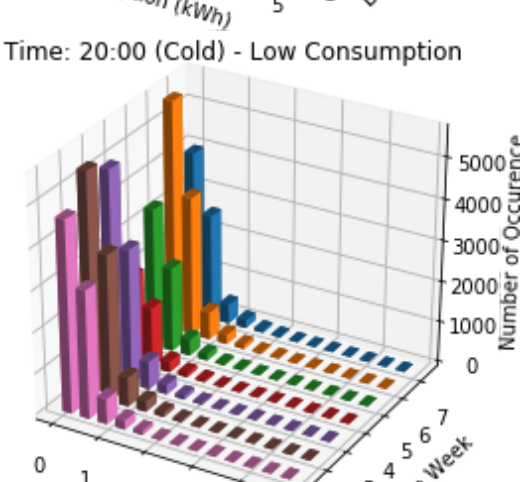
Time: 19:00 (Cold) - Low Consumption



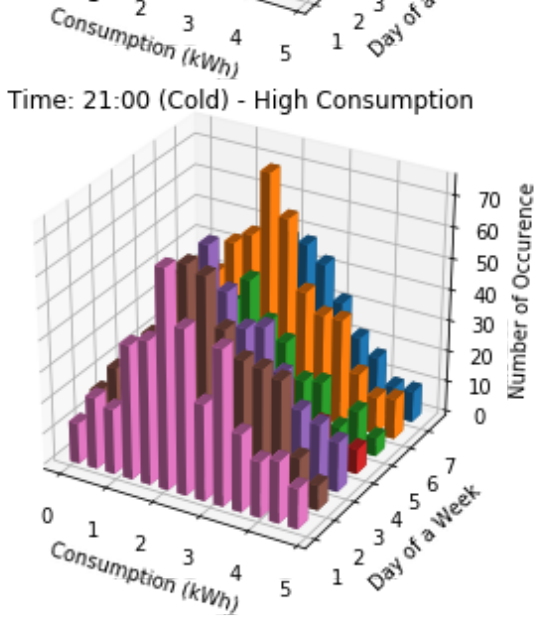
Time: 20:00 (Cold) - High Consumption



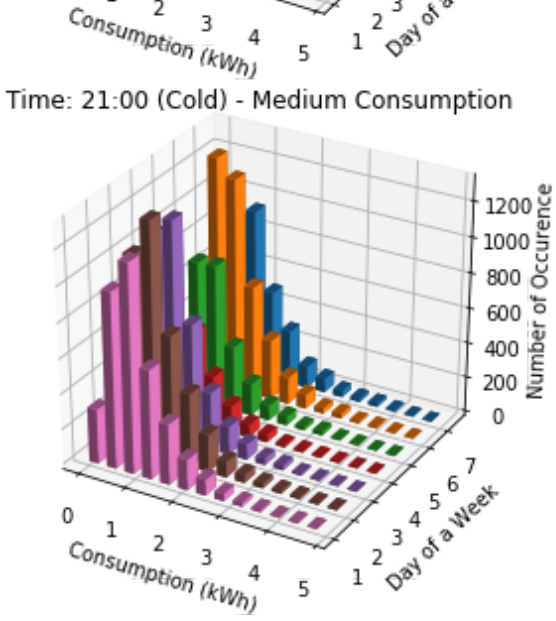
Time: 20:00 (Cold) - Medium Consumption



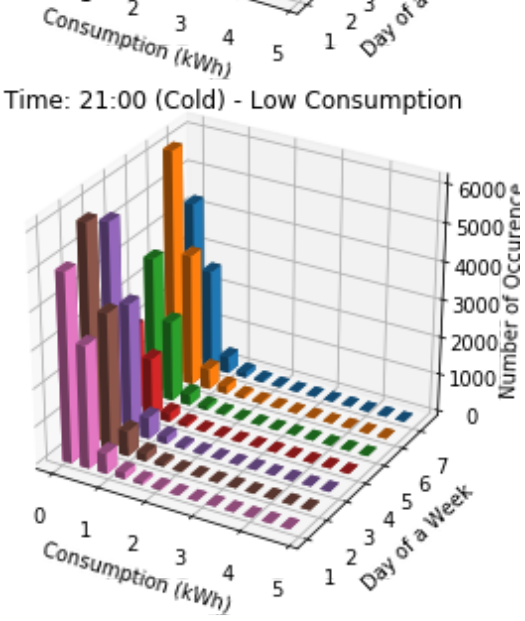
Time: 20:00 (Cold) - Low Consumption



Time: 21:00 (Cold) - High Consumption



Time: 21:00 (Cold) - Medium Consumption

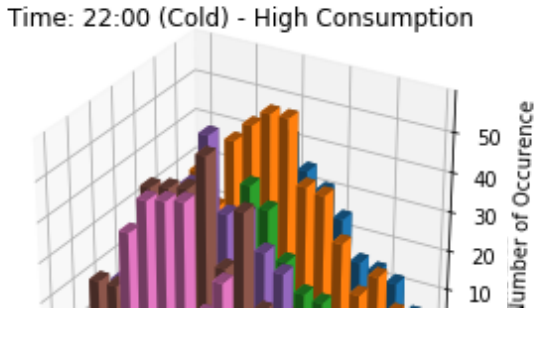


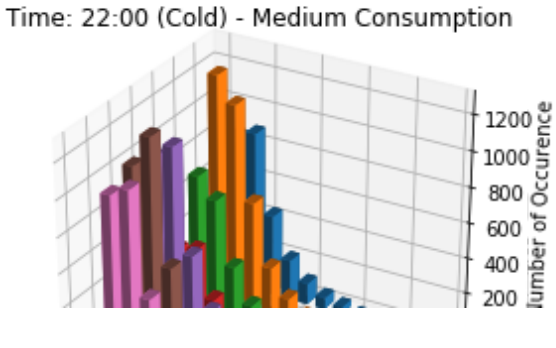
Time: 21:00 (Cold) - Low Consumption

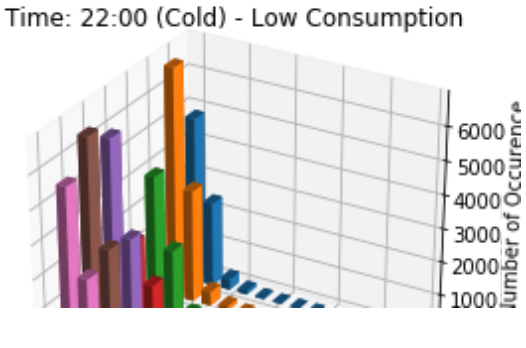
Time: 22:00 (Cold) - High Consumption

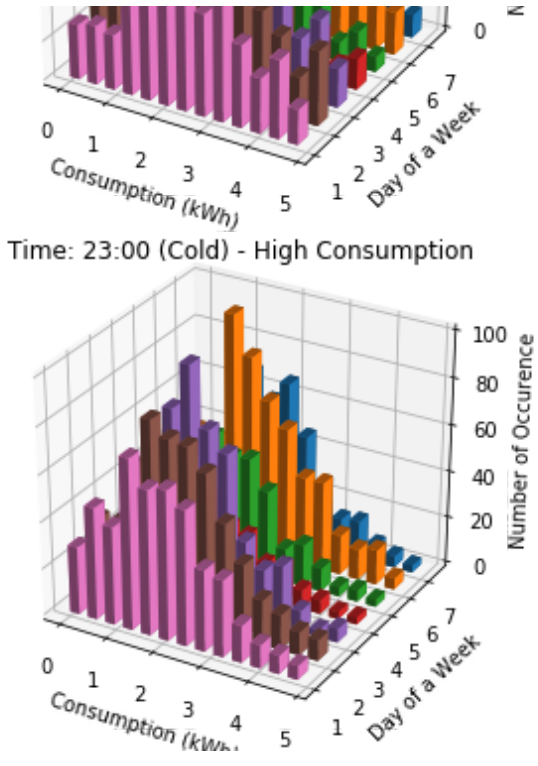
Time: 22:00 (Cold) - Medium Consumption

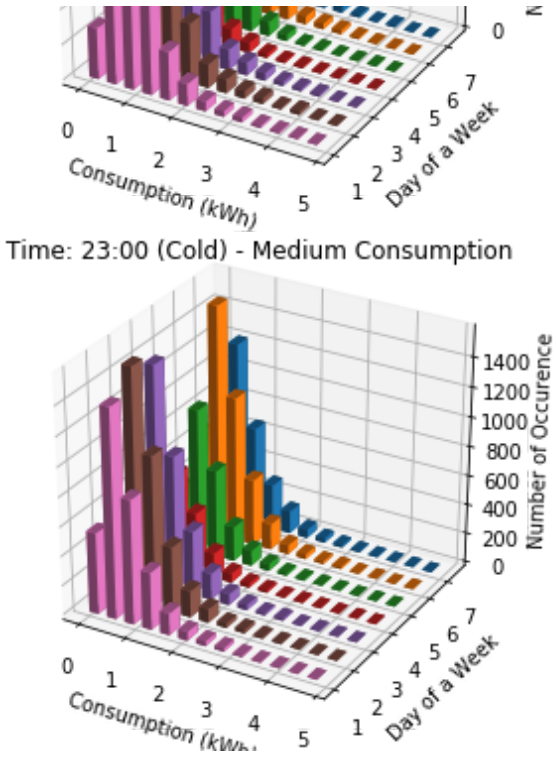
Time: 22:00 (Cold) - Low Consumption

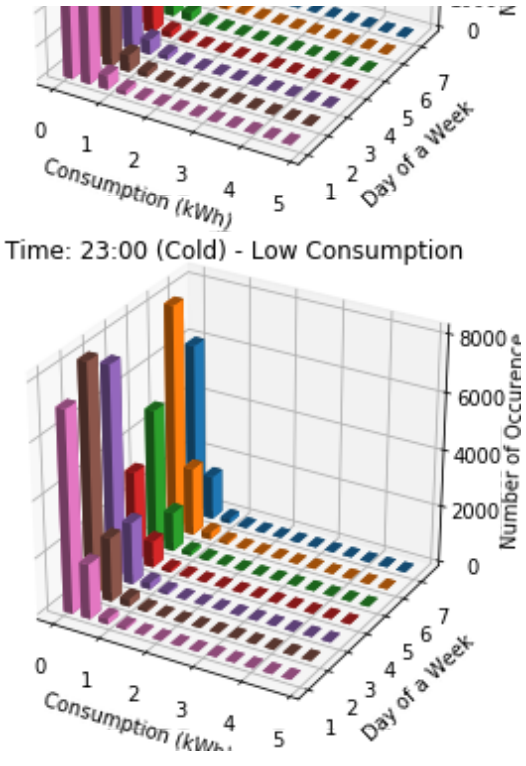






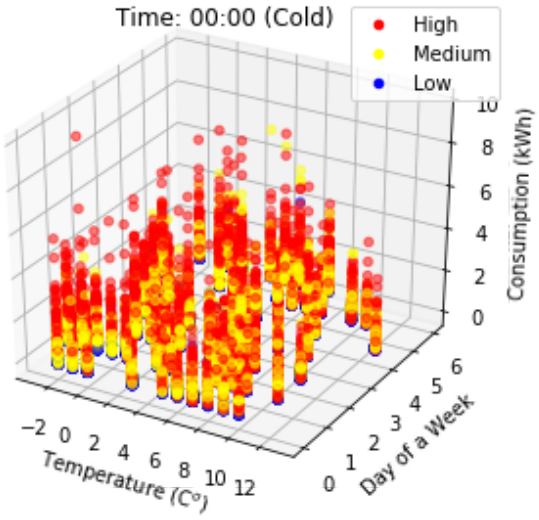


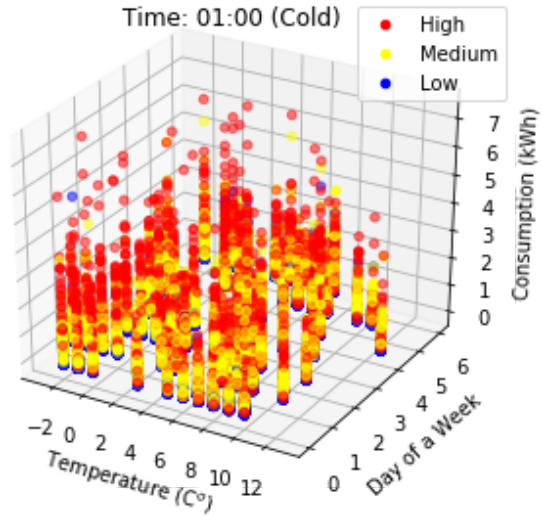


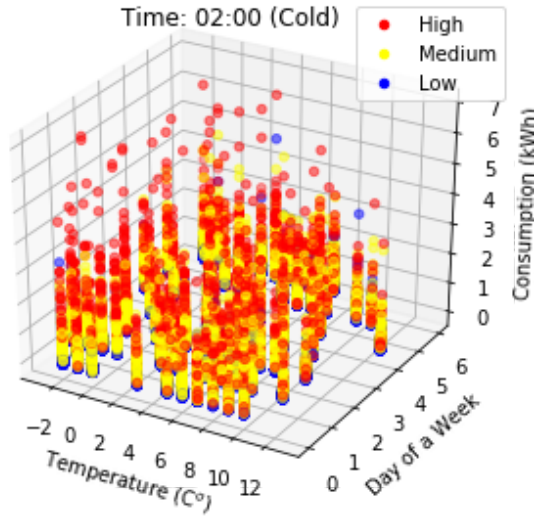


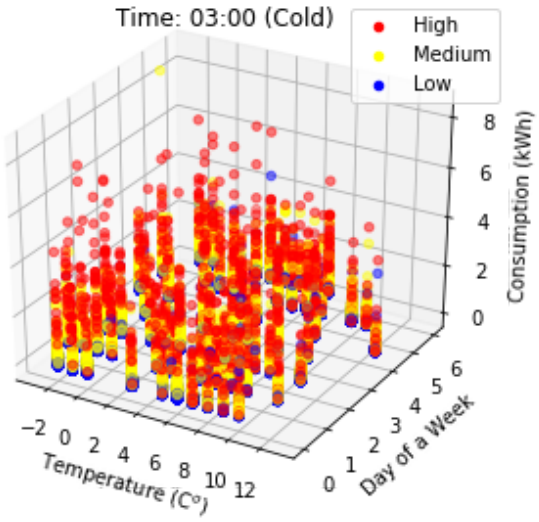
Plot the three groups generated by PCA and k-mean clustering, in a space with 3 dimensions, temperature, day of week and consumption, and see if there is any obvious pattern.

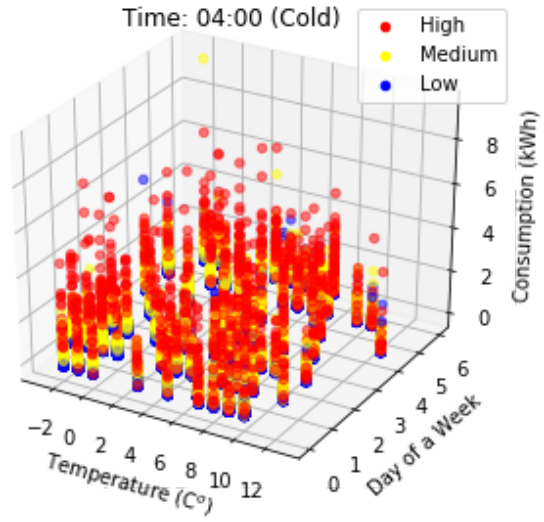
In [49]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
cons_type = ['High', 'Medium', 'Low']
color_type = ['red', 'yellow', 'blue']
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        t.reset_index(inplace = True)
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
        for k in range(n):
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            t2['DayofWeek'] = np.array(pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0'
+ str(i) + ':00:00')]['GMT']).dt.dayofweek) # add day of a week column to dataframe
            ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[1]], color = color_type[k], labe
l = cons_type[k])
            for j in range(2, 1 + sorted_label[k][1]):
                ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[j]], color = color_type[k],
alpha = 0.5)
                ax.legend()
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold)')
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Day of a Week')
        ax.set_zlabel('Consumption (kWh)')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        t = x.fillna(x.mean()) # data used for analyzing consumption distribution
        x = t.transpose() # data for the specific hour, and filtering out null value
        t.reset_index(inplace = True)
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
        for k in range(n):
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            t2['DayofWeek'] = np.array(pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(
i) + ':00:00')]['GMT']).dt.dayofweek) # add day of a week column to dataframe
            ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[1]], color = color_type[k], labe
l = cons_type[k])
            for j in range(2, 1 + sorted_label[k][1]):
                ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[j]], color = color_type[k],
alpha = 0.5)
                ax.legend()
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold)')
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Day of a Week')
        ax.set_zlabel('Consumption (kWh)')
plt.tight_layout()
```

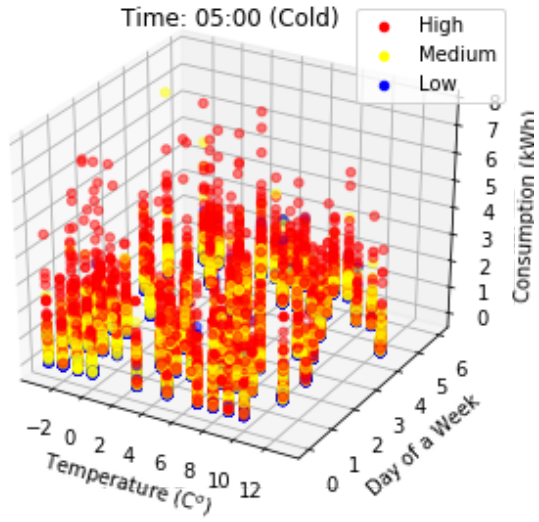

Time: 00:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

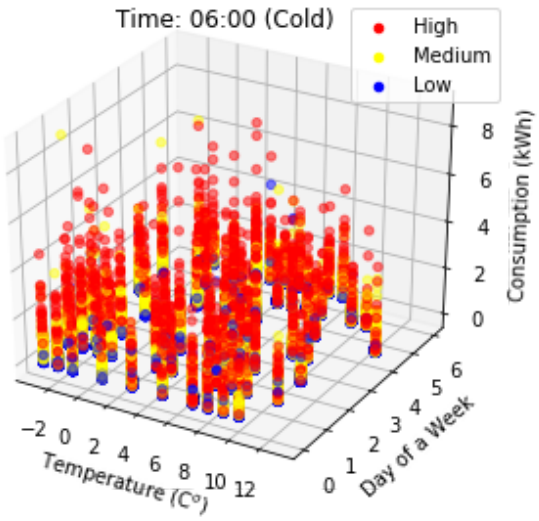

Time: 01:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

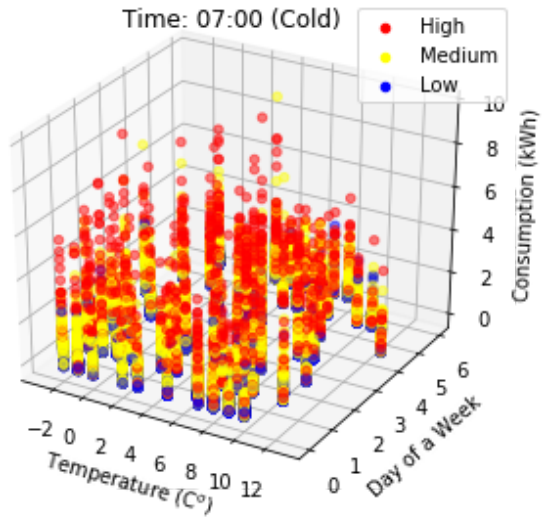

Time: 02:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

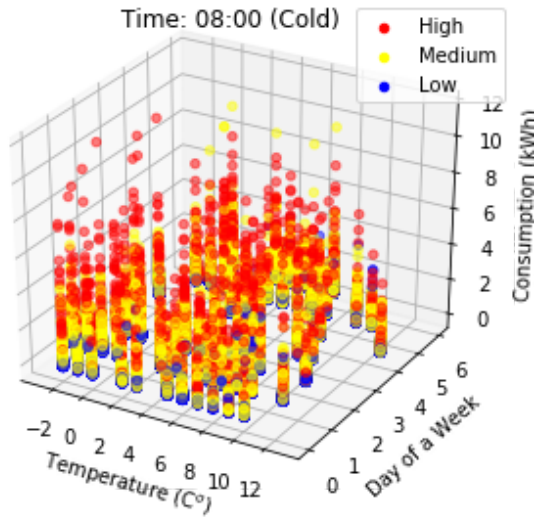

Time: 03:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

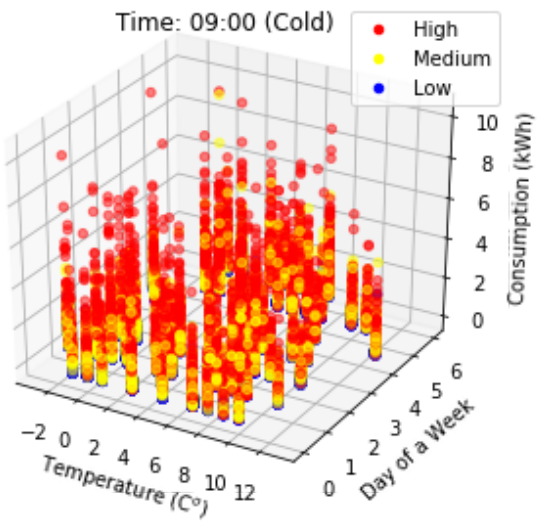

Time: 04:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

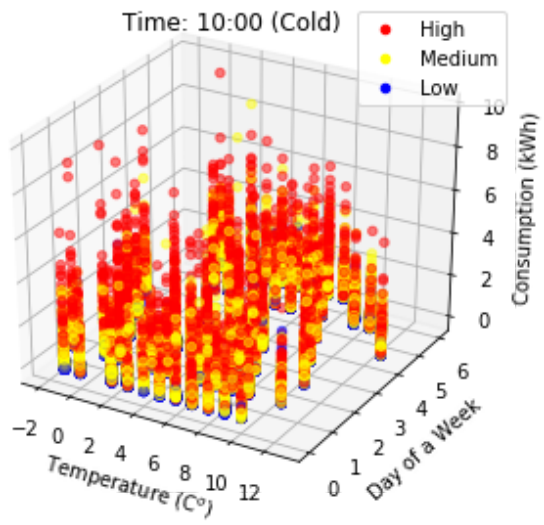

Time: 05:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

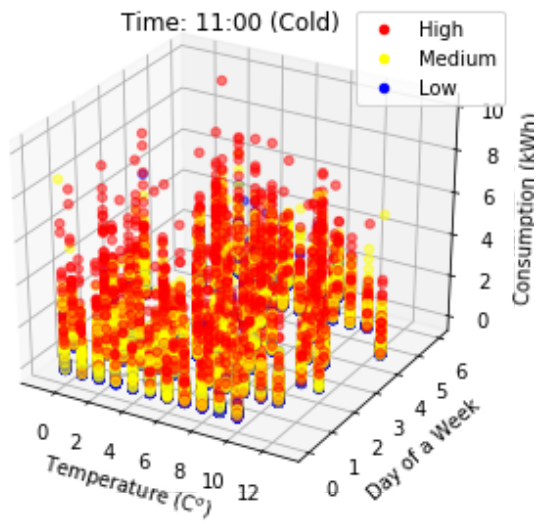

Time: 06:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

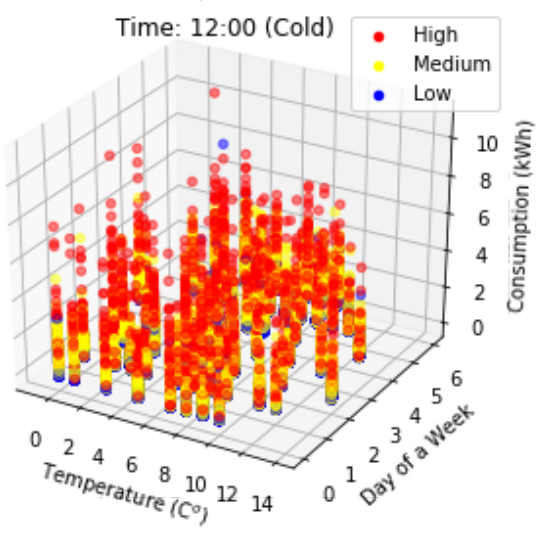

Time: 07:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

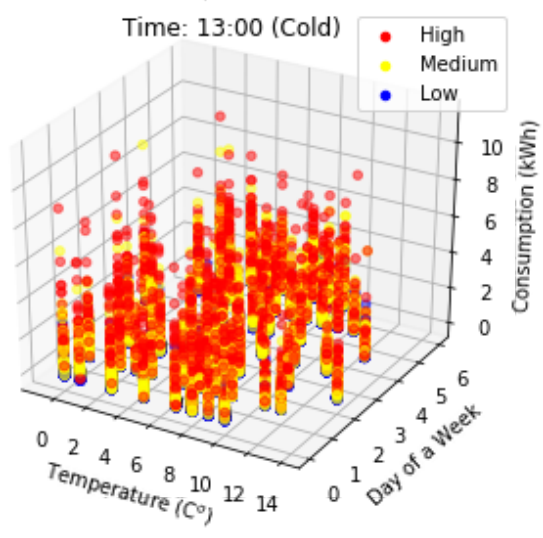

Time: 08:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

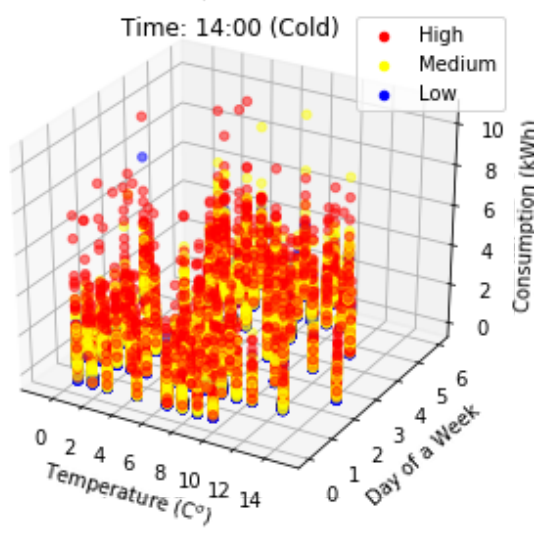

Time: 09:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

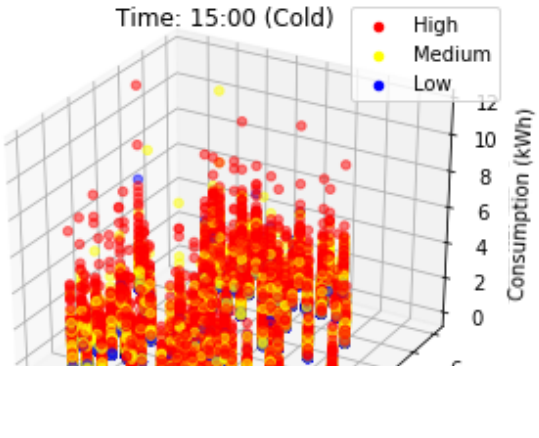

Time: 10:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

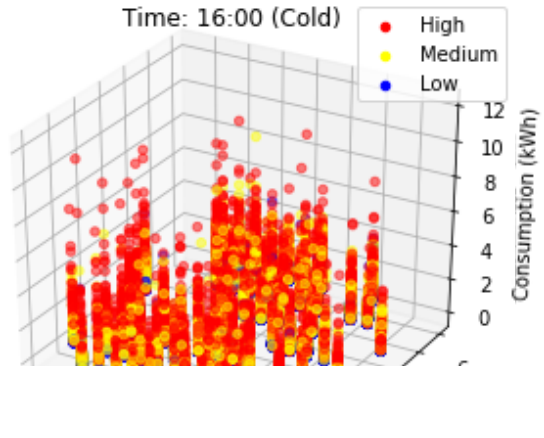

Time: 11:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

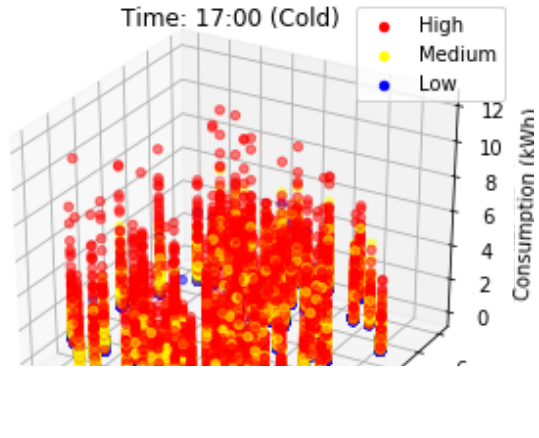

Time: 12:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)

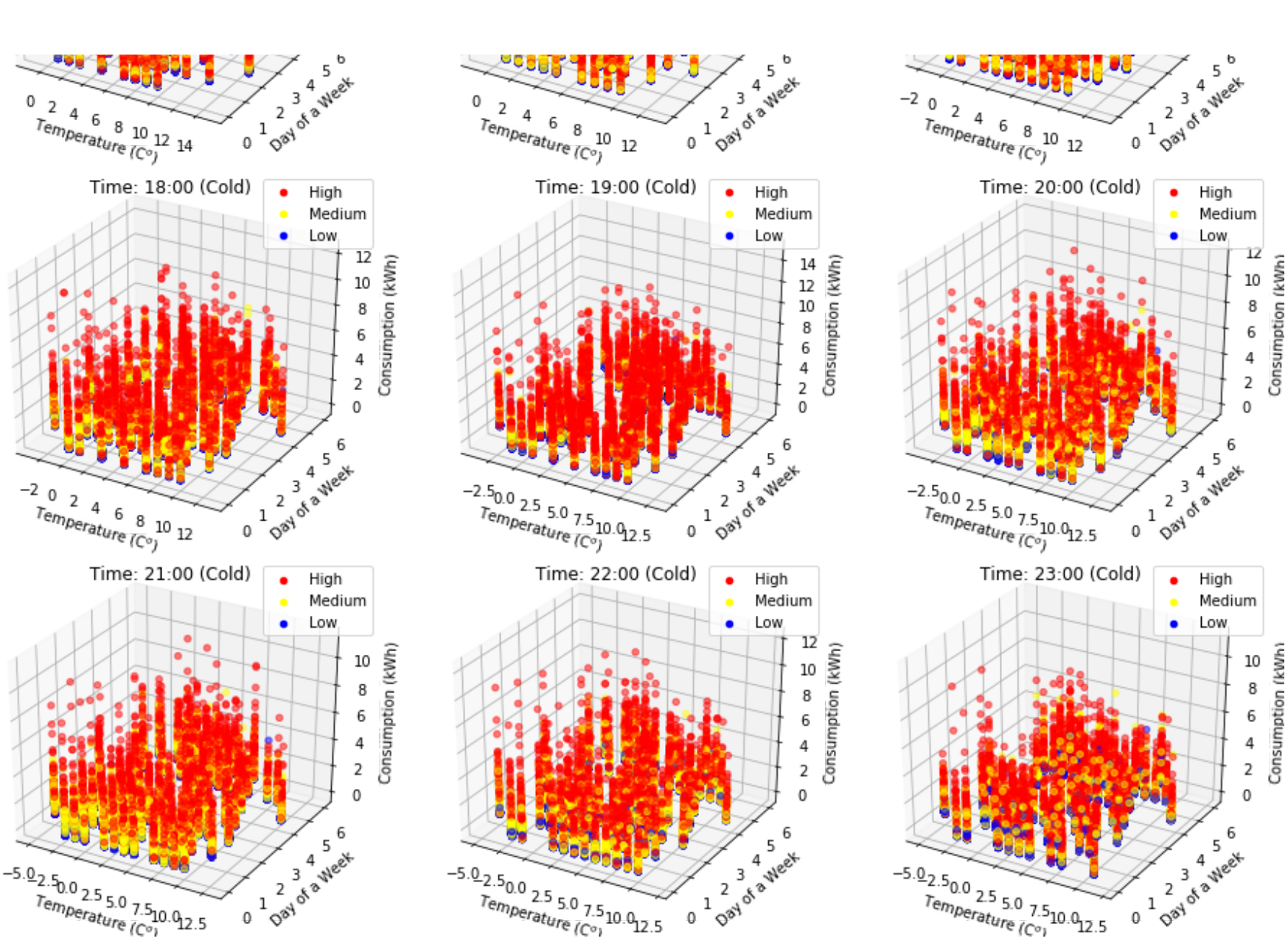

Time: 13:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)


Time: 14:00 (Cold)
High
Medium
Low
Temperature (C°)
Day of a Week
Consumption (kWh)


Time: 15:00 (Cold)
High
Medium
Low
Consumption (kWh)


Time: 16:00 (Cold)
High
Medium
Low
Consumption (kWh)


Time: 17:00 (Cold)
High
Medium
Low
Consumption (kWh)



Let us deal with the classes of households. Each household should have a label vector of 24 hours.

In [8]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
n_house = 1025 # total household numbers
houseLabel = [] # store the 24hour group labels for each hourshold
for i in range(1025):
    houseLabel.append('')
houseLabel = np.array(houseLabel)
cons_type = ['High Consumption', 'Medium Consumption', 'Low Consumption']
fig_all = plt.figure(figsize = (13,90))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.fillna(x.mean()).transpose() # data for the specific hour, and fill in null value with mean
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n, init='k-means++').fit(principleDf_nstd.iloc[:,:3])
        houseLabel = np.core.defchararray.add(houseLabel, kmeans.labels_.astype('str'))
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.fillna(x.mean()).transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n, init='k-means++').fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        houseLabel = np.core.defchararray.add(houseLabel, kmeans.labels_.astype('str'))
# We obtain the house Label shown below
# print(houseLabel)

# Group the house based on their 24 hour labels
houseLableDf = pd.DataFrame()
houseLableDf['House'] = df_tou1h_nf_cold.columns[1:]
houseLableDf['Label'] = houseLabel
houseGroupDf = houseLableDf.groupby('Label').size().reset_index(name='counts')
houseGroupDf = houseGroupDf.sort_values(by=['counts'], ascending=False)
houseGroupDf
```

Out[8]:

	Label	counts
275	10000000122100100012201 0	407
157	02221221000222221120020 1	28
237	10000000102100100012201 0	11
287	10000000122100100112201 0	10
183	10000000000222221120020 1	7
270	10000000122100100010201 0	6
286	10000000122100100022201 0	6
428	21112112211011012201112 2	5
64	02000000122100100012201 0	5
145	02221220000222221120020 1	5
361	10000020122100100012201 0	5
187	10000000002100100012201 0	4
294	10000000122100100122201 0	4
65	02000000122100100012201 1	4
271	10000000122100100012001 0	4
27	00000000122100100012201 0	4
302	10000000122100120012201 0	4
377	10000200122100100012201 0	4
204	10000000100100100012201 0	4
251	10000000120100100012201 0	3
77	02000001000222221120020 1	3
6	00000000000222221120020 1	3
295	10000000122100101012201 0	3
318	10000000122120100012201 0	3
41	00000001000222221120020 1	3
198	10000000022100100012201 0	3
15	00000000100222221120020 1	3
347	10000001122100100012201 0	3
300	10000000122100101120201 0	3
320	10000000122122100012201 0	3
...	...	...
127	02221121000222221120020 1	1
126	02221112211022012201120 2	1
125	02220221100221221101020 1	1
148	02221220100202221120020 1	1
150	02221220100221021120020 1	1
178	10000000000200200012201 0	1
166	02221221100222221120020 1	1
177	10000000000122221120001 0	1
176	10000000000122221112201 0	1
175	10000000000100221120020 1	1
174	02222121000222221221020 1	1
173	02222112000221012121020 1	1
172	02221222211011012201112 1	1
171	02221222210011012201110 1	1
170	02221221211222222201020 2	1
169	02221221211011012201112 1	1
167	02221221122100100012201 0	1
165	02221221011222221100020 1	1
151	02221220100222221012200 1	1

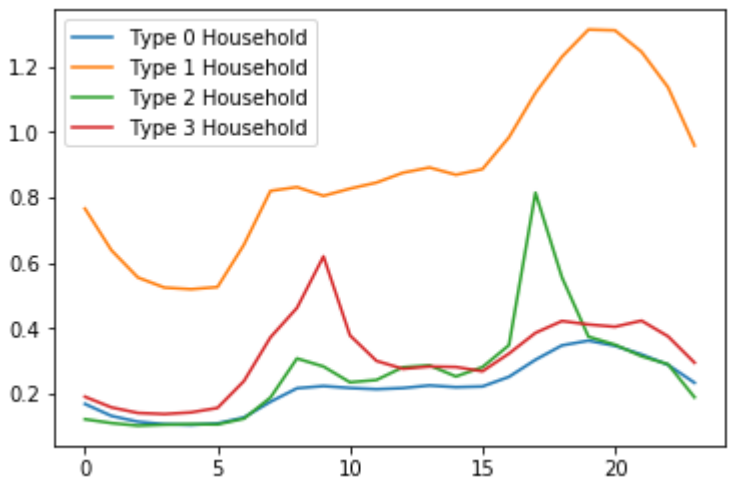
	Label	counts
164	022212210110110122011201	1
163	02221221011011012201020 1	1
162	022212210102212121200201	1
161	022212210010222111211101	1
160	022212210010210222011101	1
159	022212210002222212011101	1
155	022212210002202200120201	1
154	022212201221001000122011	1
153	022212201221001000120201	1
152	022212201002222211200201	1
453	222212212110110122011122	1

454 rows × 2 columns

```
<Figure size 936x6480 with 0 Axes>
```

Note that the generated labels for the first 3 groups may change a bit each time, this is because some group of points which are very close to each other also have equal distance two both cluster centers, which leads to different label names. However, this won't affect the group property, we will show we run it for 100 times, and average 24 hour load curve of every time will be ploted. And you will see, the average load curve almost doesn't change. So label names wouldn't be an important concern here, we more care about the property of groups themselves. In addition, from the above result we can see the first 4 groups each has more than 10 household in it. Next what we want to do is to plot the load property of each group, and see how does there average 24 hour load look like.

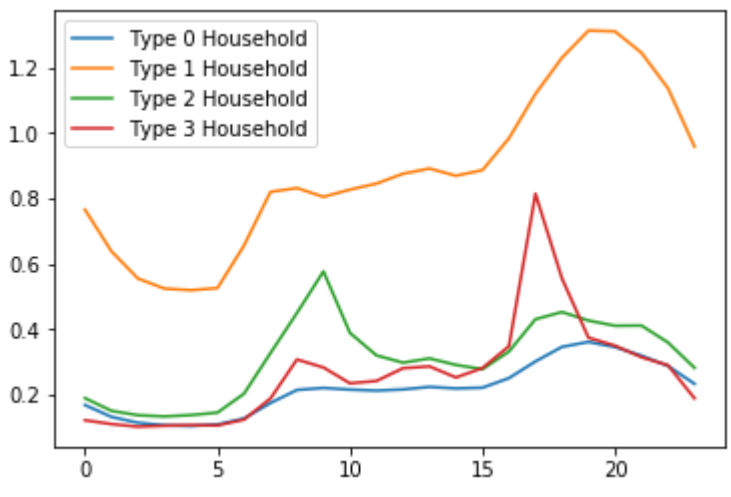
In [22]:
```
for i in range(4):
    load = []
    for j in range(24):
        if j <= 9:
            x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(j) + ':00:00')]
            x.set_index('GMT', inplace = True)
            x = x.fillna(x.mean())
            x = x[list(houseLableDf[houseLableDf.Label == houseGroupDf.Label.iloc[i]].House)]
            load.append(x.mean().mean())
        else:
            x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(j) + ':00:00')]
            x.set_index('GMT', inplace = True)
            x = x.fillna(x.mean())
            x = x[list(houseLableDf[houseLableDf.Label == houseGroupDf.Label.iloc[i]].House)]
            load.append(x.mean().mean())
    plt.plot(load, label = 'Type ' + str(i) + ' Household')
plt.legend()
plt.show()
```



Note: the above grouping process are based on kmean-clustering, it's worth mentioning that, the kmean-clustering results highly realted to the initial value randomization. And it seems that this randomization affects the household 24 hour group significantly. So this means that it may not be an appropriate approach to use k-mean clustering algorithm here for the household grouping.

For example, if we specify the kmean to be randomized initilization, and see if the final main groups have a significant difference.

In [25]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 3 # cluster numbers
n_house = 1025 # total household numbers
houseLabel = [] # store the 24hour group labels for each hourshold
for i in range(1025):
    houseLabel.append('')
houseLabel = np.array(houseLabel)
cons_type = ['High Consumption', 'Medium Consumption', 'Low Consumption']
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.fillna(x.mean()).transpose() # data for the specific hour, and fill in null value with mean
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n, init='k-means++').fit(principleDf_nstd.iloc[:,:3])
        houseLabel = np.core.defchararray.add(houseLabel, kmeans.labels_.astype('str'))
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        x.set_index('GMT', inplace = True)
        x = x.fillna(x.mean()).transpose() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n, init='k-means++').fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        houseLabel = np.core.defchararray.add(houseLabel, kmeans.labels_.astype('str'))
# We obtain the house Label shown below
# print(houseLabel)

# Group the house based on their 24 hour labels
houseLableDf = pd.DataFrame()
houseLableDf['House'] = df_tou1h_nf_cold.columns[1:]
houseLableDf['Label'] = houseLabel
houseGroupDf = houseLableDf.groupby('Label').size().reset_index(name='counts')
houseGroupDf = houseGroupDf.sort_values(by=['counts'], ascending=False)

for i in range(4):
    load = []
    for j in range(24):
        if j <= 9:
            x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(j) + ':00:00')]
            x.set_index('GMT', inplace = True)
            x = x.fillna(x.mean())
            x = x[list(houseLableDf[houseLableDf.Label == houseGroupDf.Label.iloc[i]].House)]
            load.append(x.mean().mean())
        else:
            x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(j) + ':00:00')]
            x.set_index('GMT', inplace = True)
            x = x.fillna(x.mean())
            x = x[list(houseLableDf[houseLableDf.Label == houseGroupDf.Label.iloc[i]].House)]
            load.append(x.mean().mean())
    plt.plot(load, label = 'Type ' + str(i) + ' Household')
plt.legend()
plt.show()
```



Some observations and thoughts of the above daily load curves:

1) the type 0 household with the most amount of households has a relatively smooth and low consumption during a day, they consume the most amount of energy around 19:00, and consume least at 02:00-05:00.

2) type 1 household is high energy consumption households, besides of all the consumption is around 4 times the consumption of type 0 group, they have a relatively similar trend of energy usage pattern peak at 19:00 and valley at 02:00-05:00

3) type 2 household has a special usage pattern where their peak happens at 17:00 (maybe they come back home earlier), other parts about the same.

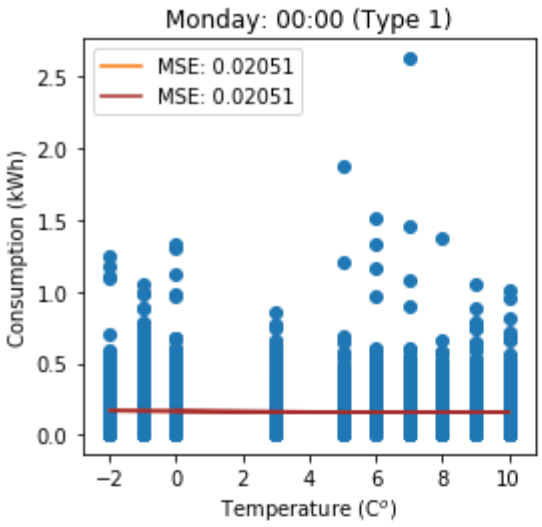
4) type 3 household also has a special usage pattern, where their peak happens at 09:00 (maybe they work in the morning, at home with some works that consumes a lot of energy, maybe breakfast, maybe heating at every morning?)

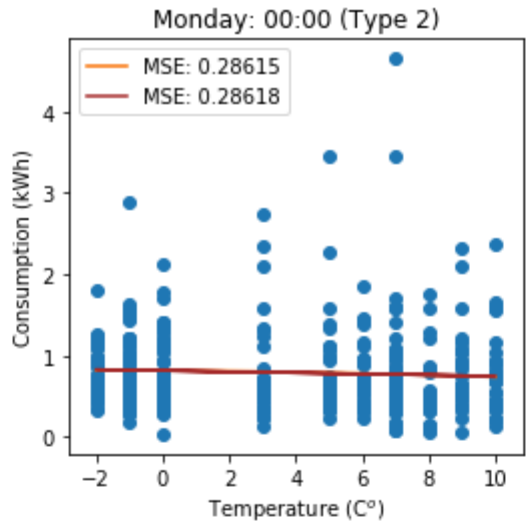
3. Multiple Regression

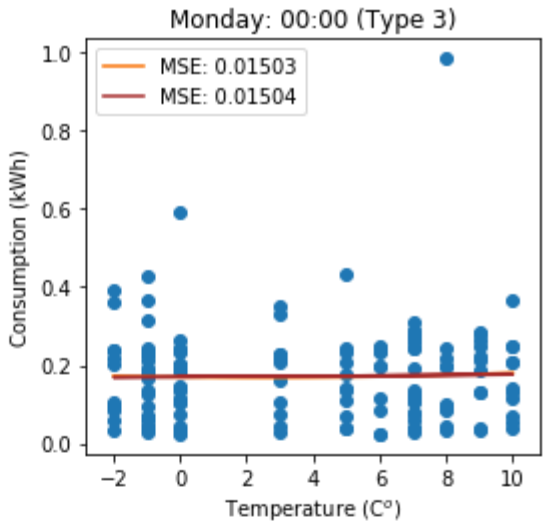
Multiple Regression is to predict the consumption of the group in event (pricing change) time periods, so that we can analyze the price responsivness later.

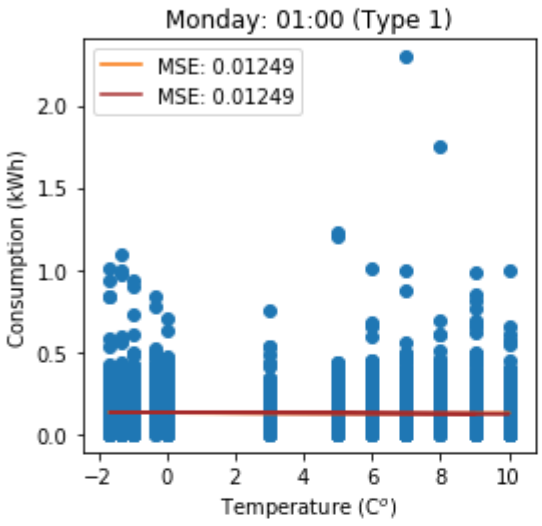
It is clear that the temperature and days of a weeks are kinda correlated with the the first 3 PCA coefficients, we are using them as the features to do multiple regression (for each hour, temperature and days of week do regression).

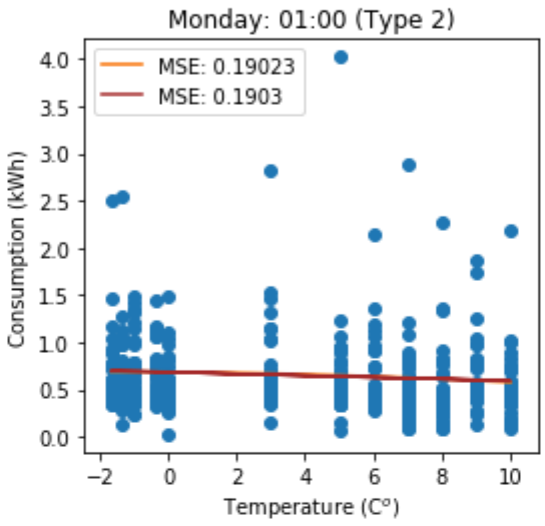
In [125]:
```
# Monday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 0 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for Monday
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for Monday
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

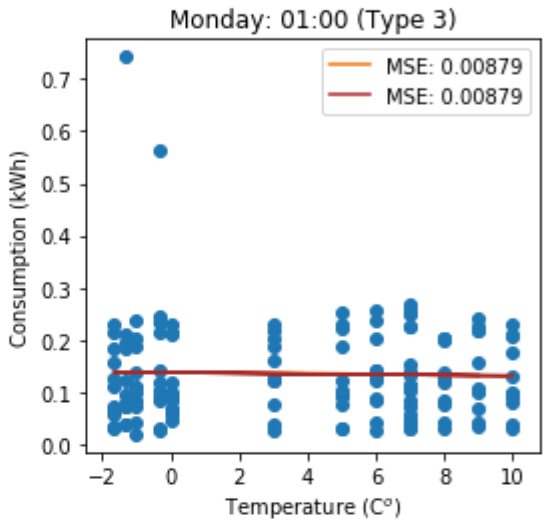

Monday: 00:00 (Type 1)
MSE: 0.02051
MSE: 0.02051
Consumption (kWh)
Temperature (C°)

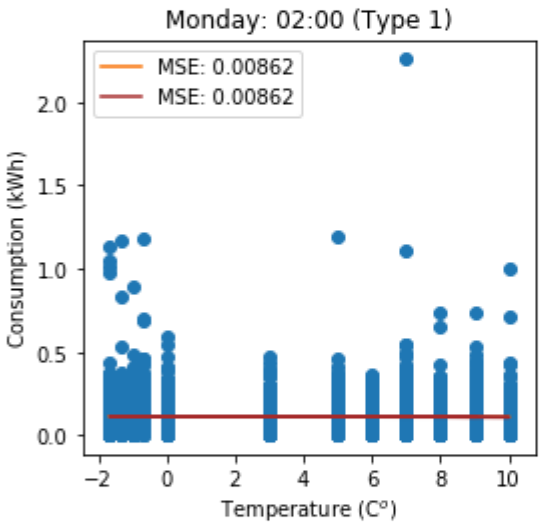

Monday: 00:00 (Type 2)
MSE: 0.28615
MSE: 0.28618
Consumption (kWh)
Temperature (C°)

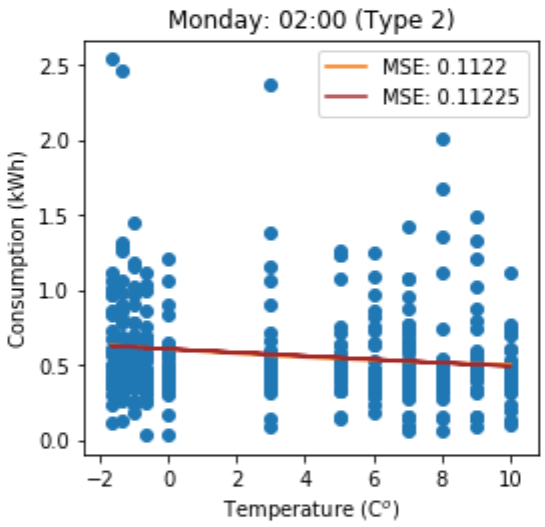

Monday: 00:00 (Type 3)
MSE: 0.01503
MSE: 0.01504
Consumption (kWh)
Temperature (C°)

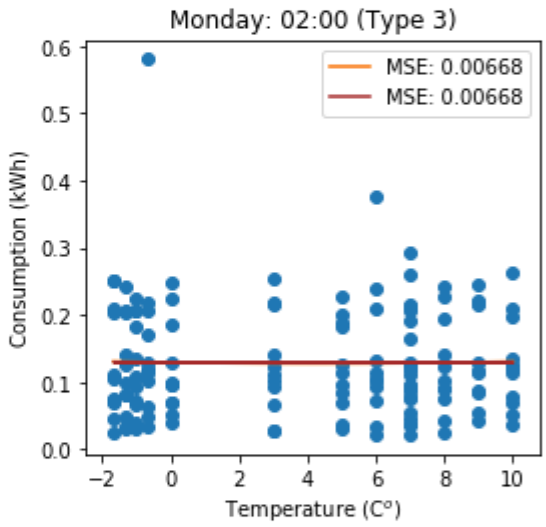

Monday: 01:00 (Type 1)
MSE: 0.01249
MSE: 0.01249
Consumption (kWh)
Temperature (C°)

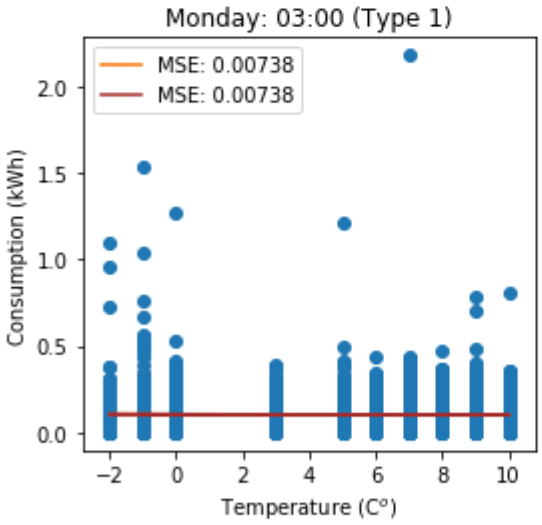

Monday: 01:00 (Type 2)
MSE: 0.19023
MSE: 0.1903
Consumption (kWh)
Temperature (C°)

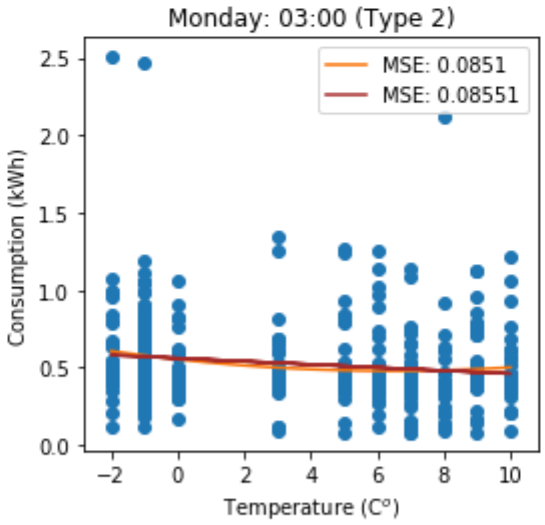

Monday: 01:00 (Type 3)
MSE: 0.00879
MSE: 0.00879
Consumption (kWh)
Temperature (C°)

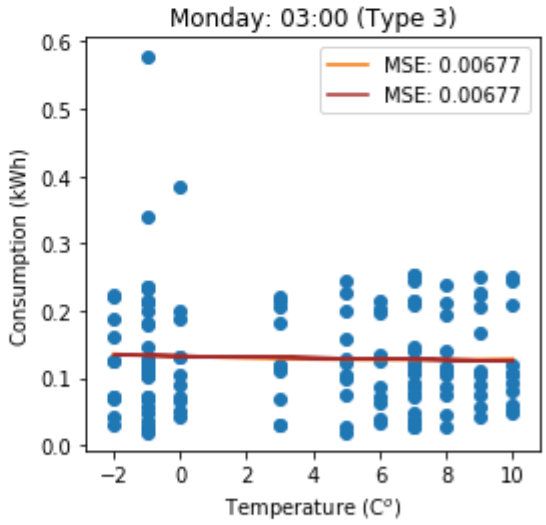

Monday: 02:00 (Type 1)
MSE: 0.00862
MSE: 0.00862
Consumption (kWh)
Temperature (C°)

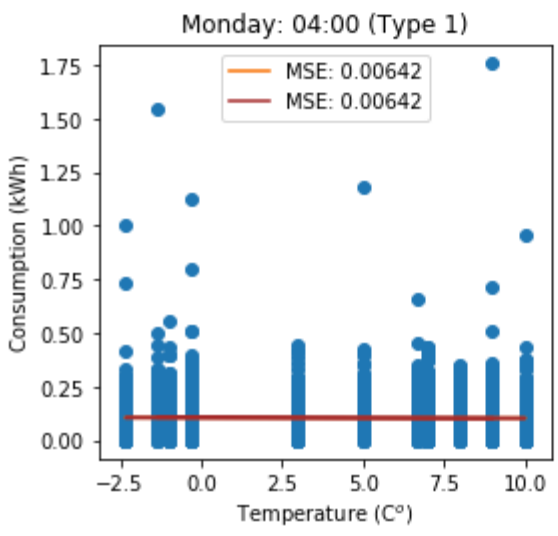

Monday: 02:00 (Type 2)
MSE: 0.1122
MSE: 0.11225
Consumption (kWh)
Temperature (C°)

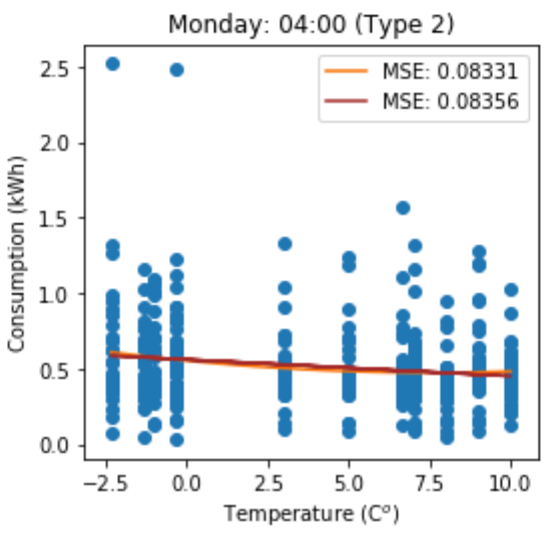

Monday: 02:00 (Type 3)
MSE: 0.00668
MSE: 0.00668
Consumption (kWh)
Temperature (C°)

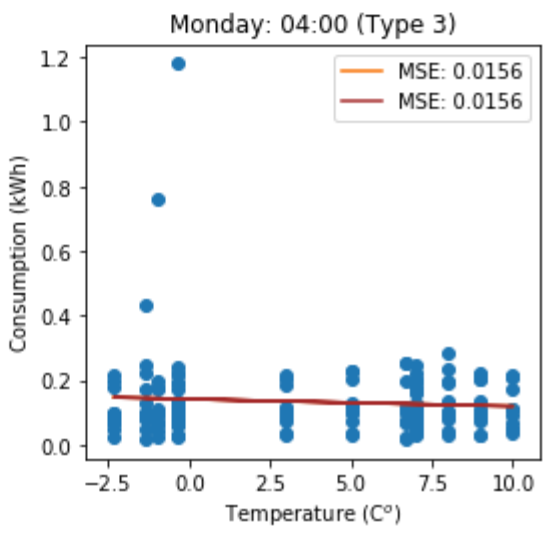

Monday: 03:00 (Type 1)
MSE: 0.00738
MSE: 0.00738
Consumption (kWh)
Temperature (C°)


Monday: 03:00 (Type 2)
MSE: 0.0851
MSE: 0.08551
Consumption (kWh)
Temperature (C°)


Monday: 03:00 (Type 3)
MSE: 0.00677
MSE: 0.00677
Consumption (kWh)
Temperature (C°)


Monday: 04:00 (Type 1)
MSE: 0.00642
MSE: 0.00642
Consumption (kWh)
Temperature (C°)

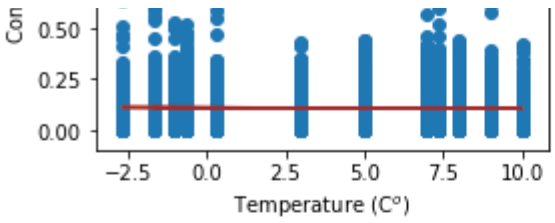

Monday: 04:00 (Type 2)
MSE: 0.08331
MSE: 0.08356
Consumption (kWh)
Temperature (C°)

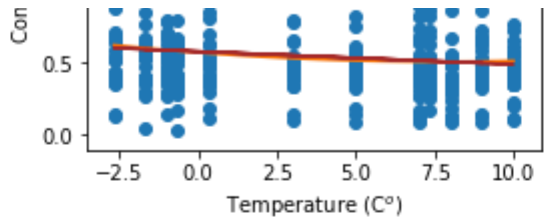

Monday: 04:00 (Type 3)
MSE: 0.0156
MSE: 0.0156
Consumption (kWh)
Temperature (C°)

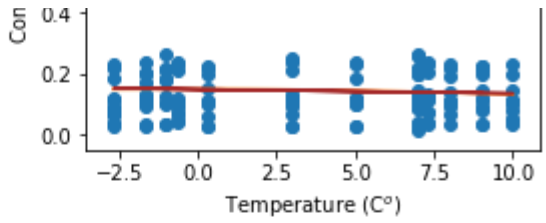
Monday: 05:00 (Type 1)
MSE: 0.00795
MSE: 0.00795

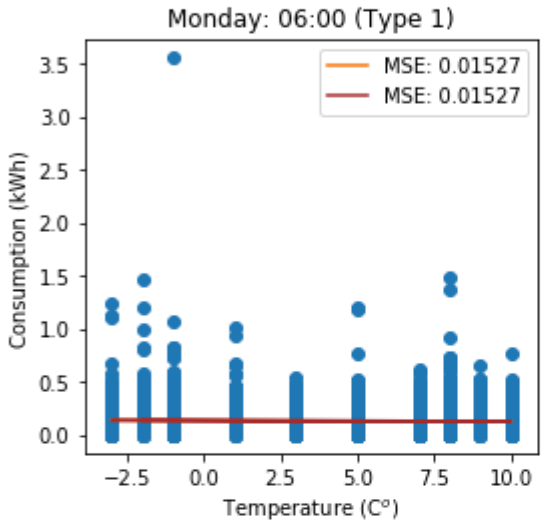
Monday: 05:00 (Type 2)
MSE: 0.09728
MSE: 0.09745

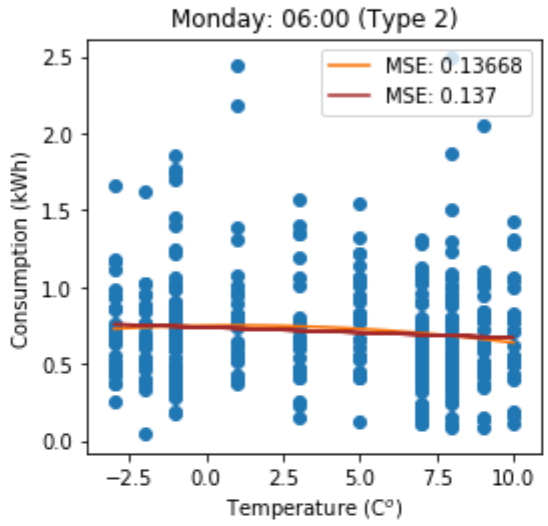
Monday: 05:00 (Type 3)
MSE: 0.01722
MSE: 0.01722

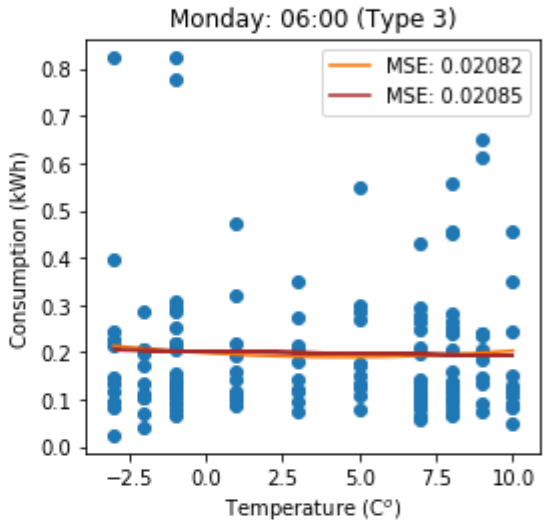

Con
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

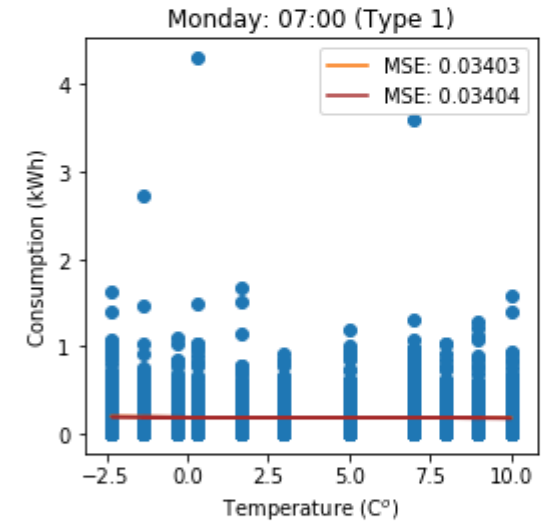

Con
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

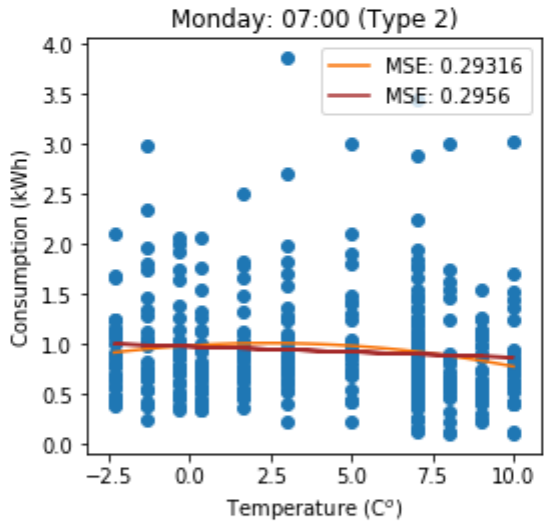

Con
0.4
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

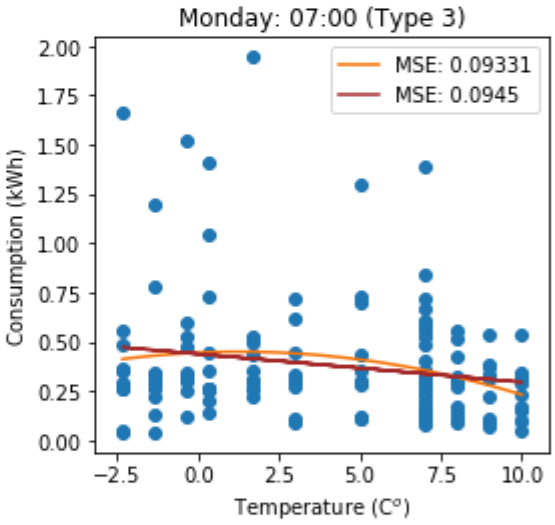

Monday: 06:00 (Type 1)
MSE: 0.01527
MSE: 0.01527
Consumption (kWh)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

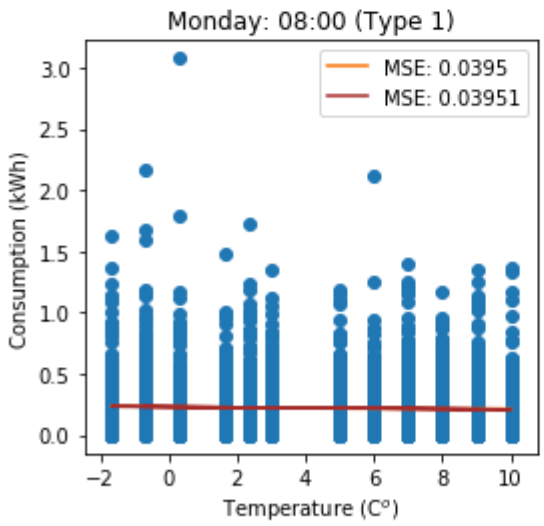

Monday: 06:00 (Type 2)
MSE: 0.13668
MSE: 0.137
Consumption (kWh)
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

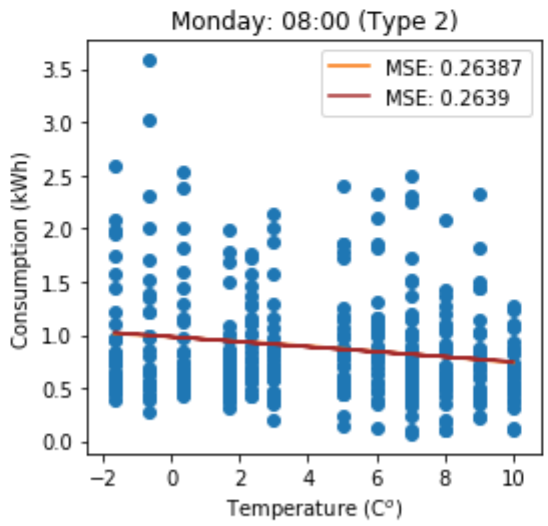

Monday: 06:00 (Type 3)
MSE: 0.02082
MSE: 0.02085
Consumption (kWh)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

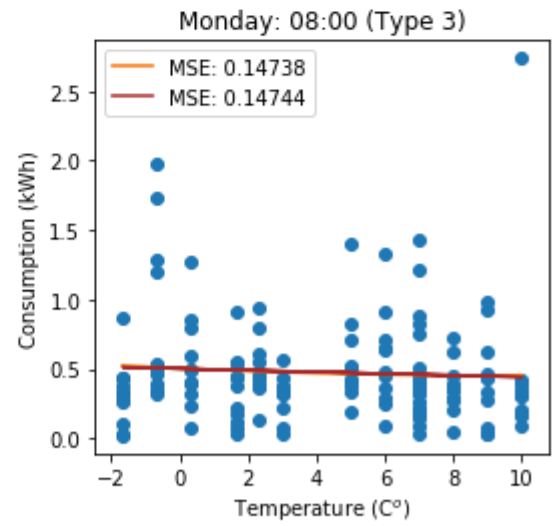

Monday: 07:00 (Type 1)
MSE: 0.03403
MSE: 0.03404
Consumption (kWh)
4
3
2
1
0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

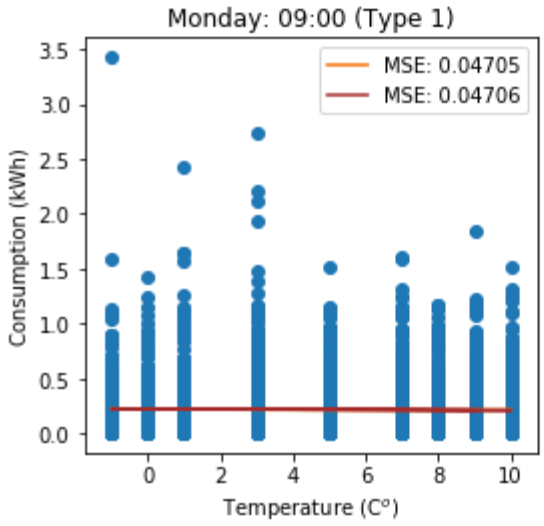

Monday: 07:00 (Type 2)
MSE: 0.29316
MSE: 0.2956
Consumption (kWh)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

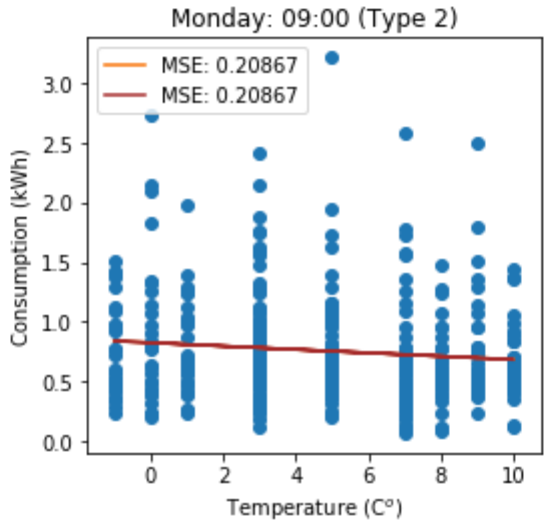

Monday: 07:00 (Type 3)
MSE: 0.09331
MSE: 0.0945
Consumption (kWh)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

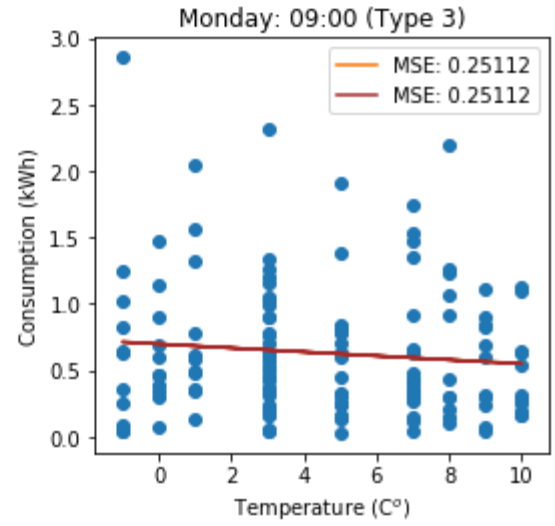

Monday: 08:00 (Type 1)
MSE: 0.0395
MSE: 0.03951
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2
0
2
4
6
8
10
Temperature (C°)

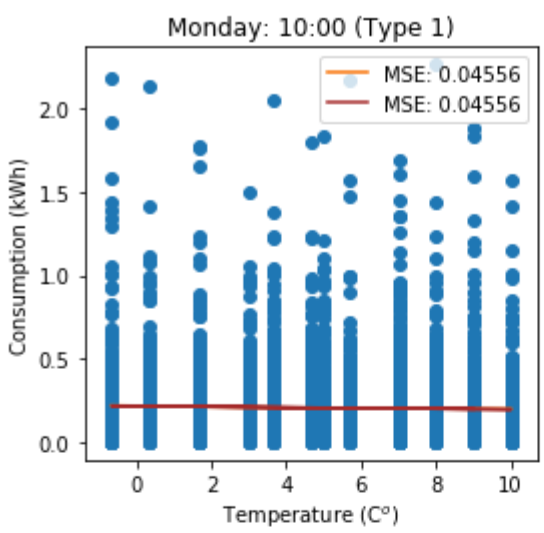

Monday: 08:00 (Type 2)
MSE: 0.26387
MSE: 0.2639
Consumption (kWh)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2
0
2
4
6
8
10
Temperature (C°)

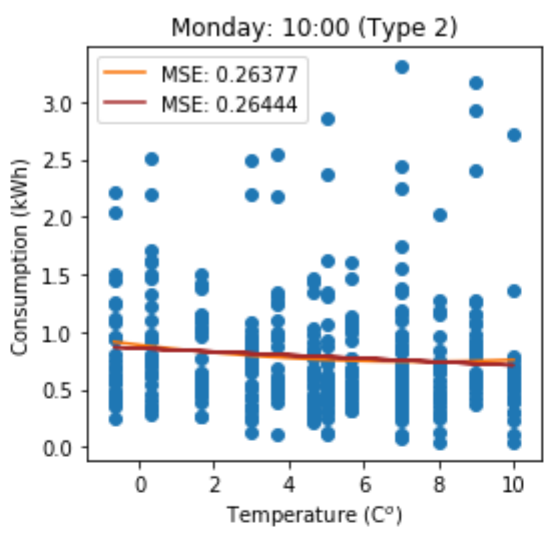

Monday: 08:00 (Type 3)
MSE: 0.14738
MSE: 0.14744
Consumption (kWh)
2.5
2.0
1.5
1.0
0.5
0.0
−2
0
2
4
6
8
10
Temperature (C°)

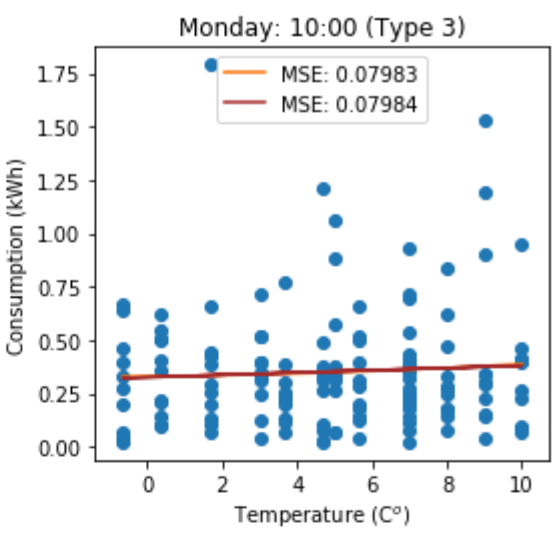

Monday: 09:00 (Type 1)
MSE: 0.04705
MSE: 0.04706
Consumption (kWh)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Temperature (C°)


Monday: 09:00 (Type 2)
MSE: 0.20867
MSE: 0.20867
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Temperature (C°)

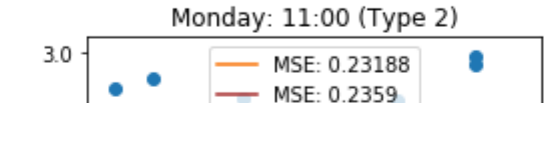

Monday: 09:00 (Type 3)
MSE: 0.25112
MSE: 0.25112
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Temperature (C°)


Monday: 10:00 (Type 1)
MSE: 0.04556
MSE: 0.04556
Consumption (kWh)
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Temperature (C°)

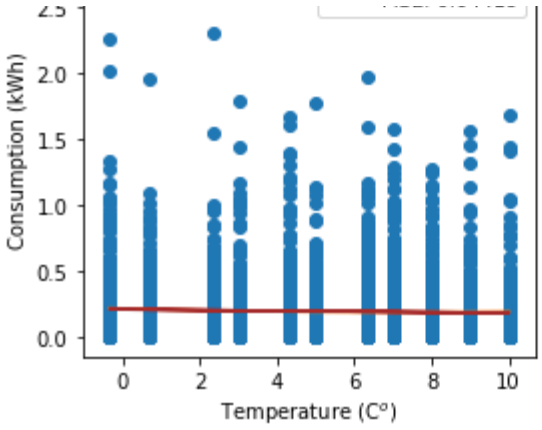

Monday: 10:00 (Type 2)
MSE: 0.26377
MSE: 0.26444
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Temperature (C°)

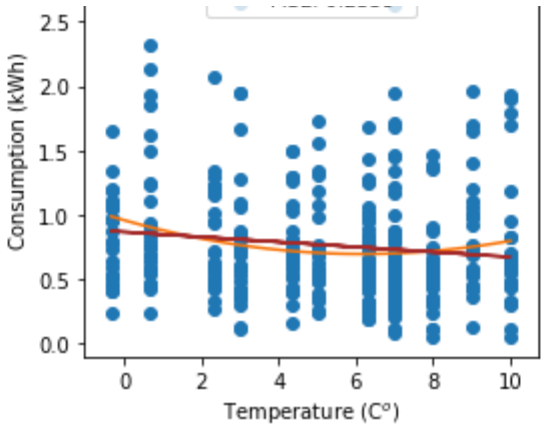

Monday: 10:00 (Type 3)
MSE: 0.07983
MSE: 0.07984
Consumption (kWh)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
Temperature (C°)

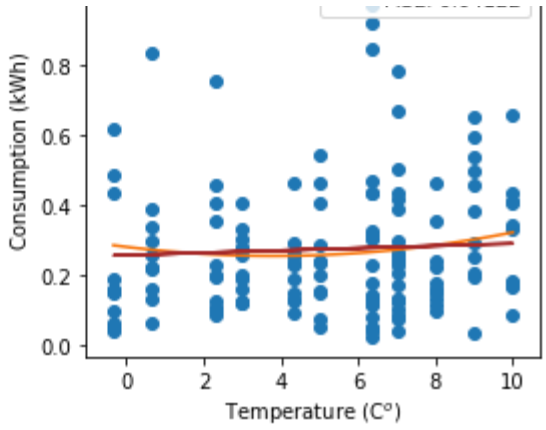

Monday: 11:00 (Type 1)
MSE: 0.04417
MSE: 0.04419
Consumption (kWh)
3.0

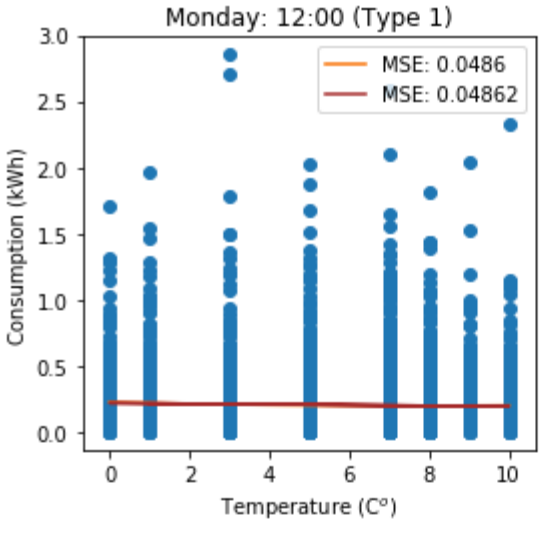

Monday: 11:00 (Type 2)
MSE: 0.23188
MSE: 0.2359
3.0

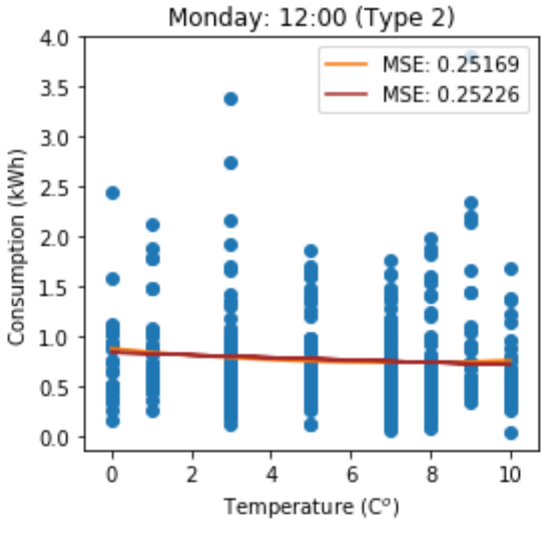

Monday: 11:00 (Type 3)
MSE: 0.04096
MSE: 0.04122
1.0

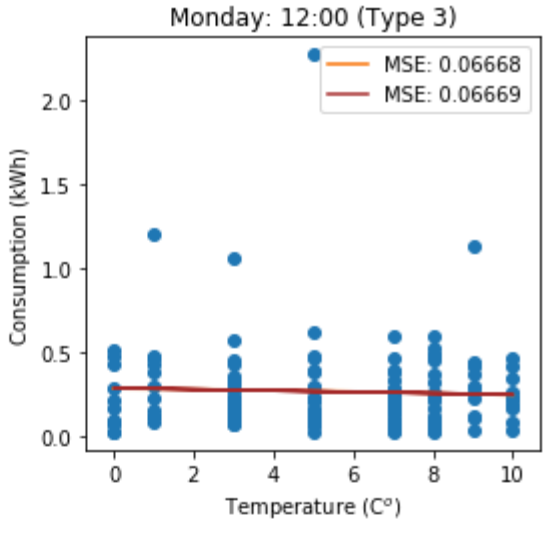

Consumption (kWh)
Temperature (C°)

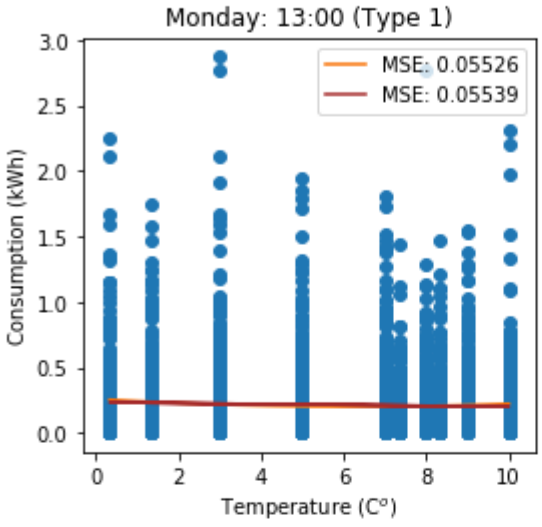

Consumption (kWh)
Temperature (C°)

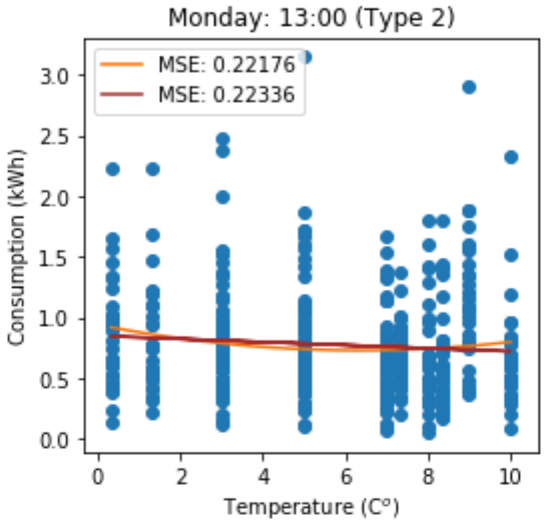

Consumption (kWh)
Temperature (C°)

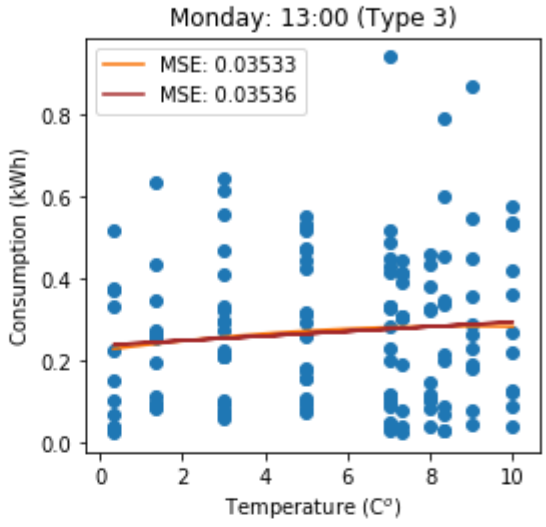

Monday: 12:00 (Type 1)
MSE: 0.0486
MSE: 0.04862
Consumption (kWh)
Temperature (C°)

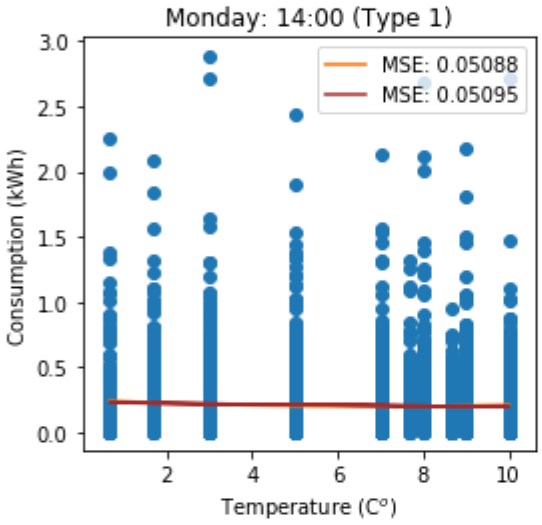

Monday: 12:00 (Type 2)
MSE: 0.25169
MSE: 0.25226
Consumption (kWh)
Temperature (C°)

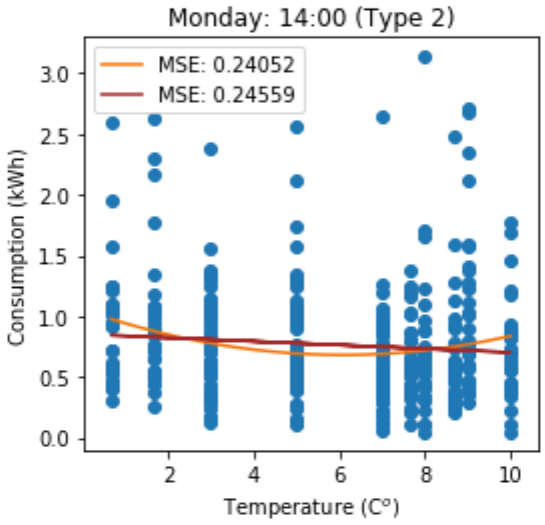

Monday: 12:00 (Type 3)
MSE: 0.06668
MSE: 0.06669
Consumption (kWh)
Temperature (C°)

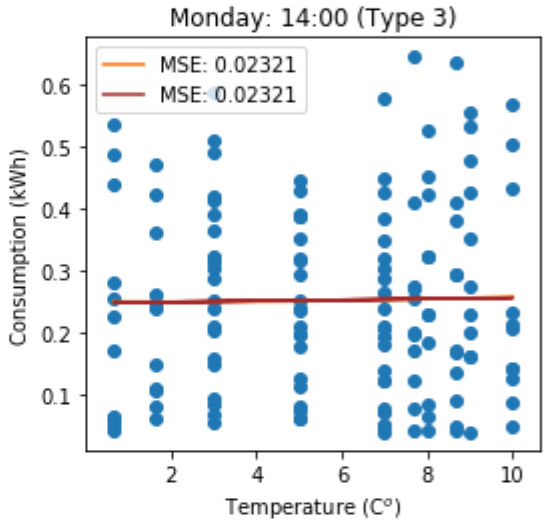

Monday: 13:00 (Type 1)
MSE: 0.05526
MSE: 0.05539
Consumption (kWh)
Temperature (C°)

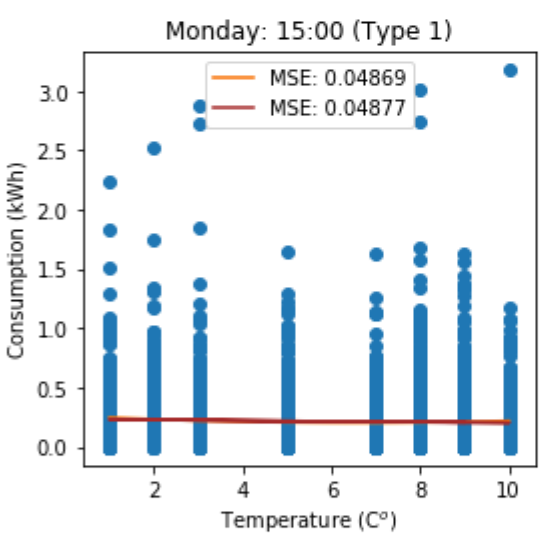

Monday: 13:00 (Type 2)
MSE: 0.22176
MSE: 0.22336
Consumption (kWh)
Temperature (C°)

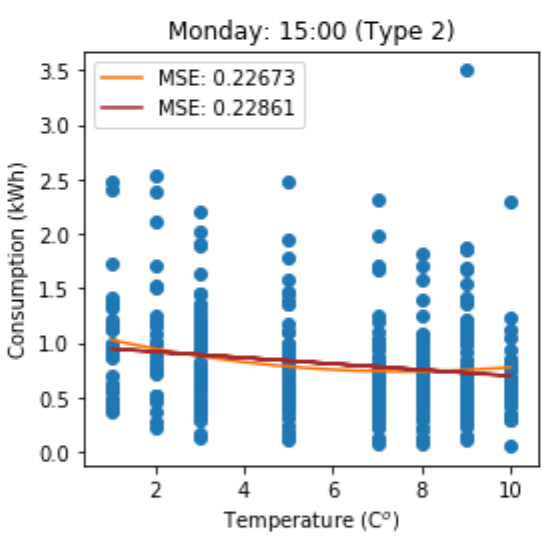

Monday: 13:00 (Type 3)
MSE: 0.03533
MSE: 0.03536
Consumption (kWh)
Temperature (C°)

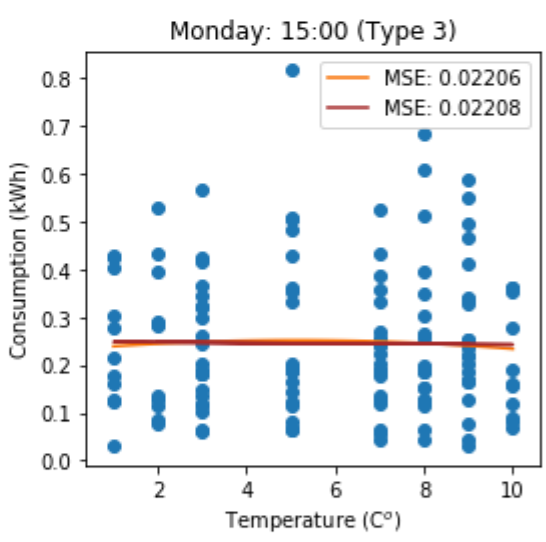

Monday: 14:00 (Type 1)
MSE: 0.05088
MSE: 0.05095
Consumption (kWh)
Temperature (C°)

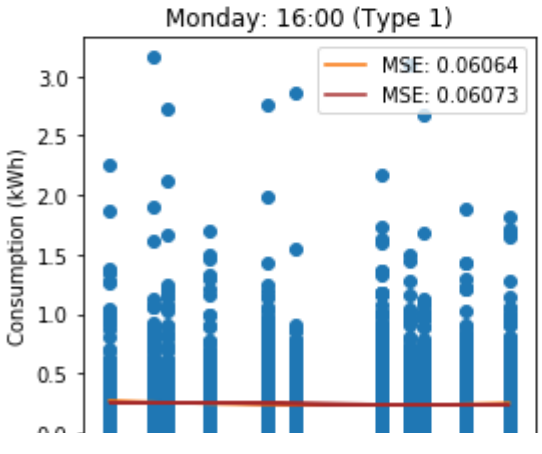

Monday: 14:00 (Type 2)
MSE: 0.24052
MSE: 0.24559
Consumption (kWh)
Temperature (C°)

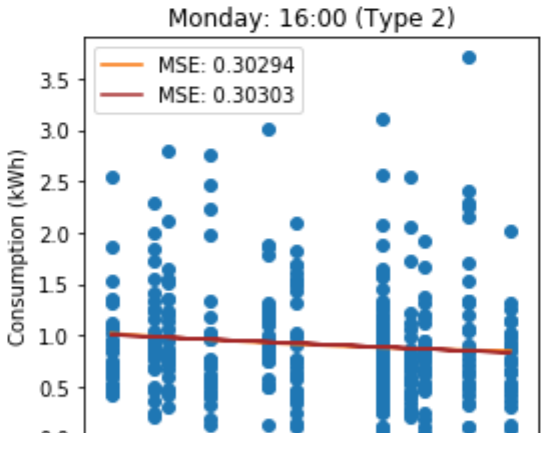

Monday: 14:00 (Type 3)
MSE: 0.02321
MSE: 0.02321
Consumption (kWh)
Temperature (C°)

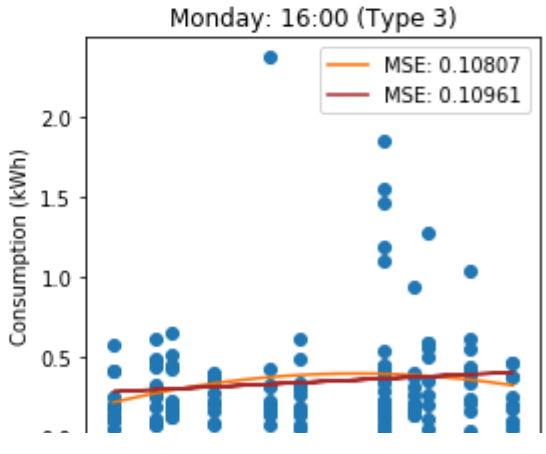

Monday: 15:00 (Type 1)
MSE: 0.04869
MSE: 0.04877
Consumption (kWh)
Temperature (C°)

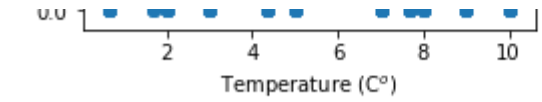

Monday: 15:00 (Type 2)
MSE: 0.22673
MSE: 0.22861
Consumption (kWh)
Temperature (C°)

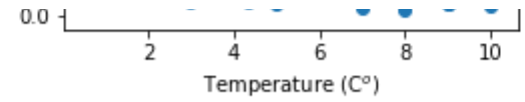

Monday: 15:00 (Type 3)
MSE: 0.02206
MSE: 0.02208
Consumption (kWh)
Temperature (C°)

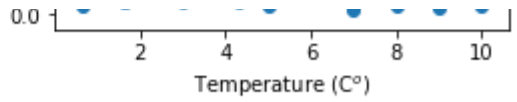

Monday: 16:00 (Type 1)
MSE: 0.06064
MSE: 0.06073
Consumption (kWh)

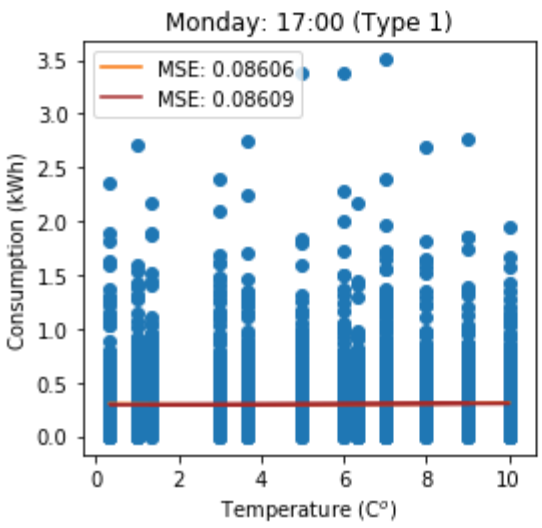

Monday: 16:00 (Type 2)
MSE: 0.30294
MSE: 0.30303
Consumption (kWh)

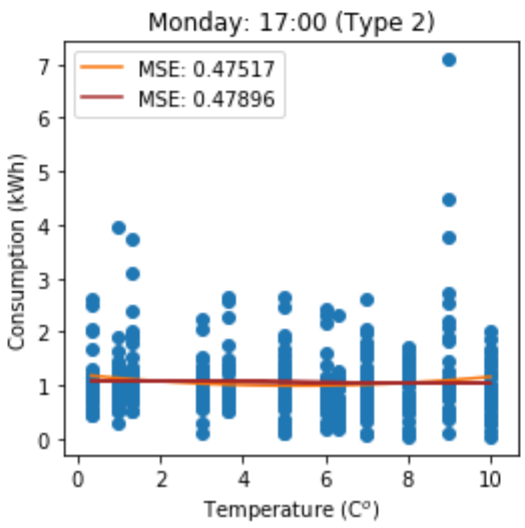

Monday: 16:00 (Type 3)
MSE: 0.10807
MSE: 0.10961
Consumption (kWh)

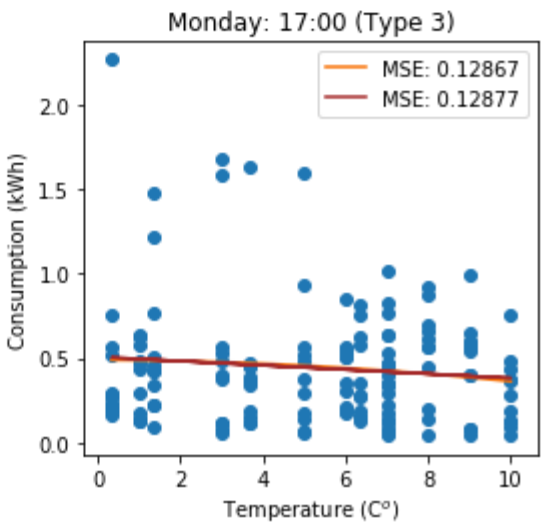

Temperature (C°)

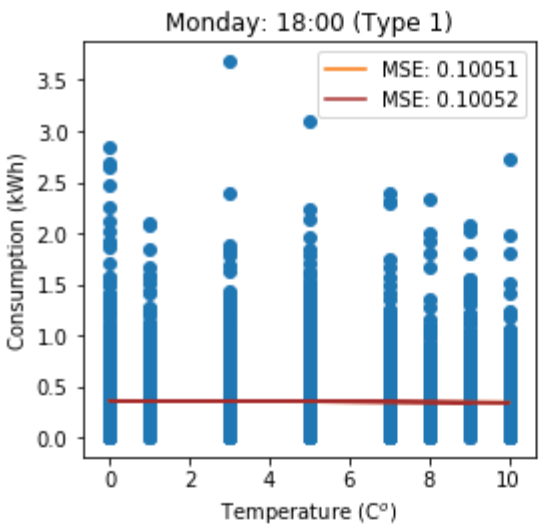

Temperature (C°)

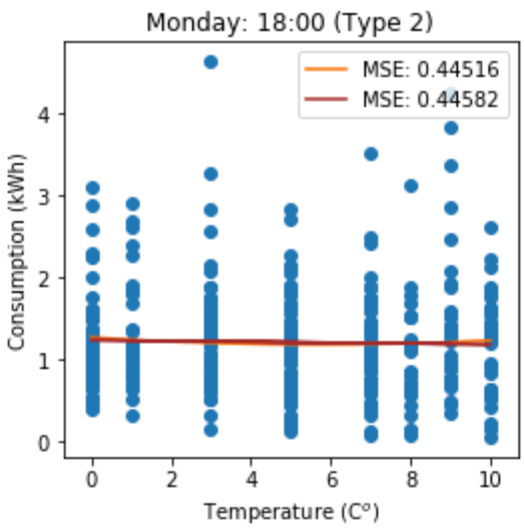

Temperature (C°)

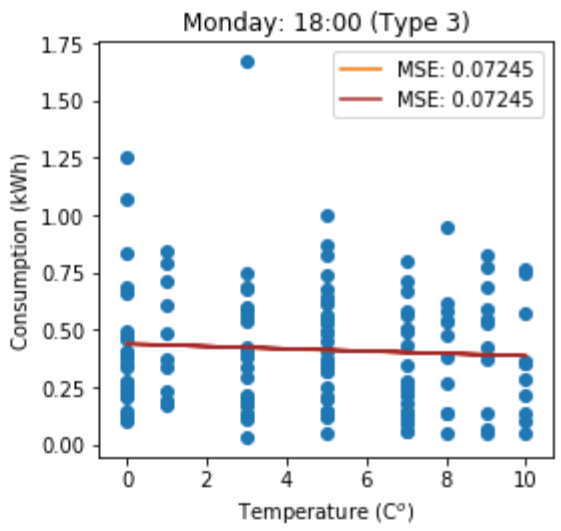

Monday: 17:00 (Type 1)
MSE: 0.08606
MSE: 0.08609
Consumption (kWh)
Temperature (C°)

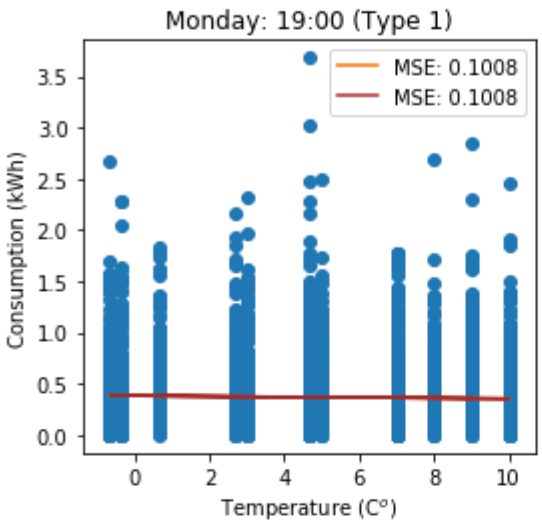

Monday: 17:00 (Type 2)
MSE: 0.47517
MSE: 0.47896
Consumption (kWh)
Temperature (C°)

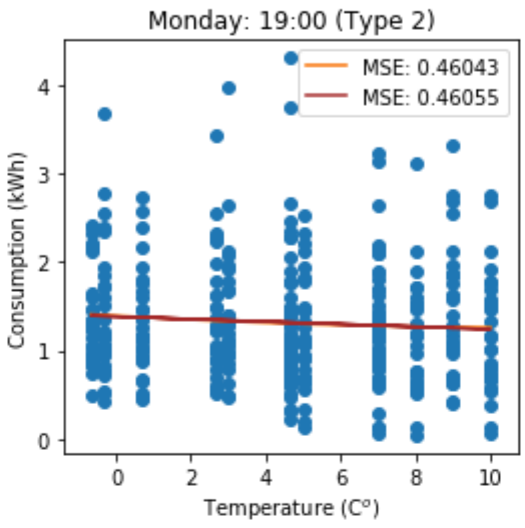

Monday: 17:00 (Type 3)
MSE: 0.12867
MSE: 0.12877
Consumption (kWh)
Temperature (C°)

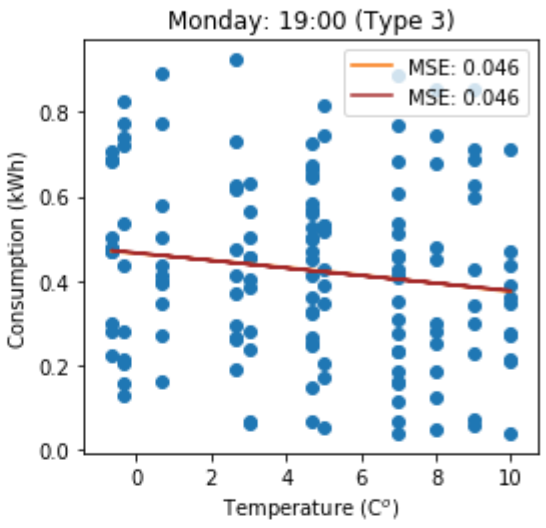

Monday: 18:00 (Type 1)
MSE: 0.10051
MSE: 0.10052
Consumption (kWh)
Temperature (C°)

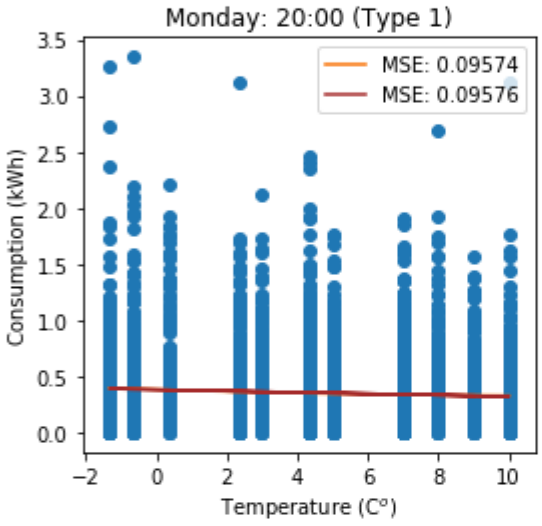

Monday: 18:00 (Type 2)
MSE: 0.44516
MSE: 0.44582
Consumption (kWh)
Temperature (C°)

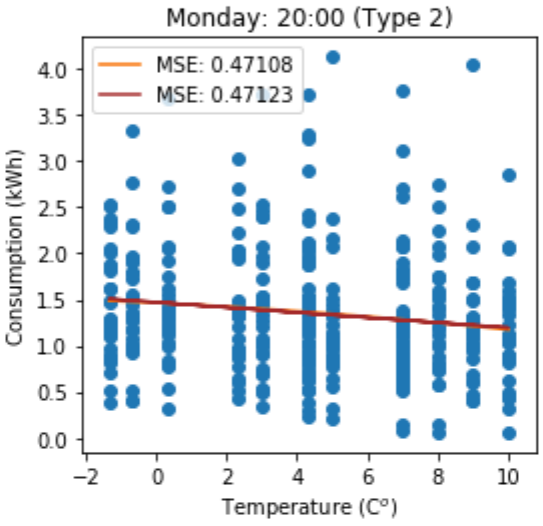

Monday: 18:00 (Type 3)
MSE: 0.07245
MSE: 0.07245
Consumption (kWh)
Temperature (C°)

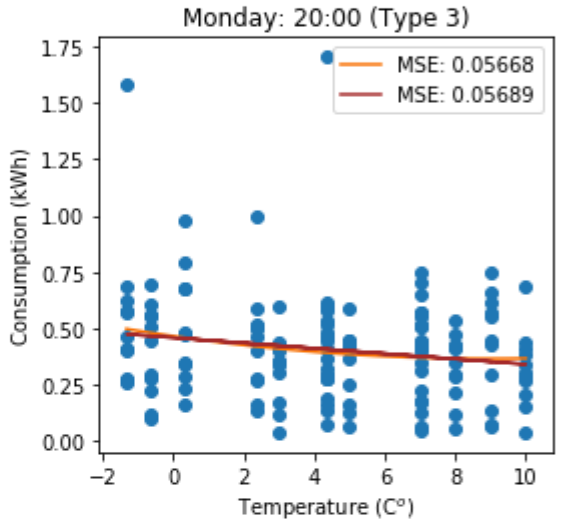

Monday: 19:00 (Type 1)
MSE: 0.1008
MSE: 0.1008
Consumption (kWh)
Temperature (C°)

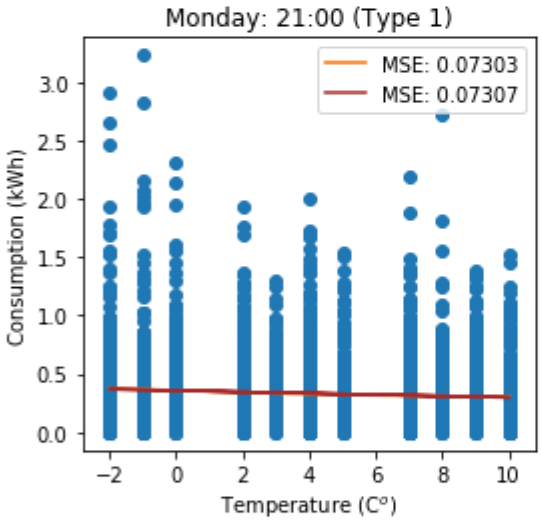

Monday: 19:00 (Type 2)
MSE: 0.46043
MSE: 0.46055
Consumption (kWh)
Temperature (C°)

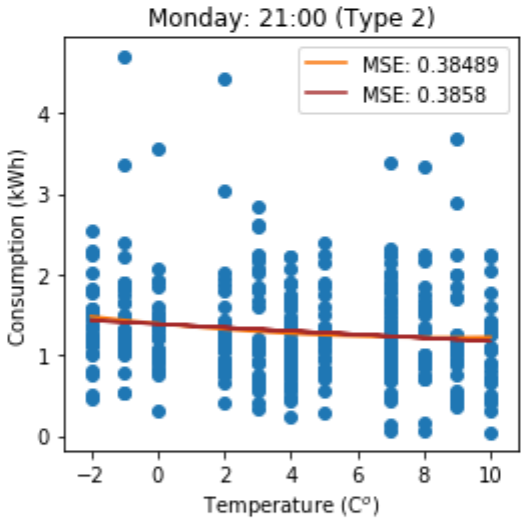

Monday: 19:00 (Type 3)
MSE: 0.046
MSE: 0.046
Consumption (kWh)
Temperature (C°)

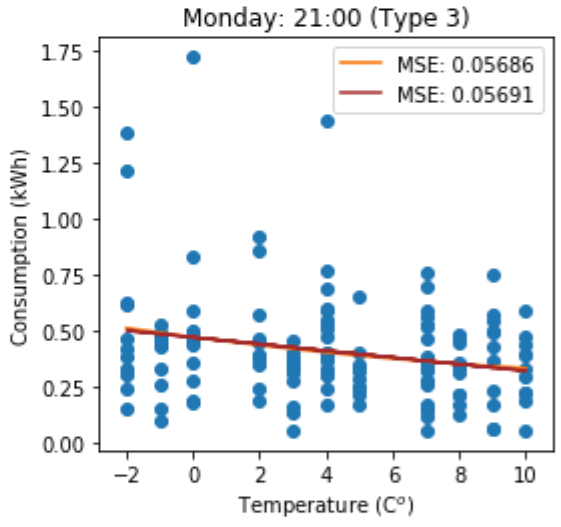

Monday: 20:00 (Type 1)
MSE: 0.09574
MSE: 0.09576
Consumption (kWh)
Temperature (C°)


Monday: 20:00 (Type 2)
MSE: 0.47108
MSE: 0.47123
Consumption (kWh)
Temperature (C°)


Monday: 20:00 (Type 3)
MSE: 0.05668
MSE: 0.05689
Consumption (kWh)
Temperature (C°)


Monday: 21:00 (Type 1)
MSE: 0.07303
MSE: 0.07307
Consumption (kWh)
Temperature (C°)

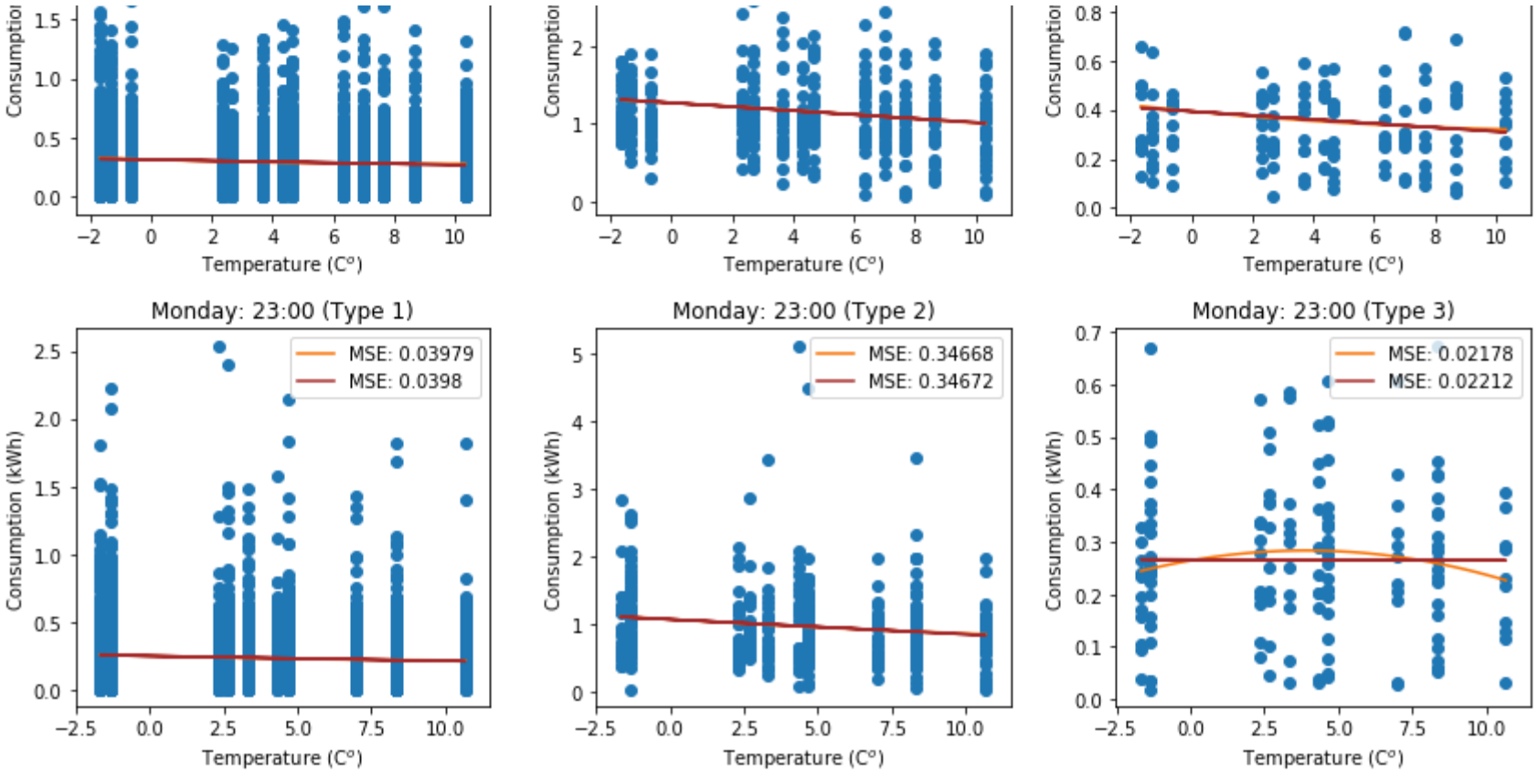

Monday: 21:00 (Type 2)
MSE: 0.38489
MSE: 0.3858
Consumption (kWh)
Temperature (C°)


Monday: 21:00 (Type 3)
MSE: 0.05686
MSE: 0.05691
Consumption (kWh)
Temperature (C°)

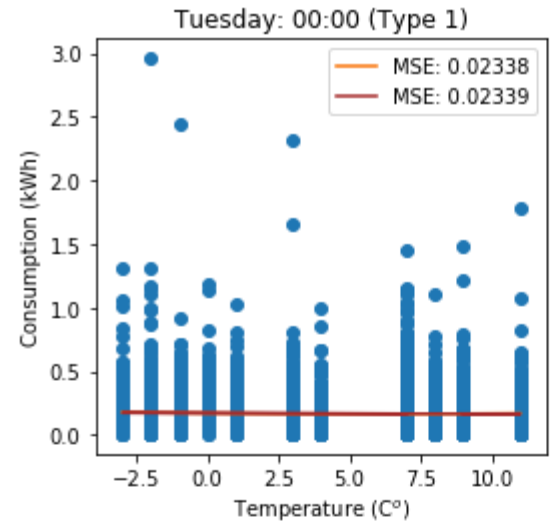
Monday: 22:00 (Type 1)
MSE: 0.05073
MSE: 0.05076
Temperature (C°)

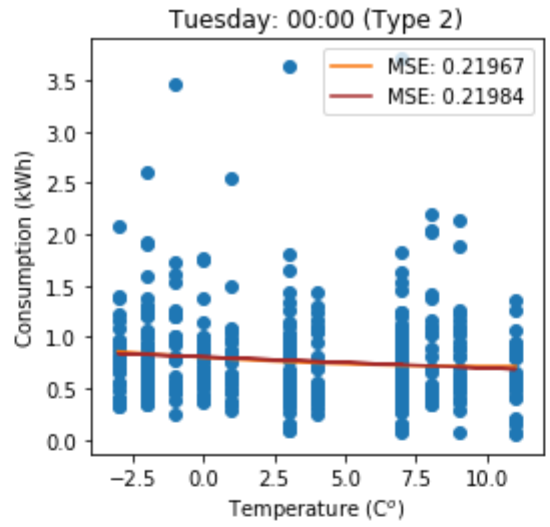
Monday: 22:00 (Type 2)
MSE: 0.34142
MSE: 0.34143
Temperature (C°)

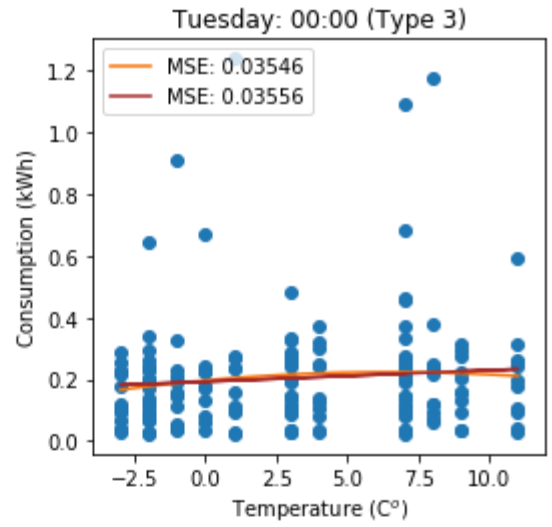
Monday: 22:00 (Type 3)
MSE: 0.0487
MSE: 0.04873

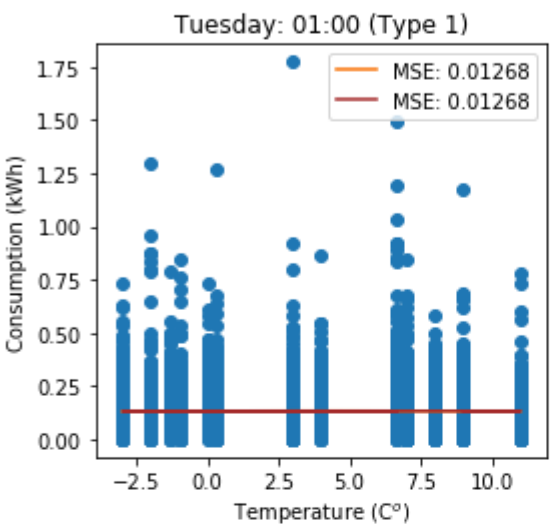

Monday: 23:00 (Type 1)
MSE: 0.03979
MSE: 0.0398
Monday: 23:00 (Type 2)
MSE: 0.34668
MSE: 0.34672
Monday: 23:00 (Type 3)
MSE: 0.02178
MSE: 0.02212
Consumption (kWh)
Temperature (C°)

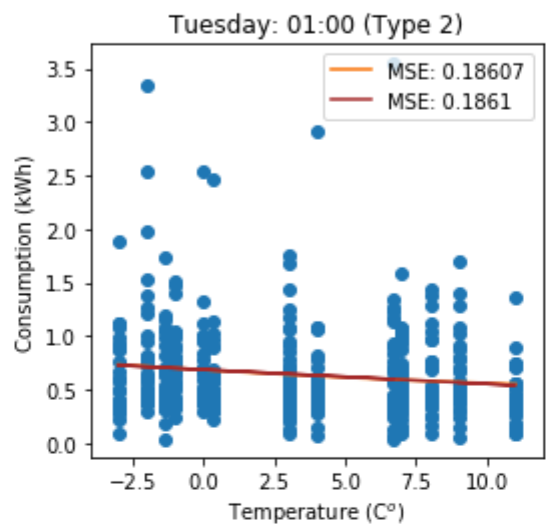
In [9]:
```
# Tuesday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 1 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for Tuesday
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for Tuesday
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

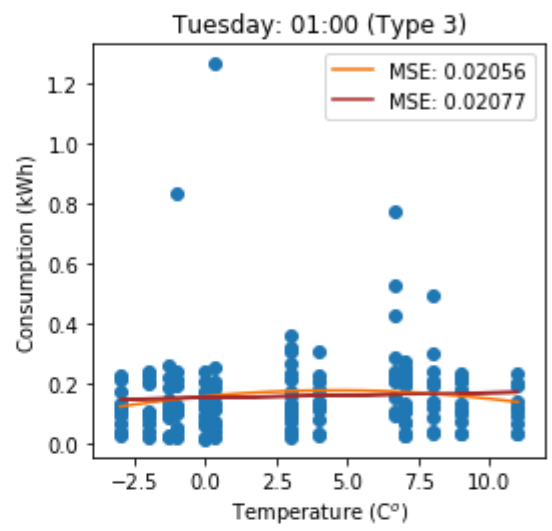

Tuesday: 00:00 (Type 1)
MSE: 0.02338
MSE: 0.02339
Consumption (kWh)
Temperature (C°)

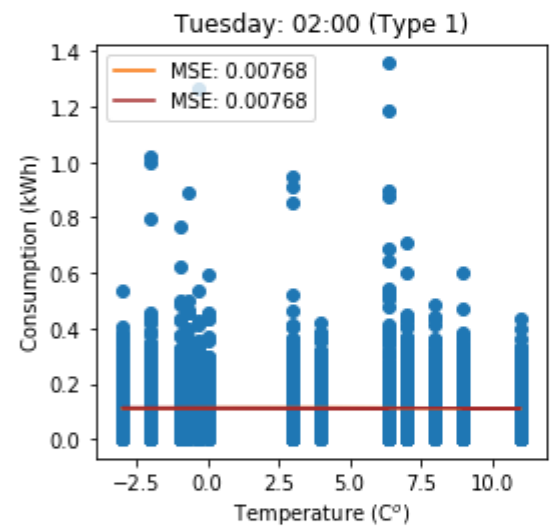

Tuesday: 00:00 (Type 2)
MSE: 0.21967
MSE: 0.21984
Consumption (kWh)
Temperature (C°)

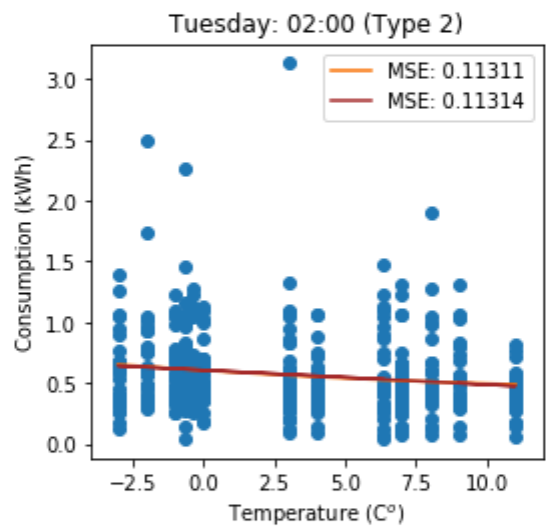

Tuesday: 00:00 (Type 3)
MSE: 0.03546
MSE: 0.03556
Consumption (kWh)
Temperature (C°)

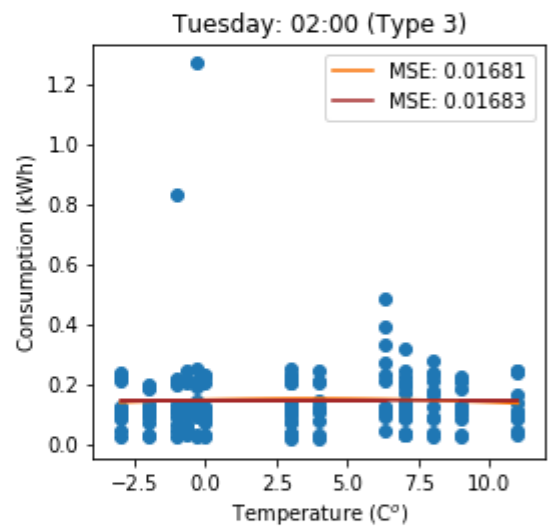

Tuesday: 01:00 (Type 1)
MSE: 0.01268
MSE: 0.01268
Consumption (kWh)
Temperature (C°)

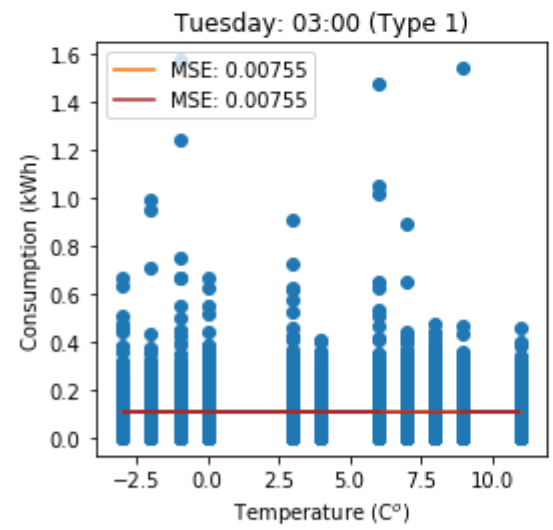

Tuesday: 01:00 (Type 2)
MSE: 0.18607
MSE: 0.1861
Consumption (kWh)
Temperature (C°)

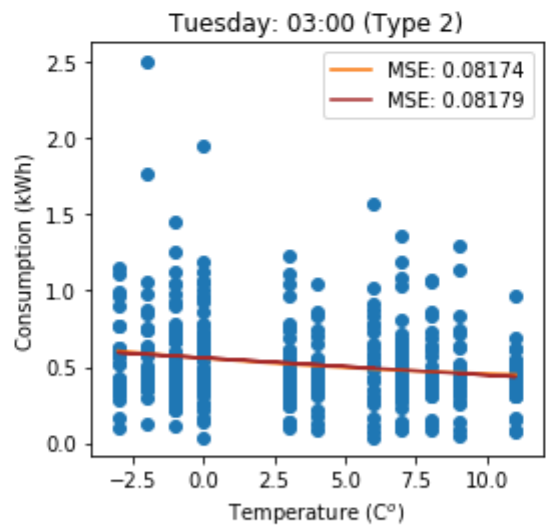

Tuesday: 01:00 (Type 3)
MSE: 0.02056
MSE: 0.02077
Consumption (kWh)
Temperature (C°)

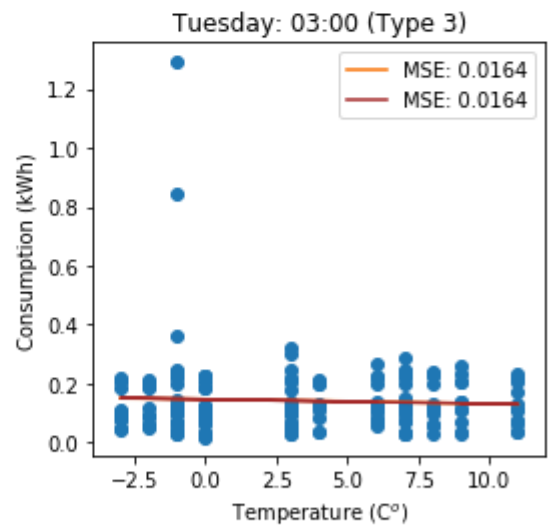

Tuesday: 02:00 (Type 1)
MSE: 0.00768
MSE: 0.00768
Consumption (kWh)
Temperature (C°)

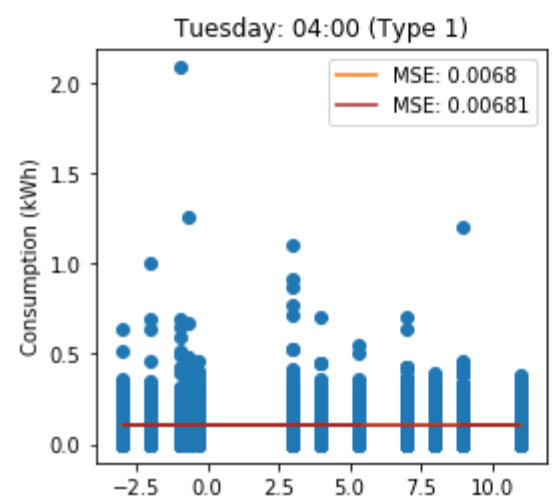

Tuesday: 02:00 (Type 2)
MSE: 0.11311
MSE: 0.11314
Consumption (kWh)
Temperature (C°)

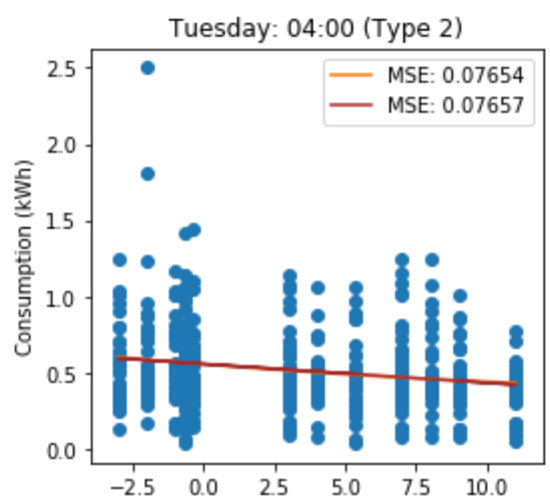

Tuesday: 02:00 (Type 3)
MSE: 0.01681
MSE: 0.01683
Consumption (kWh)
Temperature (C°)

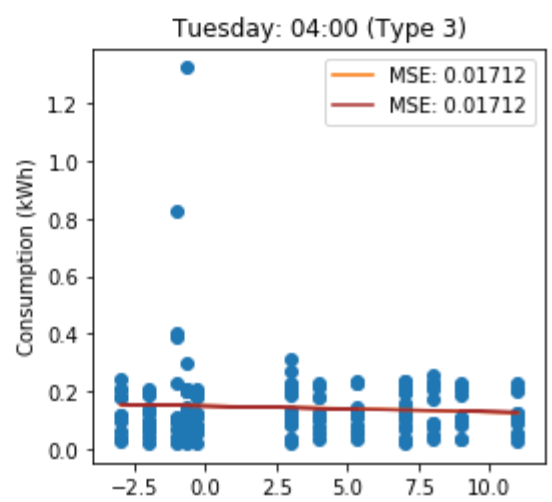

Tuesday: 03:00 (Type 1)
MSE: 0.00755
MSE: 0.00755
Consumption (kWh)
Temperature (C°)

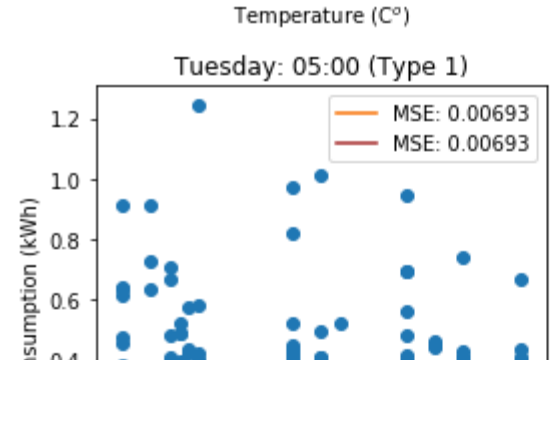

Tuesday: 03:00 (Type 2)
MSE: 0.08174
MSE: 0.08179
Consumption (kWh)
Temperature (C°)

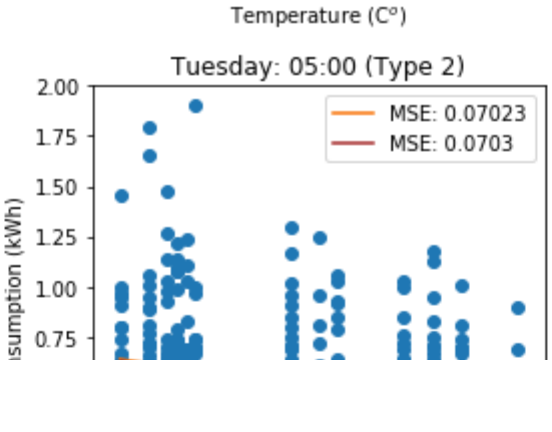

Tuesday: 03:00 (Type 3)
MSE: 0.0164
MSE: 0.0164
Consumption (kWh)
Temperature (C°)

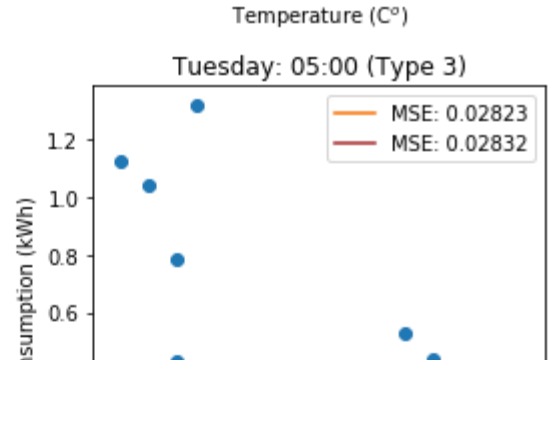

Tuesday: 04:00 (Type 1)
MSE: 0.0068
MSE: 0.00681
Consumption (kWh)
Temperature (C°)

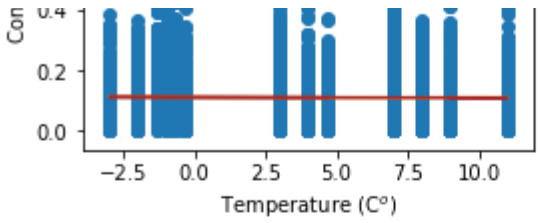

Tuesday: 04:00 (Type 2)
MSE: 0.07654
MSE: 0.07657
Consumption (kWh)
Temperature (C°)

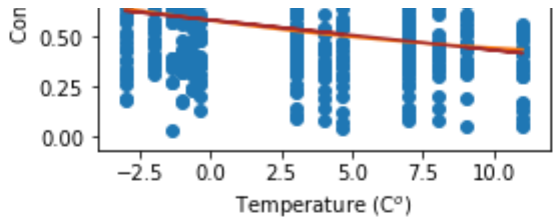

Tuesday: 04:00 (Type 3)
MSE: 0.01712
MSE: 0.01712
Consumption (kWh)
Temperature (C°)

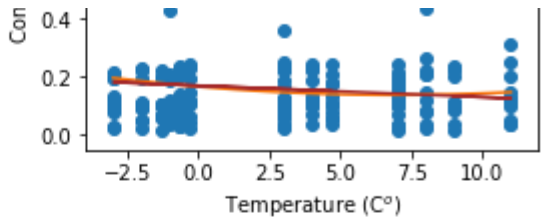

Tuesday: 05:00 (Type 1)
MSE: 0.00693
MSE: 0.00693
Temperature (C°)

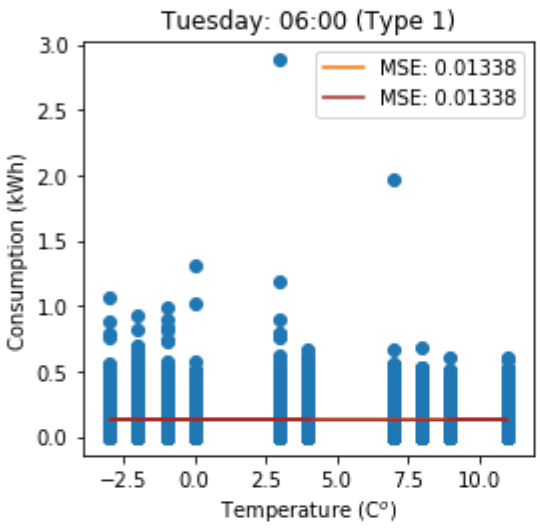

Tuesday: 05:00 (Type 2)
MSE: 0.07023
MSE: 0.0703
Temperature (C°)

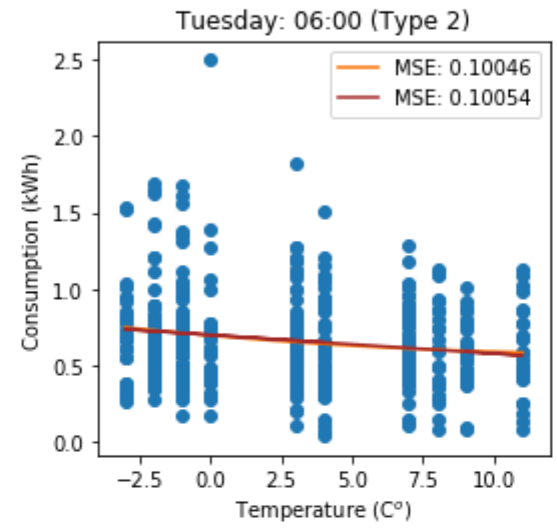

Tuesday: 05:00 (Type 3)
MSE: 0.02823
MSE: 0.02832
Temperature (C°)

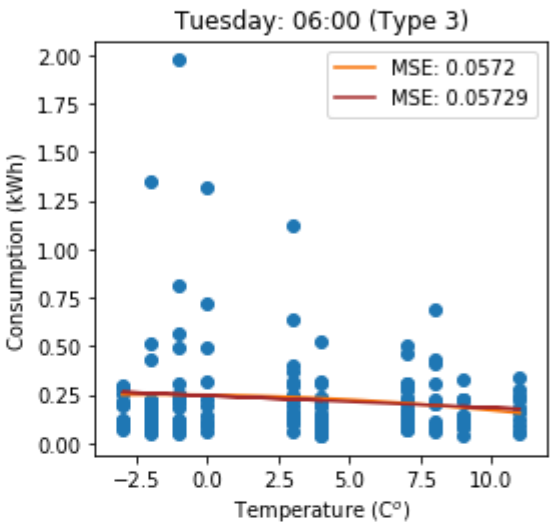

Con
0.4
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

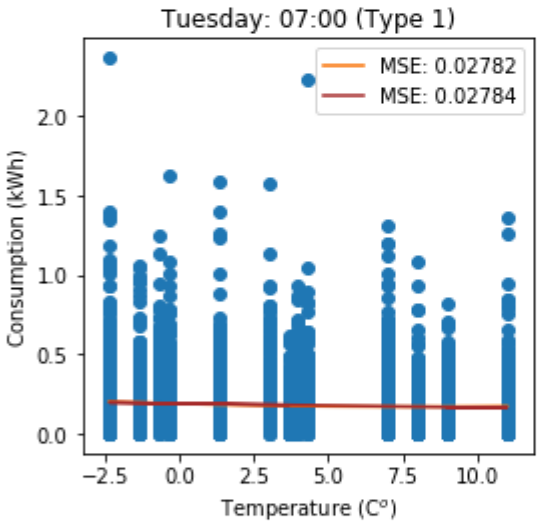

Con
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

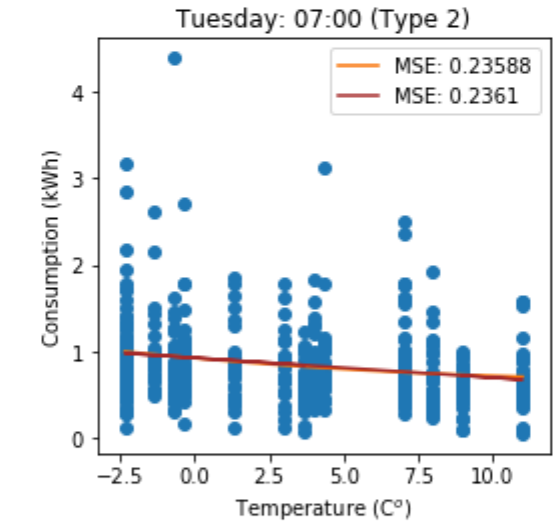

Con
0.4
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

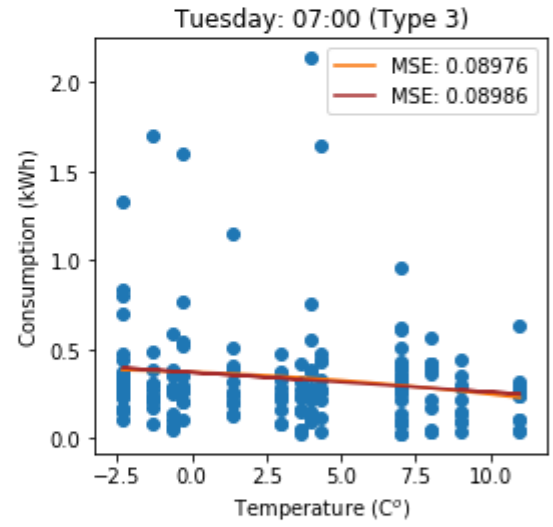

Tuesday: 06:00 (Type 1)
MSE: 0.01338
MSE: 0.01338
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

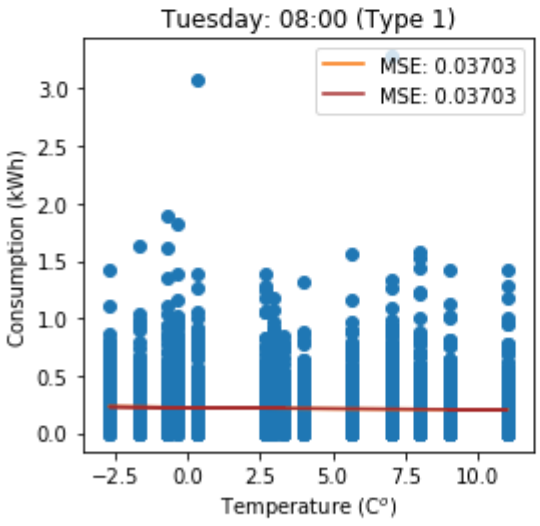

Tuesday: 06:00 (Type 2)
MSE: 0.10046
MSE: 0.10054
Consumption (kWh)
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

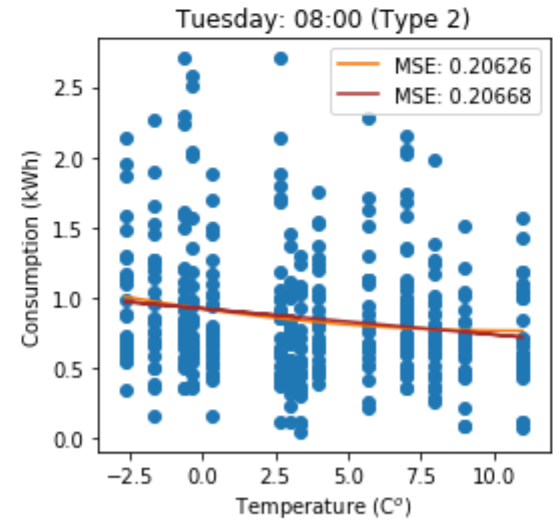

Tuesday: 06:00 (Type 3)
MSE: 0.0572
MSE: 0.05729
Consumption (kWh)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

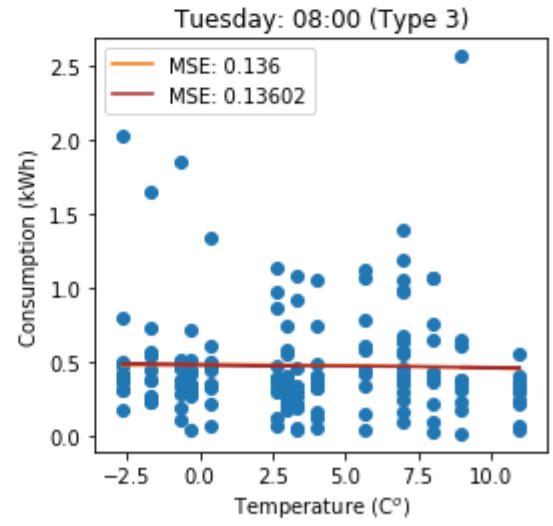

Tuesday: 07:00 (Type 1)
MSE: 0.02782
MSE: 0.02784
Consumption (kWh)
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

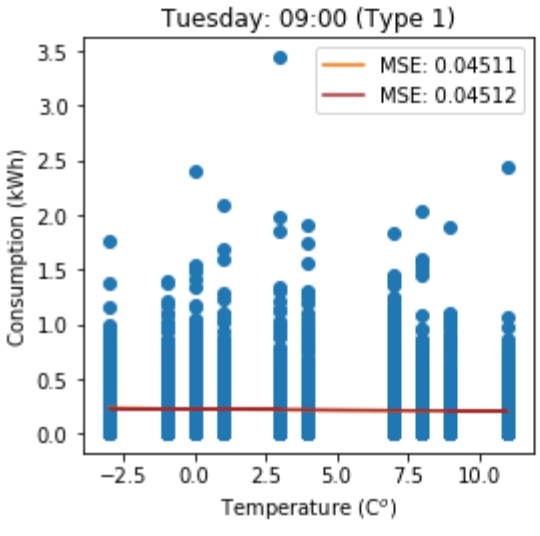

Tuesday: 07:00 (Type 2)
MSE: 0.23588
MSE: 0.2361
Consumption (kWh)
4
3
2
1
0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

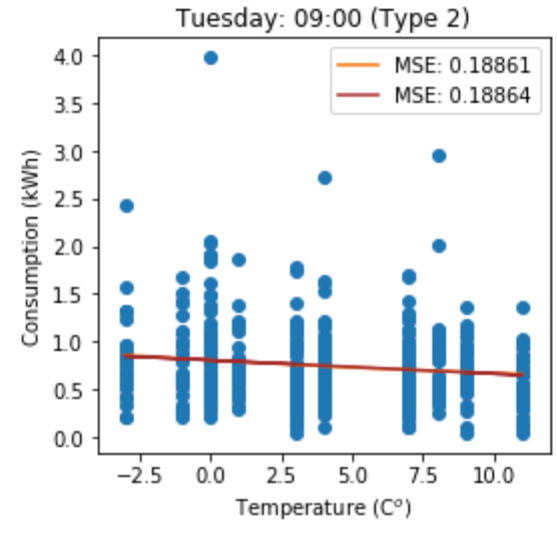

Tuesday: 07:00 (Type 3)
MSE: 0.08976
MSE: 0.08986
Consumption (kWh)
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

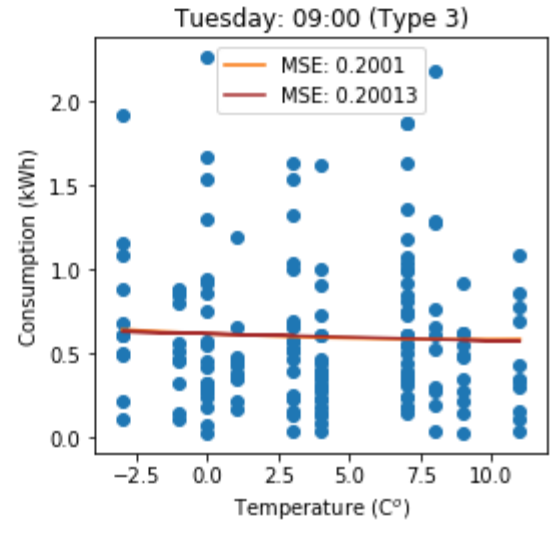

Tuesday: 08:00 (Type 1)
MSE: 0.03703
MSE: 0.03703
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

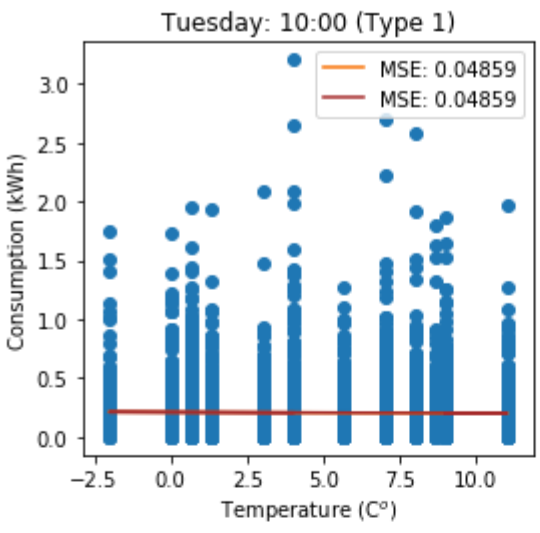

Tuesday: 08:00 (Type 2)
MSE: 0.20626
MSE: 0.20668
Consumption (kWh)
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

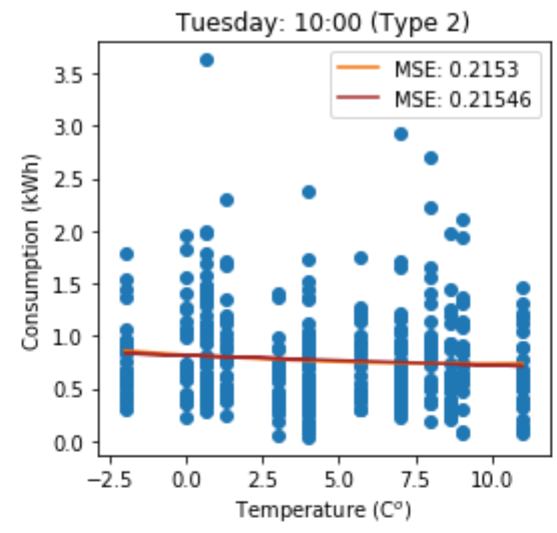

Tuesday: 08:00 (Type 3)
MSE: 0.136
MSE: 0.13602
Consumption (kWh)
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

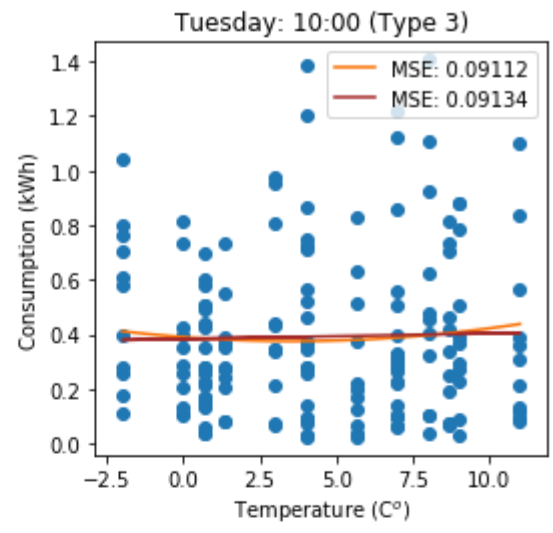

Tuesday: 09:00 (Type 1)
MSE: 0.04511
MSE: 0.04512
Consumption (kWh)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)


Tuesday: 09:00 (Type 2)
MSE: 0.18861
MSE: 0.18864
Consumption (kWh)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)


Tuesday: 09:00 (Type 3)
MSE: 0.2001
MSE: 0.20013
Consumption (kWh)
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)


Tuesday: 10:00 (Type 1)
MSE: 0.04859
MSE: 0.04859
Consumption (kWh)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

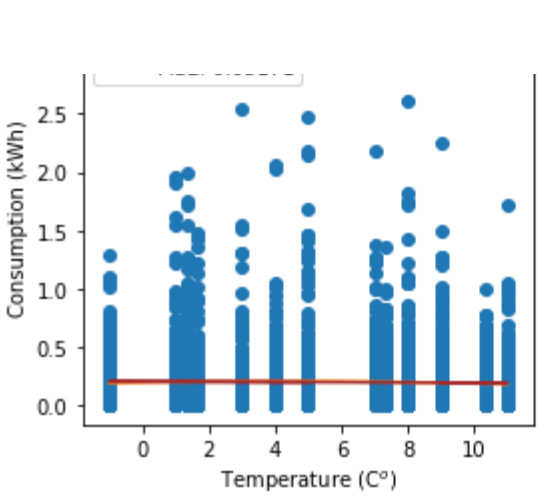

Tuesday: 10:00 (Type 2)
MSE: 0.2153
MSE: 0.21546
Consumption (kWh)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

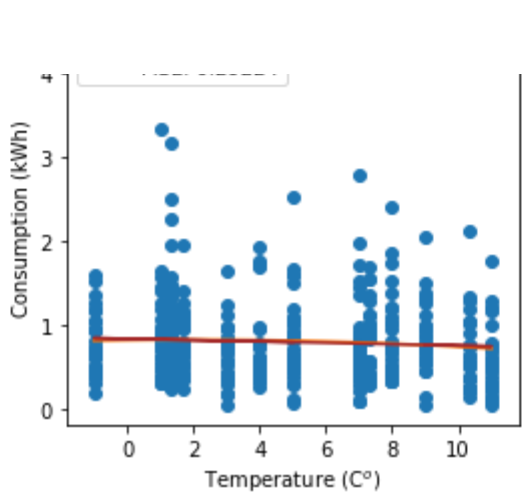

Tuesday: 10:00 (Type 3)
MSE: 0.09112
MSE: 0.09134
Consumption (kWh)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

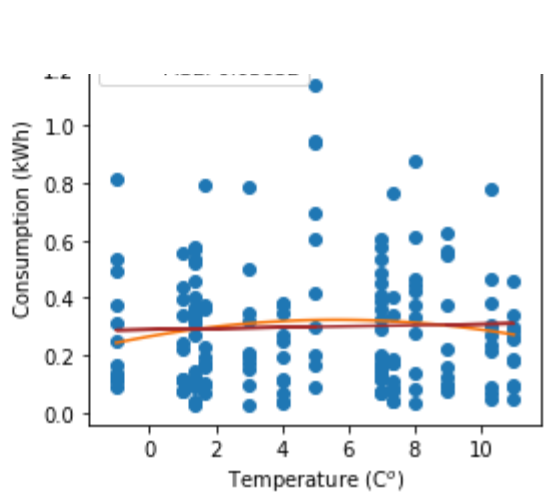
Tuesday: 11:00 (Type 1)
MSE: 0.05166
MSE: 0.05171
3.0

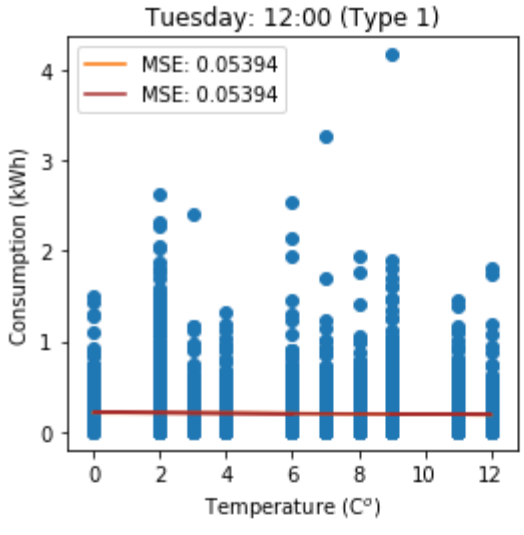
Tuesday: 11:00 (Type 2)
MSE: 0.29201
MSE: 0.29224

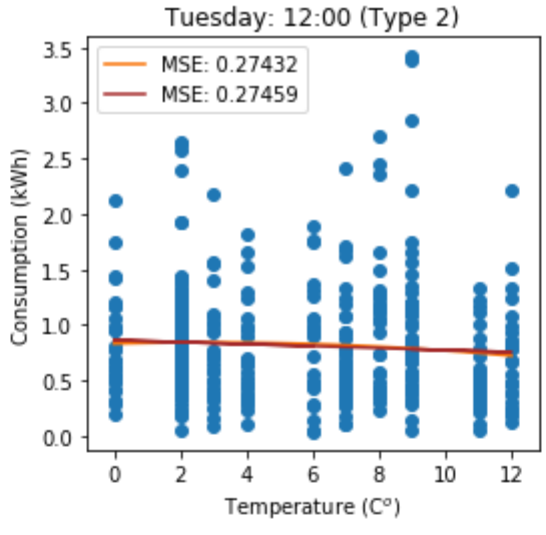
Tuesday: 11:00 (Type 3)
MSE: 0.0579
MSE: 0.05832
1.4
1.2

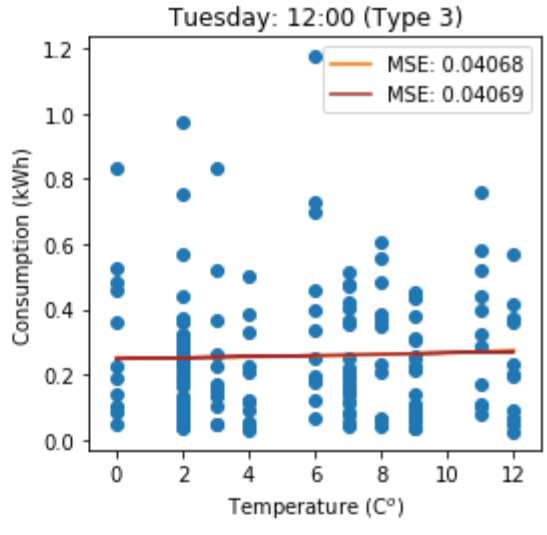

Consumption (kWh)
Temperature (C°)

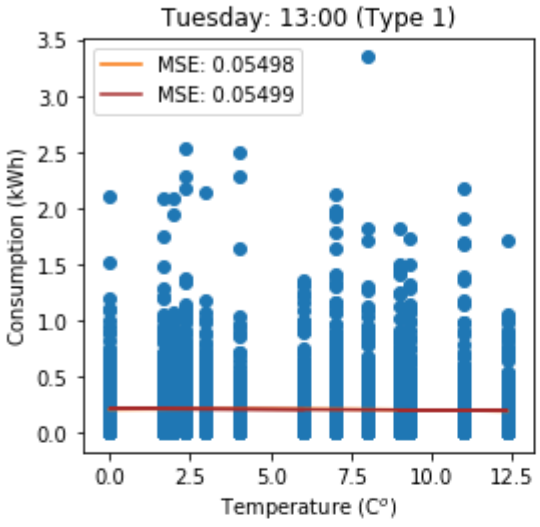

Consumption (kWh)
Temperature (C°)

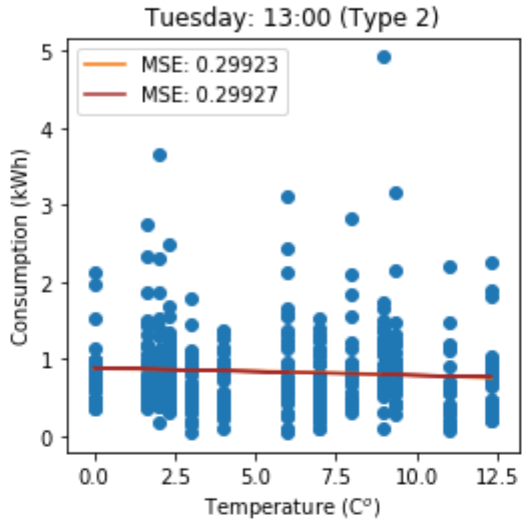

Consumption (kWh)
Temperature (C°)

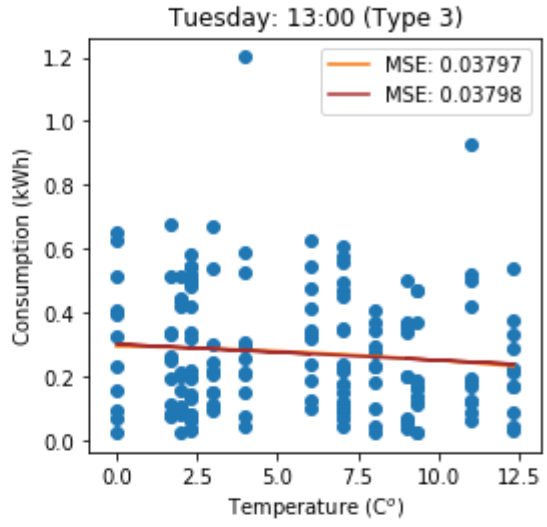

Tuesday: 12:00 (Type 1)
MSE: 0.05394
MSE: 0.05394
Consumption (kWh)
Temperature (C°)

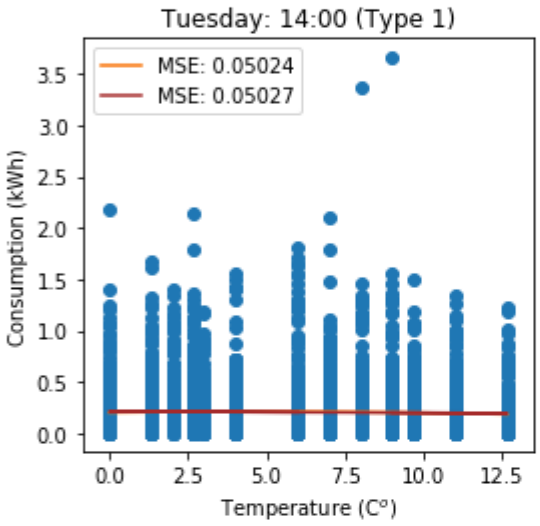

Tuesday: 12:00 (Type 2)
MSE: 0.27432
MSE: 0.27459
Consumption (kWh)
Temperature (C°)

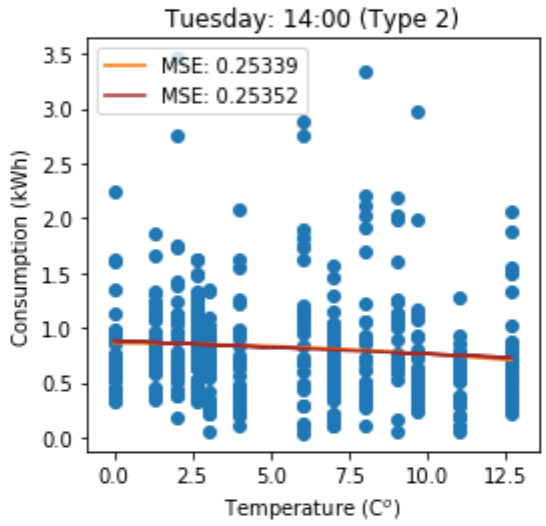

Tuesday: 12:00 (Type 3)
MSE: 0.04068
MSE: 0.04069
Consumption (kWh)
Temperature (C°)

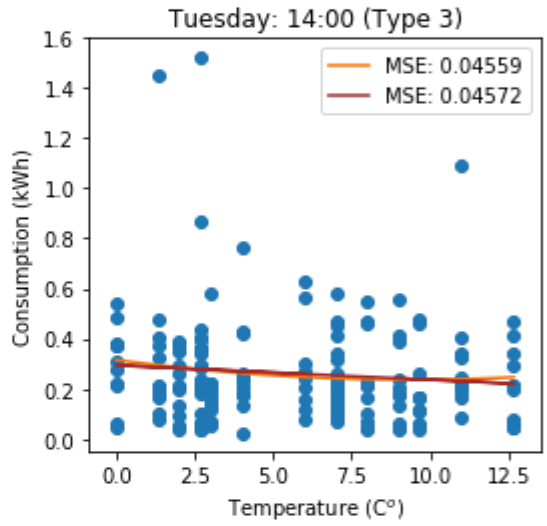

Tuesday: 13:00 (Type 1)
MSE: 0.05498
MSE: 0.05499
Consumption (kWh)
Temperature (C°)

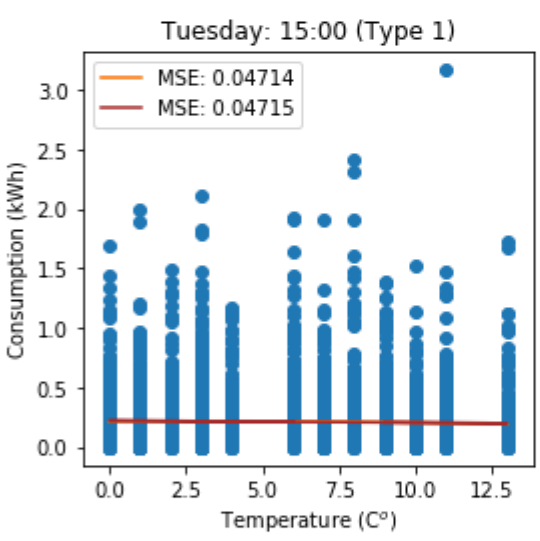

Tuesday: 13:00 (Type 2)
MSE: 0.29923
MSE: 0.29927
Consumption (kWh)
Temperature (C°)

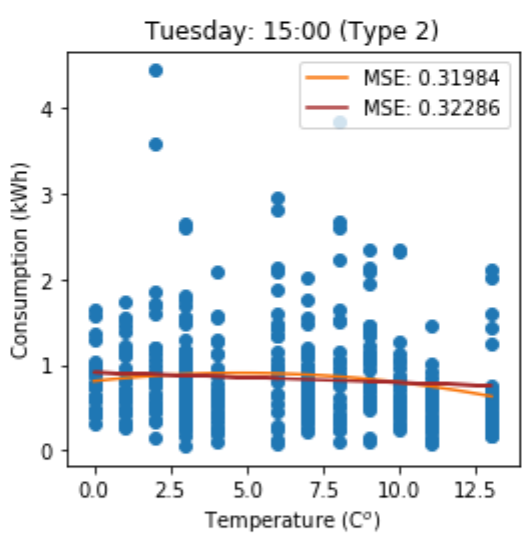

Tuesday: 13:00 (Type 3)
MSE: 0.03797
MSE: 0.03798
Consumption (kWh)
Temperature (C°)

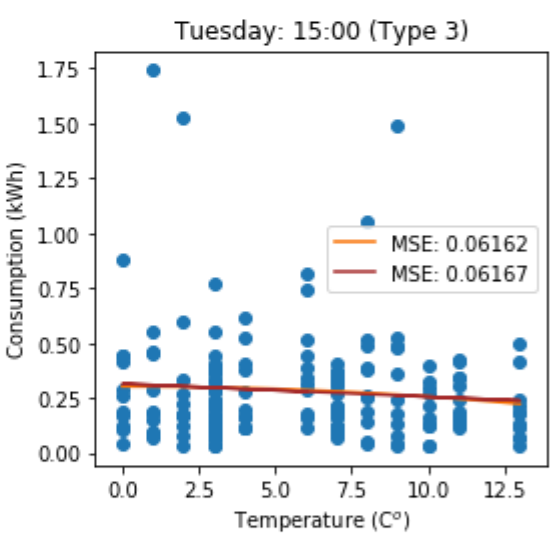

Tuesday: 14:00 (Type 1)
MSE: 0.05024
MSE: 0.05027
Consumption (kWh)
Temperature (C°)

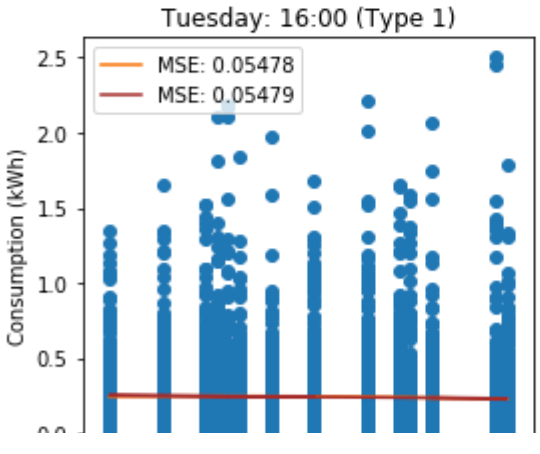

Tuesday: 14:00 (Type 2)
MSE: 0.25339
MSE: 0.25352
Consumption (kWh)
Temperature (C°)

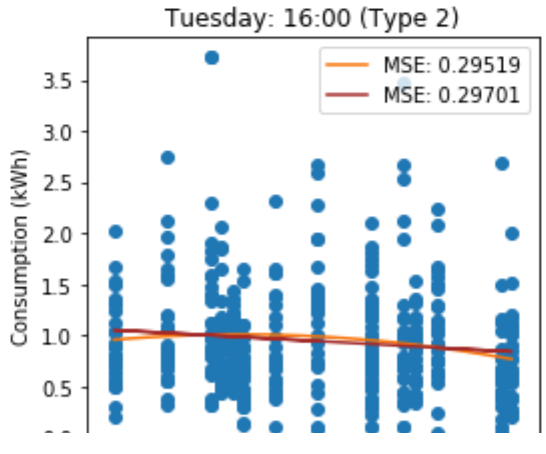

Tuesday: 14:00 (Type 3)
MSE: 0.04559
MSE: 0.04572
Consumption (kWh)
Temperature (C°)

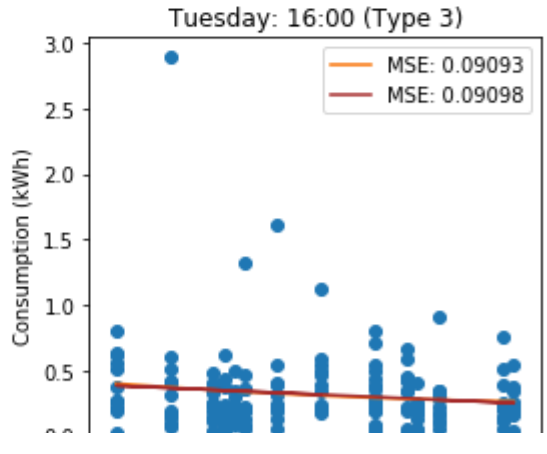

Tuesday: 15:00 (Type 1)
MSE: 0.04714
MSE: 0.04715
Consumption (kWh)
Temperature (C°)

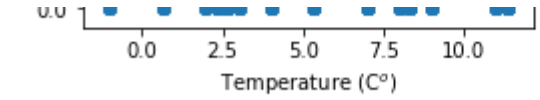

Tuesday: 15:00 (Type 2)
MSE: 0.31984
MSE: 0.32286
Consumption (kWh)
Temperature (C°)

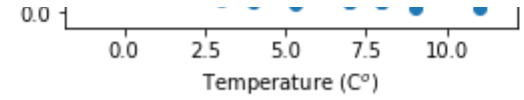

Tuesday: 15:00 (Type 3)
MSE: 0.06162
MSE: 0.06167
Consumption (kWh)
Temperature (C°)

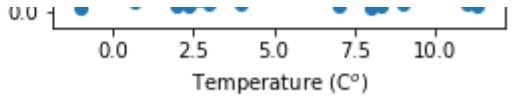

Tuesday: 16:00 (Type 1)
MSE: 0.05478
MSE: 0.05479
Consumption (kWh)

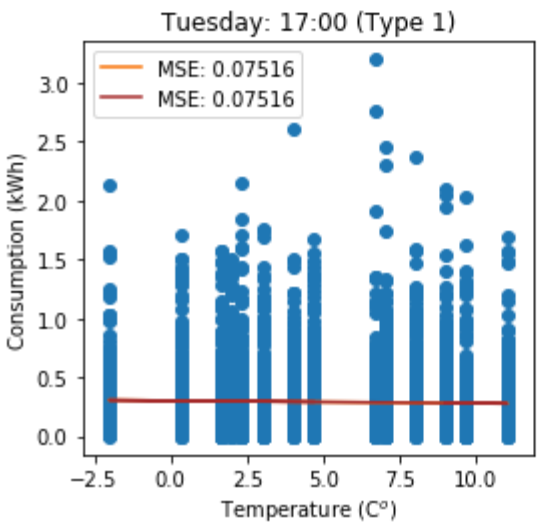

Tuesday: 16:00 (Type 2)
MSE: 0.29519
MSE: 0.29701
Consumption (kWh)

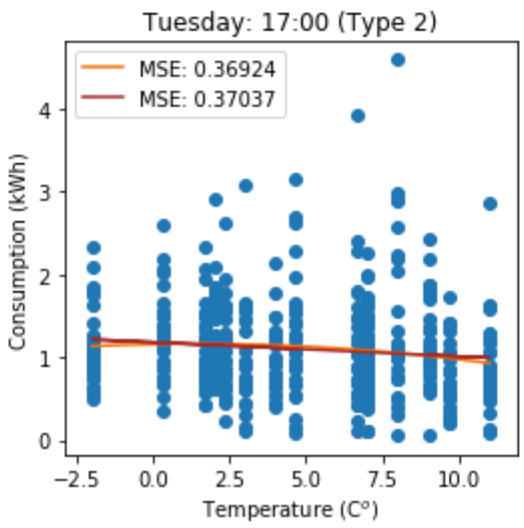

Tuesday: 16:00 (Type 3)
MSE: 0.09093
MSE: 0.09098
Consumption (kWh)

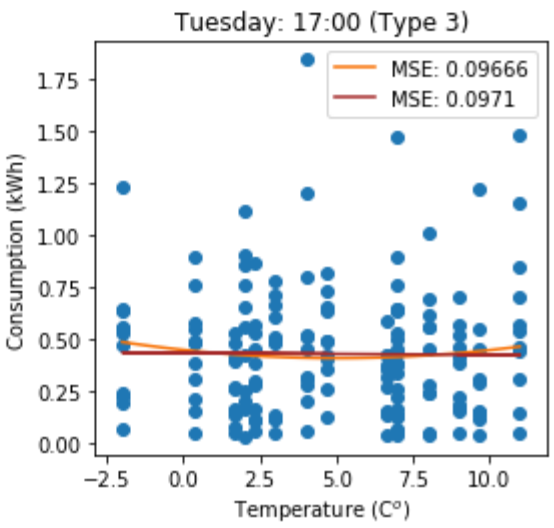

Temperature (C°)

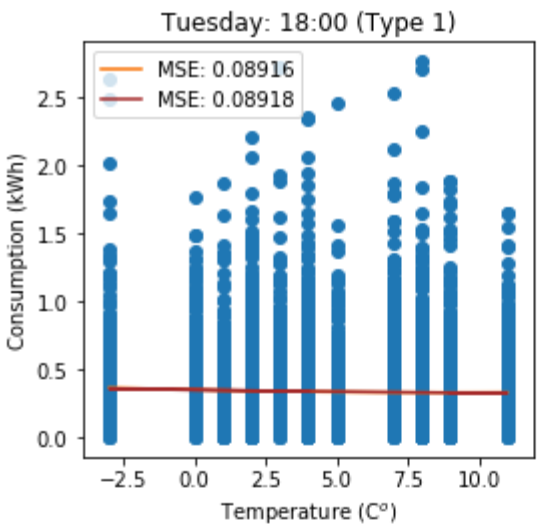

Temperature (C°)

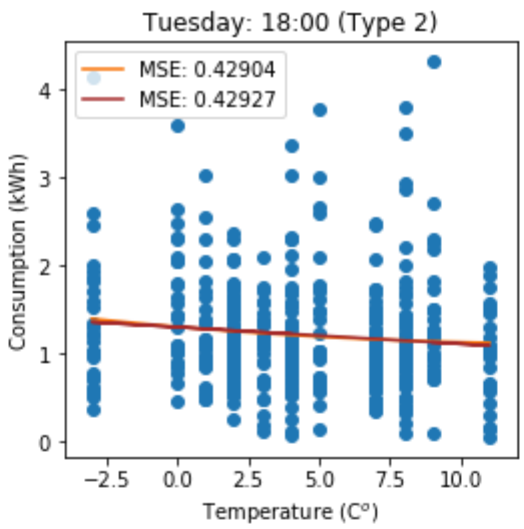

Temperature (C°)

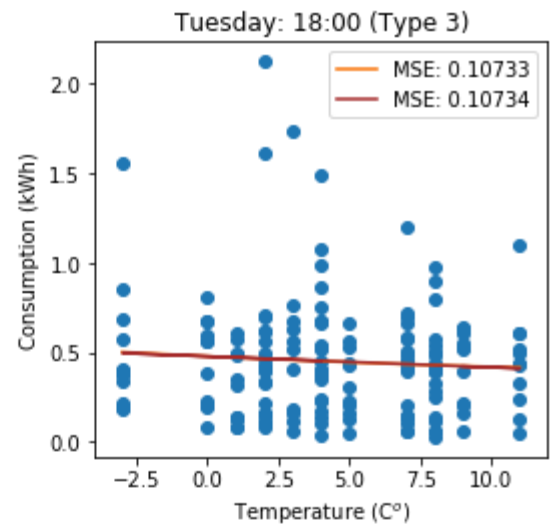

Tuesday: 17:00 (Type 1)
MSE: 0.07516
MSE: 0.07516
Consumption (kWh)
Temperature (C°)

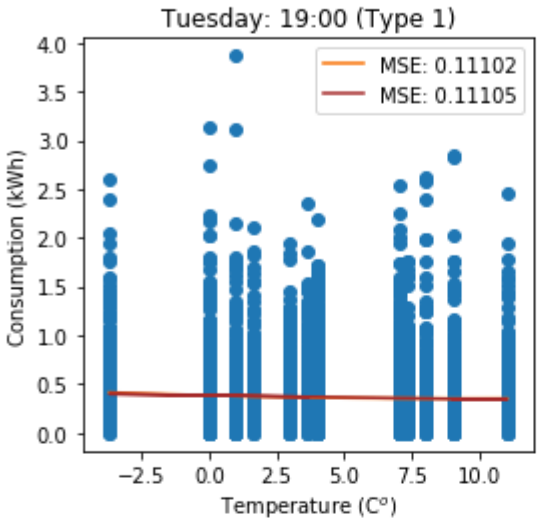

Tuesday: 17:00 (Type 2)
MSE: 0.36924
MSE: 0.37037
Consumption (kWh)
Temperature (C°)

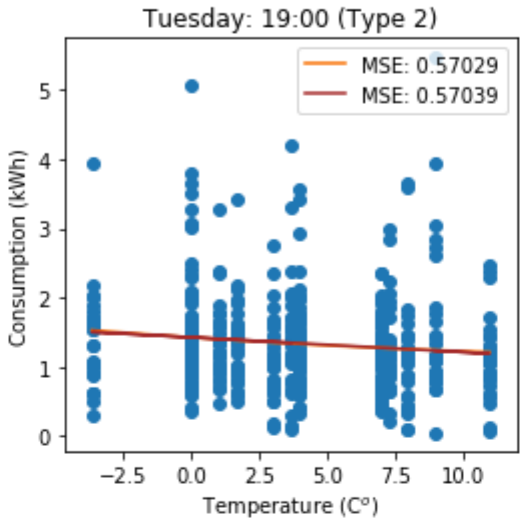

Tuesday: 17:00 (Type 3)
MSE: 0.09666
MSE: 0.0971
Consumption (kWh)
Temperature (C°)

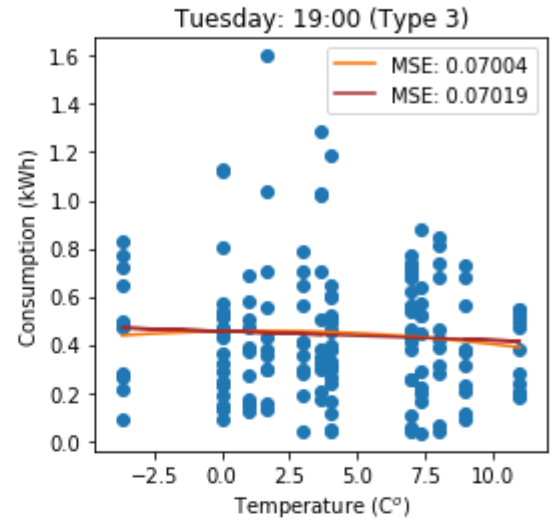

Tuesday: 18:00 (Type 1)
MSE: 0.08916
MSE: 0.08918
Consumption (kWh)
Temperature (C°)

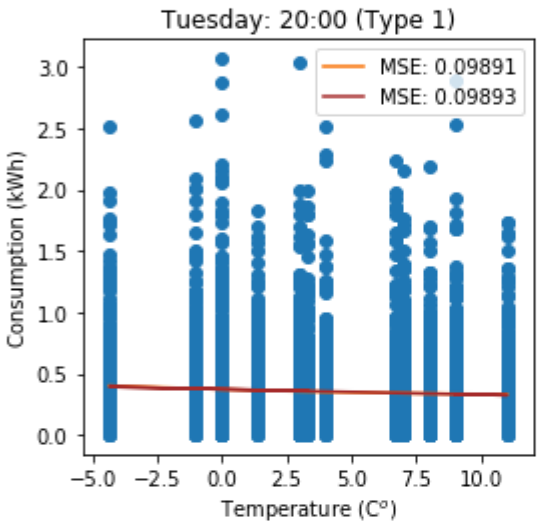

Tuesday: 18:00 (Type 2)
MSE: 0.42904
MSE: 0.42927
Consumption (kWh)
Temperature (C°)

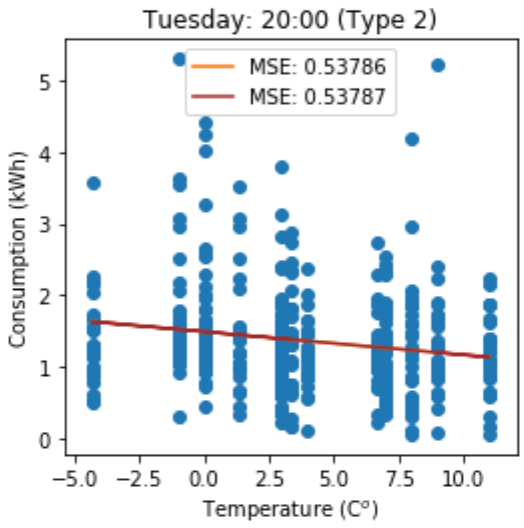

Tuesday: 18:00 (Type 3)
MSE: 0.10733
MSE: 0.10734
Consumption (kWh)
Temperature (C°)

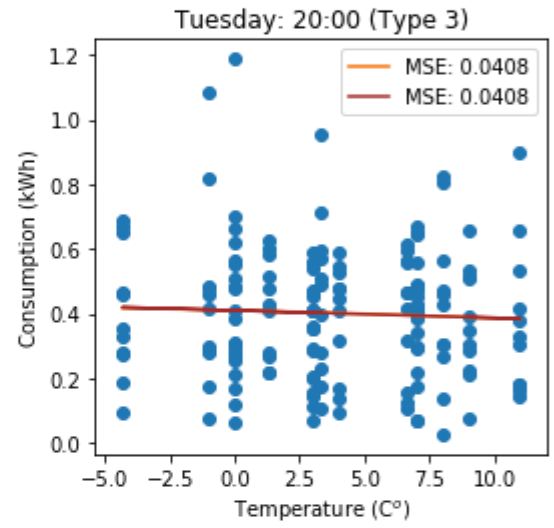

Tuesday: 19:00 (Type 1)
MSE: 0.11102
MSE: 0.11105
Consumption (kWh)
Temperature (C°)

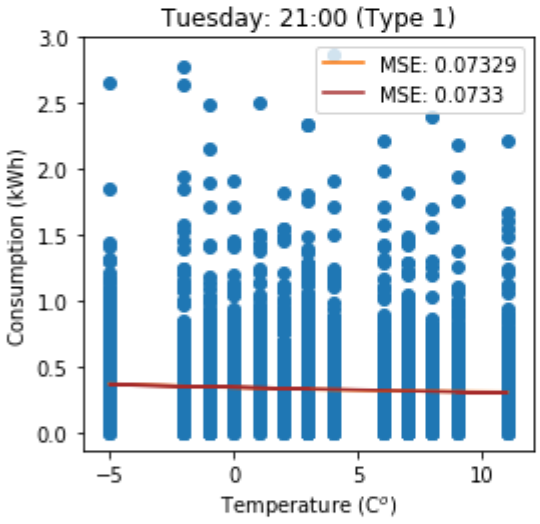

Tuesday: 19:00 (Type 2)
MSE: 0.57029
MSE: 0.57039
Consumption (kWh)
Temperature (C°)

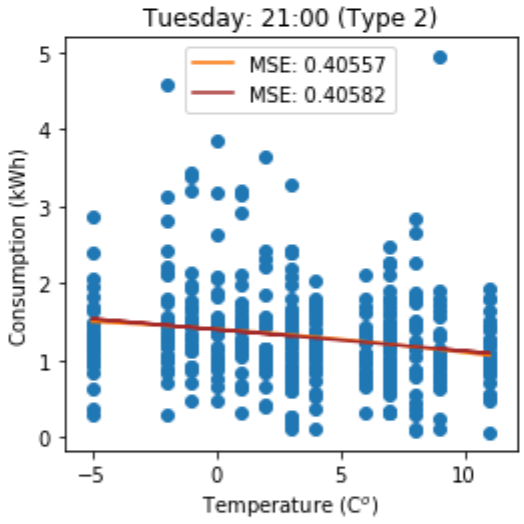

Tuesday: 19:00 (Type 3)
MSE: 0.07004
MSE: 0.07019
Consumption (kWh)
Temperature (C°)

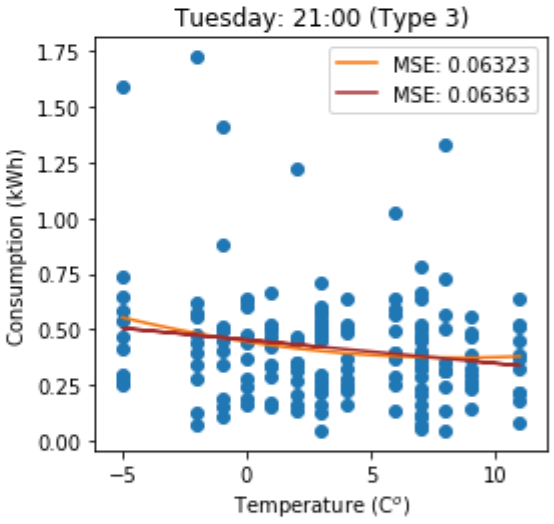

Tuesday: 20:00 (Type 1)
MSE: 0.09891
MSE: 0.09893
Consumption (kWh)
Temperature (C°)

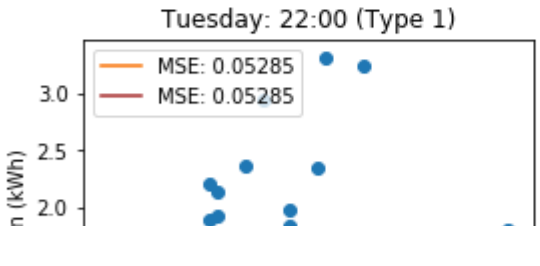

Tuesday: 20:00 (Type 2)
MSE: 0.53786
MSE: 0.53787
Consumption (kWh)
Temperature (C°)

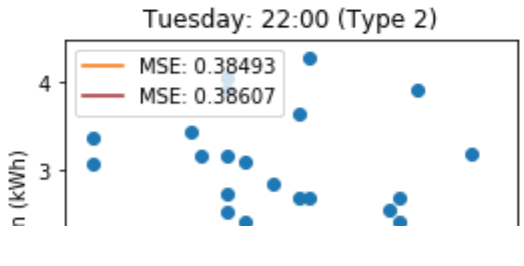

Tuesday: 20:00 (Type 3)
MSE: 0.0408
MSE: 0.0408
Consumption (kWh)
Temperature (C°)

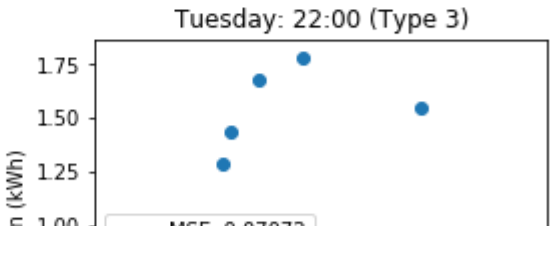

Tuesday: 21:00 (Type 1)
MSE: 0.07329
MSE: 0.0733
Consumption (kWh)
Temperature (C°)

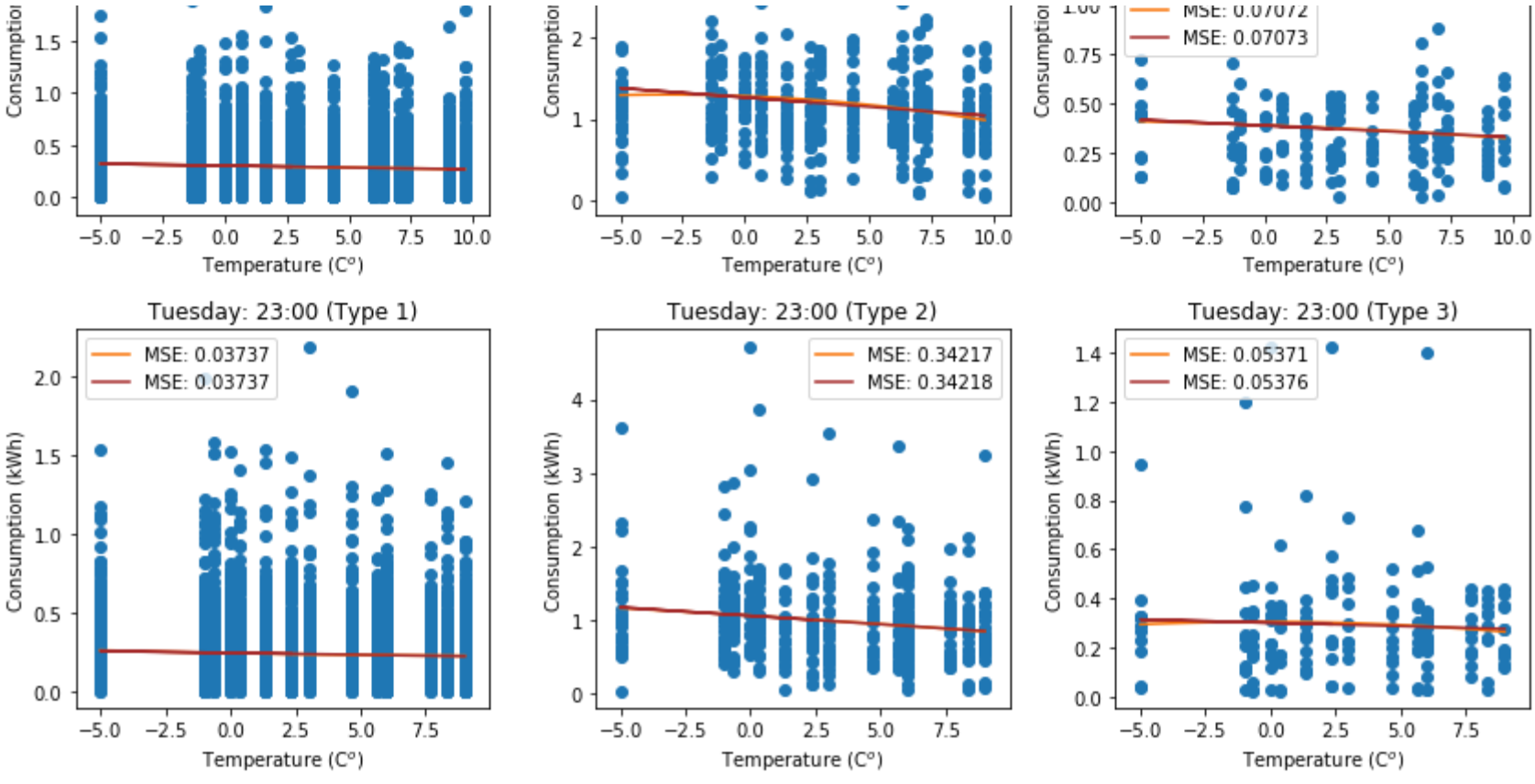

Tuesday: 21:00 (Type 2)
MSE: 0.40557
MSE: 0.40582
Consumption (kWh)
Temperature (C°)


Tuesday: 21:00 (Type 3)
MSE: 0.06323
MSE: 0.06363
Consumption (kWh)
Temperature (C°)

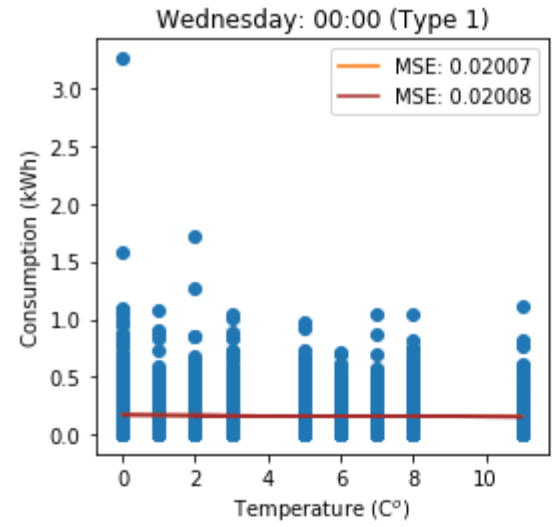

Tuesday: 22:00 (Type 1)
MSE: 0.05285
MSE: 0.05285

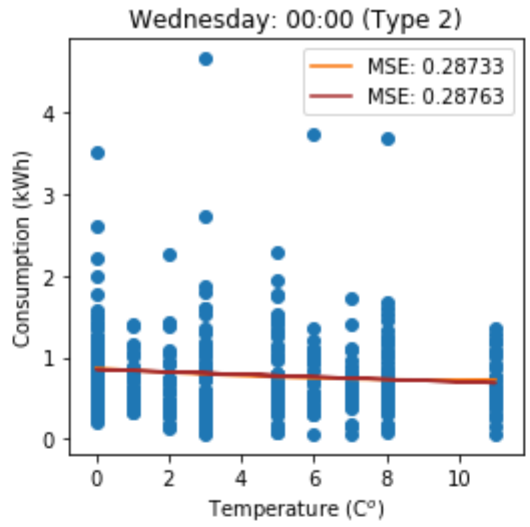

Tuesday: 22:00 (Type 2)
MSE: 0.38493
MSE: 0.38607

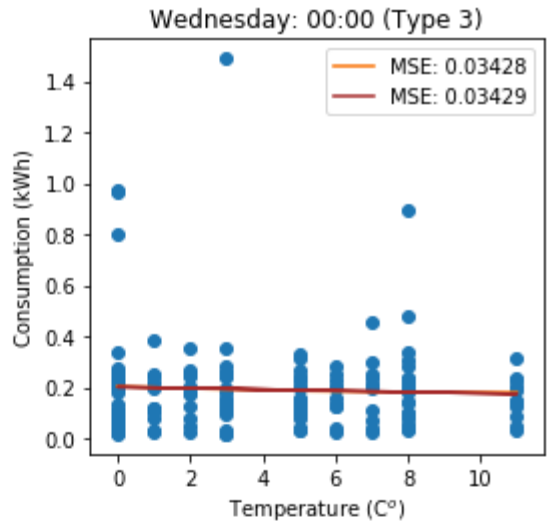

Tuesday: 22:00 (Type 3)

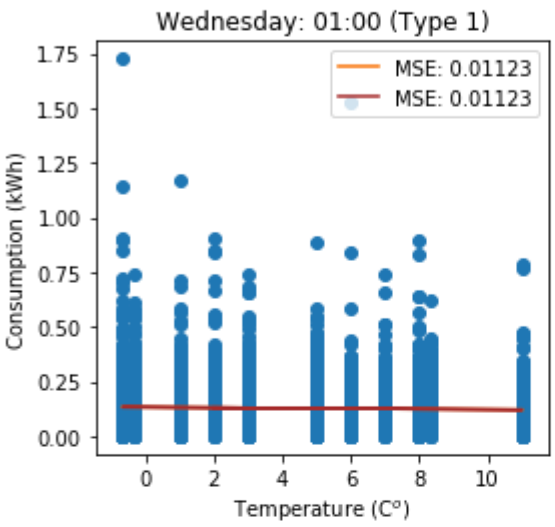

Tuesday: 23:00 (Type 1)
MSE: 0.03737
MSE: 0.03737
Tuesday: 23:00 (Type 2)
MSE: 0.34217
MSE: 0.34218
Tuesday: 23:00 (Type 3)
MSE: 0.05371
MSE: 0.05376
MSE: 0.07072
MSE: 0.07073
Consumption (kWh)
Temperature (C°)

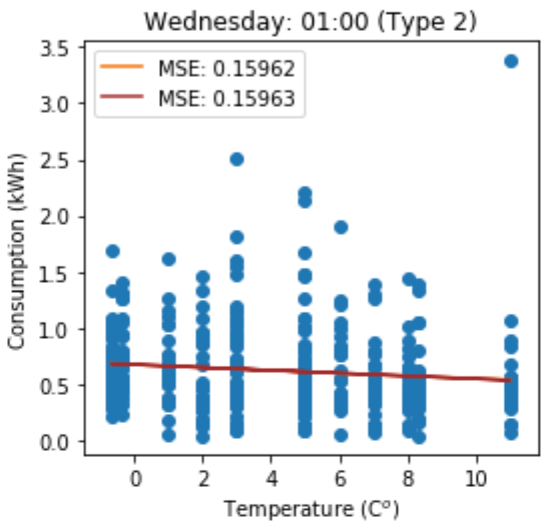
In [10]:
```
# Wednesday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 2 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

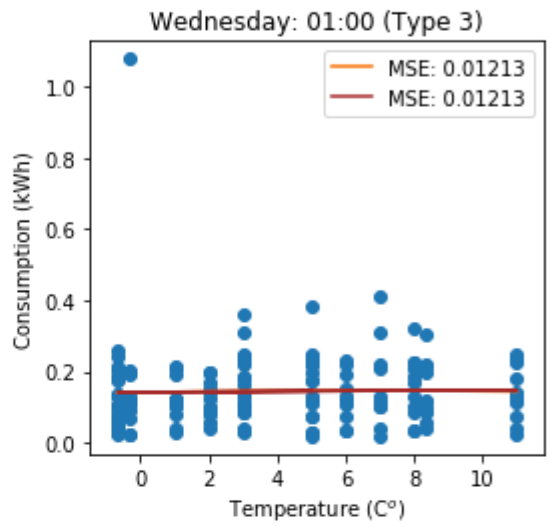

Wednesday: 00:00 (Type 1)
MSE: 0.02007
MSE: 0.02008
Consumption (kWh)
Temperature (C°)

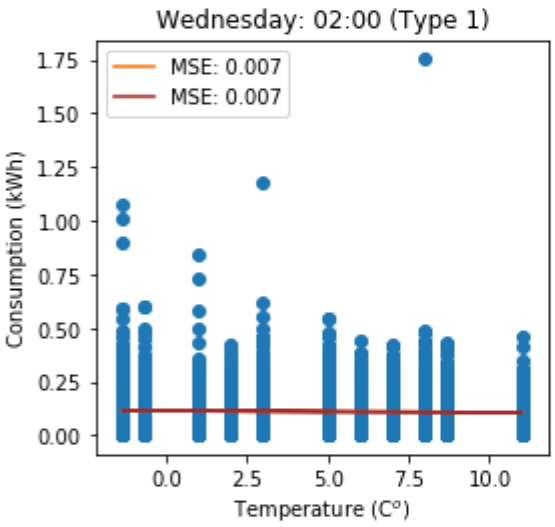

Wednesday: 00:00 (Type 2)
MSE: 0.28733
MSE: 0.28763
Consumption (kWh)
Temperature (C°)

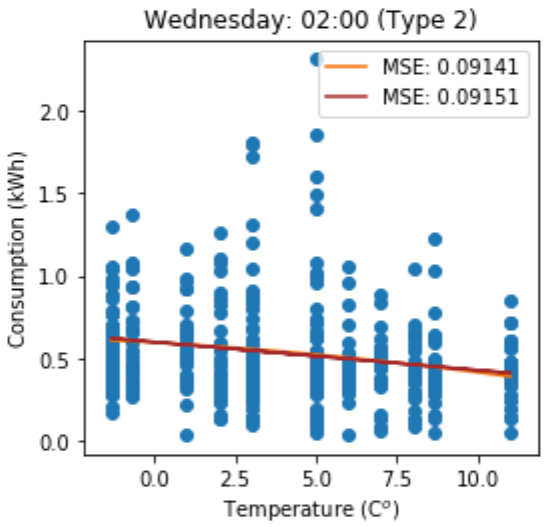

Wednesday: 00:00 (Type 3)
MSE: 0.03428
MSE: 0.03429
Consumption (kWh)
Temperature (C°)

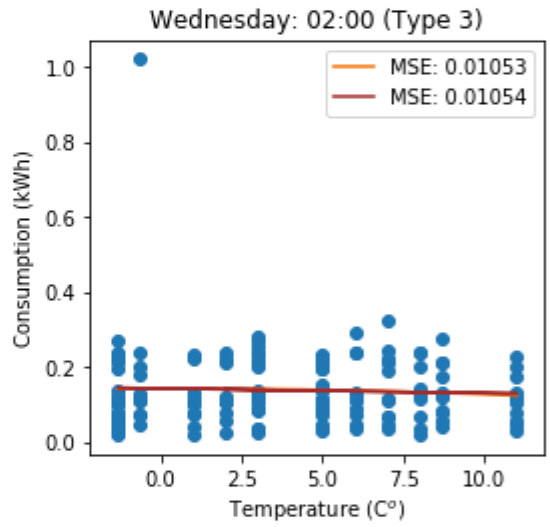

Wednesday: 01:00 (Type 1)
MSE: 0.01123
MSE: 0.01123
Consumption (kWh)
Temperature (C°)

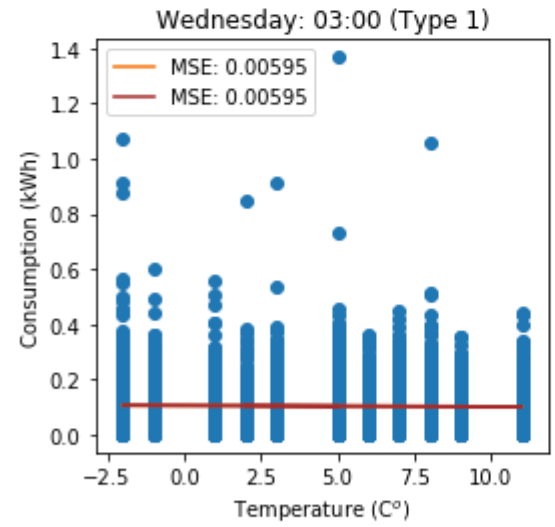

Wednesday: 01:00 (Type 2)
MSE: 0.15962
MSE: 0.15963
Consumption (kWh)
Temperature (C°)

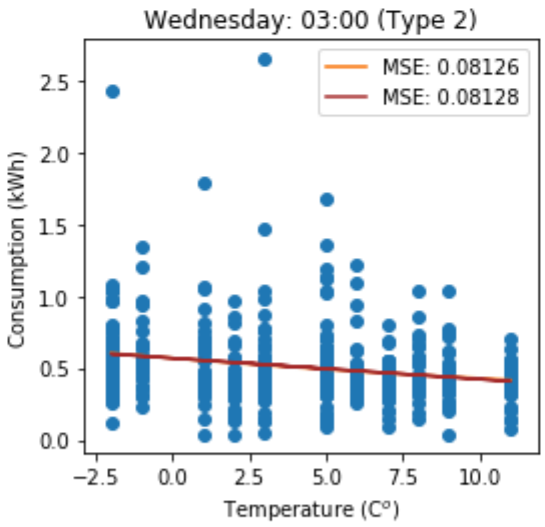

Wednesday: 01:00 (Type 3)
MSE: 0.01213
MSE: 0.01213
Consumption (kWh)
Temperature (C°)

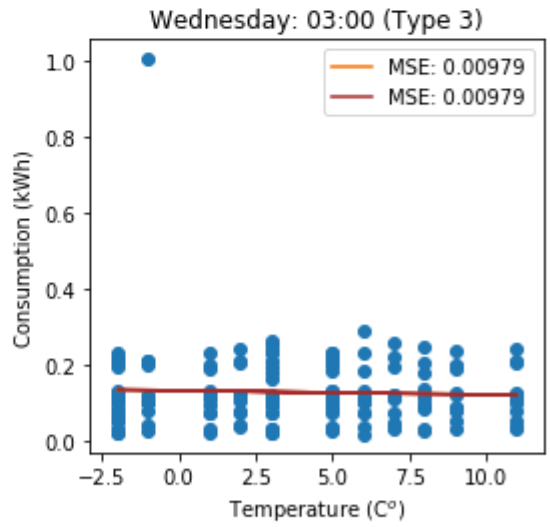

Wednesday: 02:00 (Type 1)
MSE: 0.007
MSE: 0.007
Consumption (kWh)
Temperature (C°)

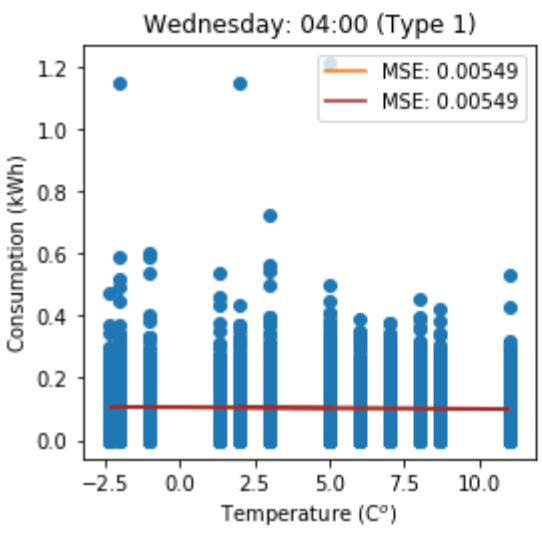

Wednesday: 02:00 (Type 2)
MSE: 0.09141
MSE: 0.09151
Consumption (kWh)
Temperature (C°)

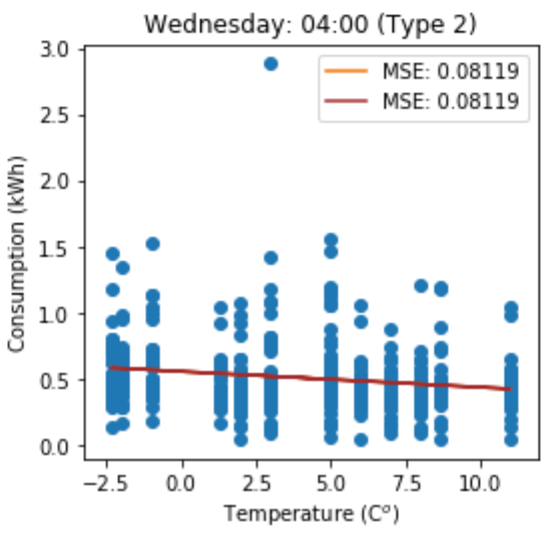

Wednesday: 02:00 (Type 3)
MSE: 0.01053
MSE: 0.01054
Consumption (kWh)
Temperature (C°)

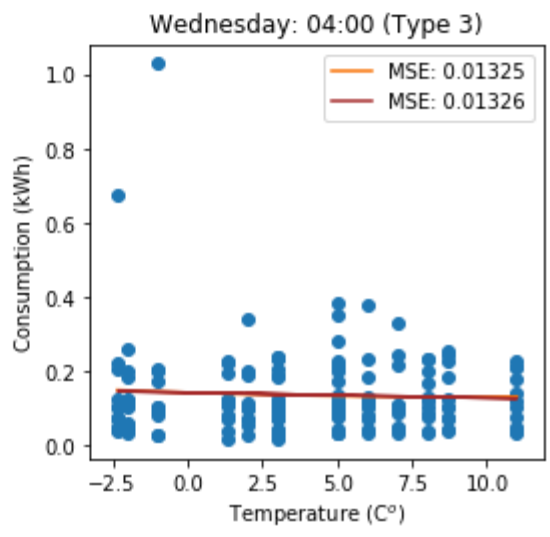

Wednesday: 03:00 (Type 1)
MSE: 0.00595
MSE: 0.00595
Consumption (kWh)
Temperature (C°)

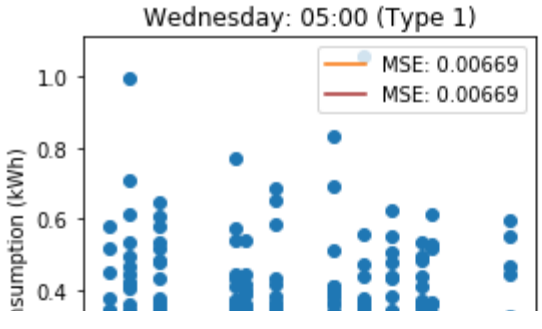

Wednesday: 03:00 (Type 2)
MSE: 0.08126
MSE: 0.08128
Consumption (kWh)
Temperature (C°)

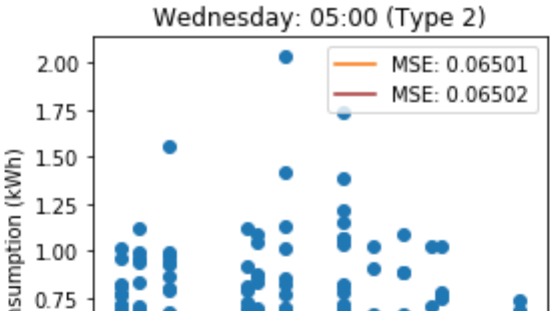

Wednesday: 03:00 (Type 3)
MSE: 0.00979
MSE: 0.00979
Consumption (kWh)
Temperature (C°)

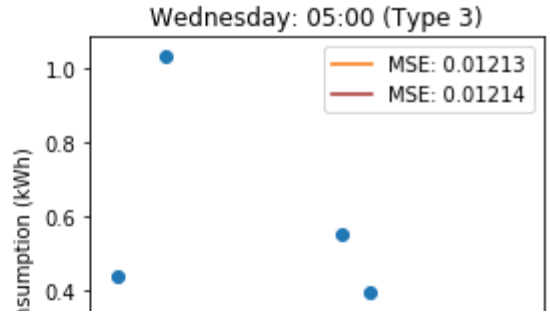

Wednesday: 04:00 (Type 1)
MSE: 0.00549
MSE: 0.00549
Consumption (kWh)
Temperature (C°)

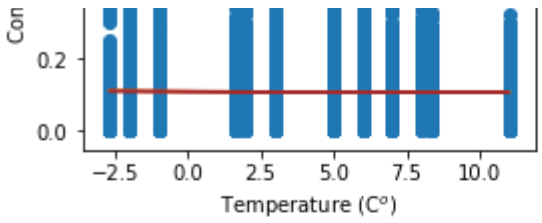

Wednesday: 04:00 (Type 2)
MSE: 0.08119
MSE: 0.08119
Consumption (kWh)
Temperature (C°)

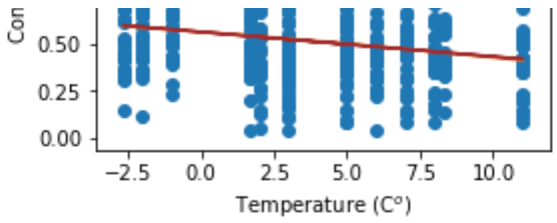

Wednesday: 04:00 (Type 3)
MSE: 0.01325
MSE: 0.01326
Consumption (kWh)
Temperature (C°)

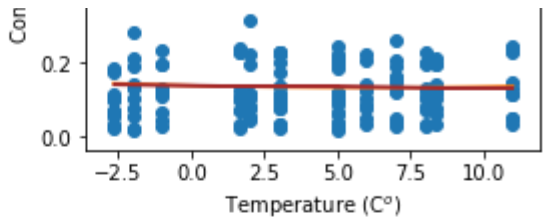

Wednesday: 05:00 (Type 1)
MSE: 0.00669
MSE: 0.00669
Temperature (C°)

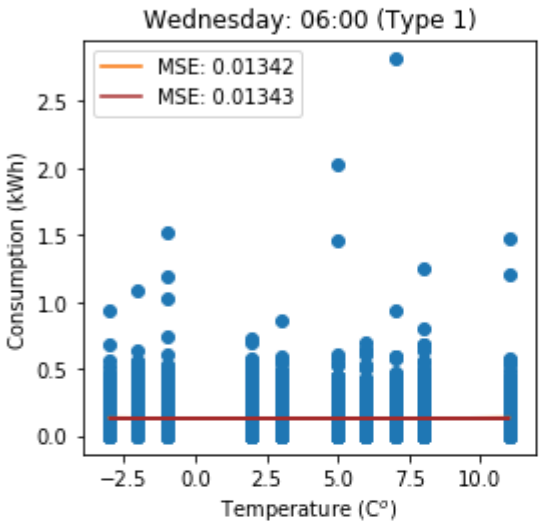

Wednesday: 05:00 (Type 2)
MSE: 0.06501
MSE: 0.06502
Temperature (C°)

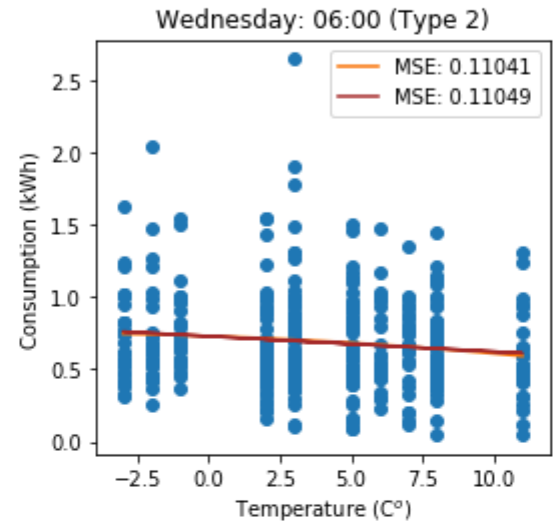

Wednesday: 05:00 (Type 3)
MSE: 0.01213
MSE: 0.01214
Temperature (C°)

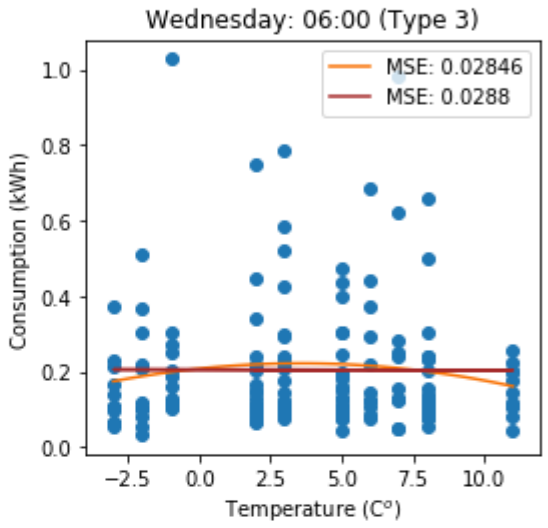

Con
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

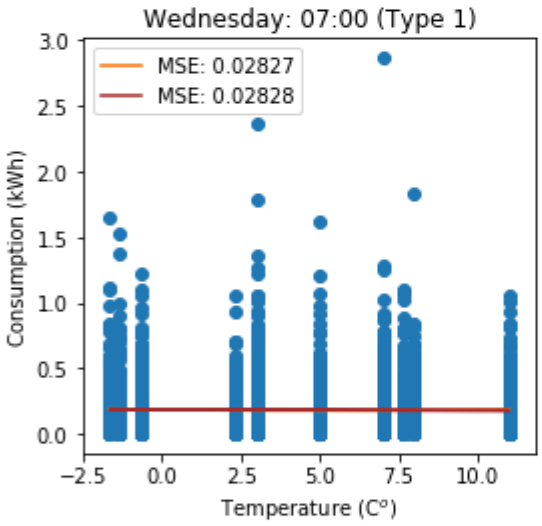

Con
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

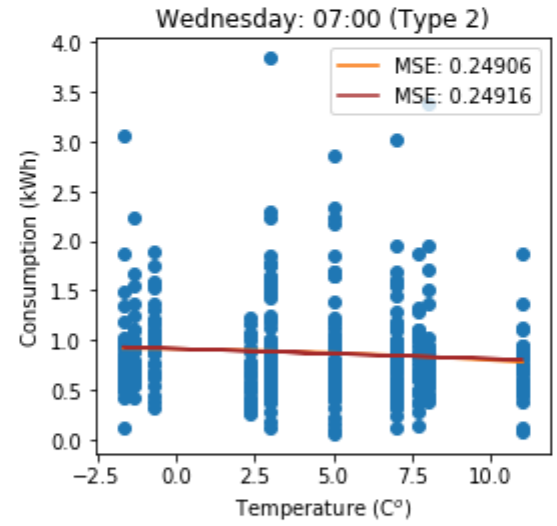

Con
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
Temperature (C°)

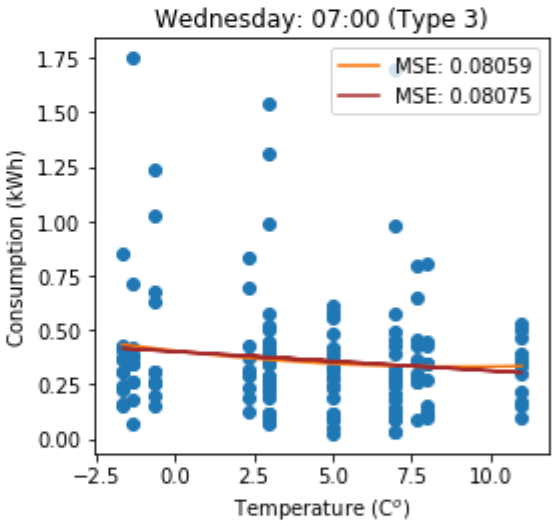

Wednesday: 06:00 (Type 1)
MSE: 0.01342
MSE: 0.01343
Consumption (kWh)
Temperature (C°)

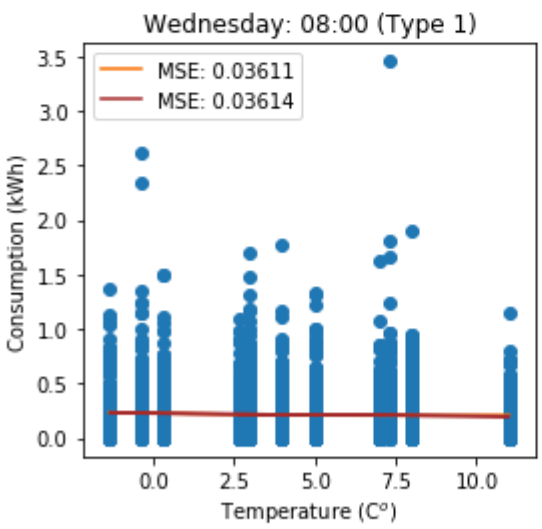

Wednesday: 06:00 (Type 2)
MSE: 0.11041
MSE: 0.11049
Consumption (kWh)
Temperature (C°)

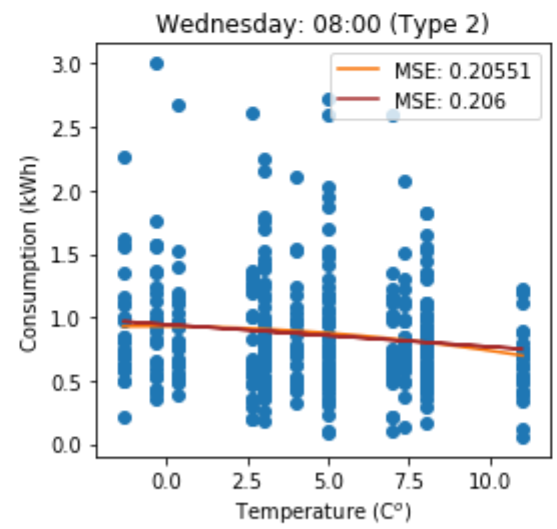

Wednesday: 06:00 (Type 3)
MSE: 0.02846
MSE: 0.0288
Consumption (kWh)
Temperature (C°)

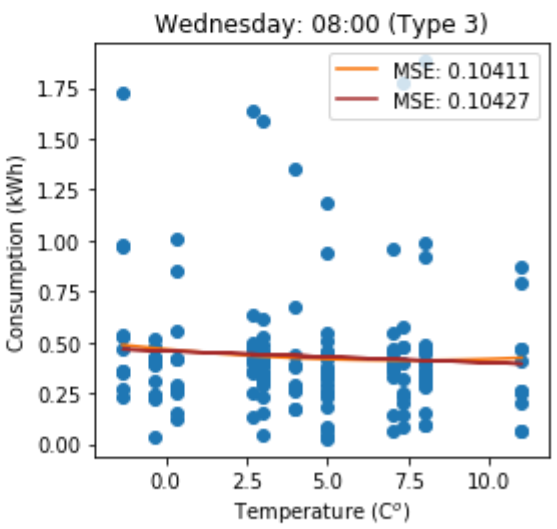

Wednesday: 07:00 (Type 1)
MSE: 0.02827
MSE: 0.02828
Consumption (kWh)
Temperature (C°)

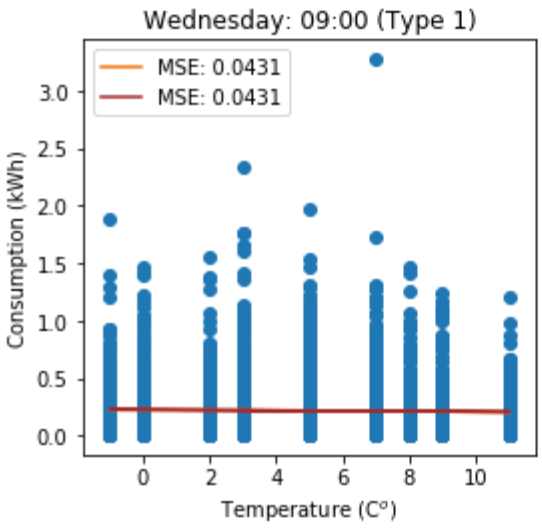

Wednesday: 07:00 (Type 2)
MSE: 0.24906
MSE: 0.24916
Consumption (kWh)
Temperature (C°)

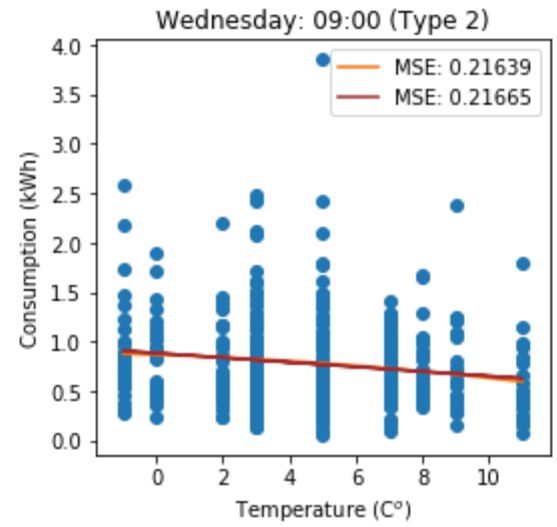

Wednesday: 07:00 (Type 3)
MSE: 0.08059
MSE: 0.08075
Consumption (kWh)
Temperature (C°)

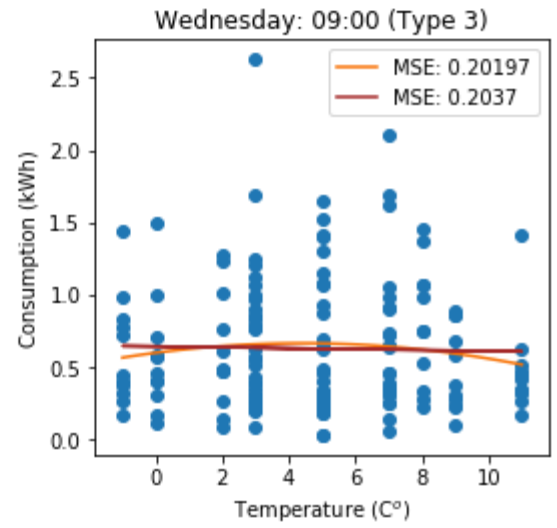

Wednesday: 08:00 (Type 1)
MSE: 0.03611
MSE: 0.03614
Consumption (kWh)
Temperature (C°)

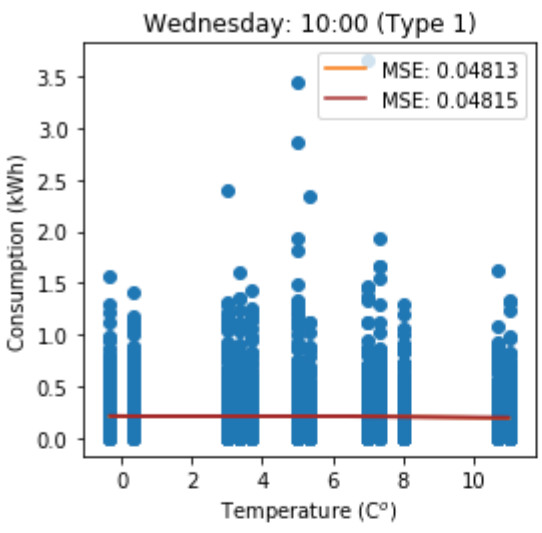

Wednesday: 08:00 (Type 2)
MSE: 0.20551
MSE: 0.206
Consumption (kWh)
Temperature (C°)

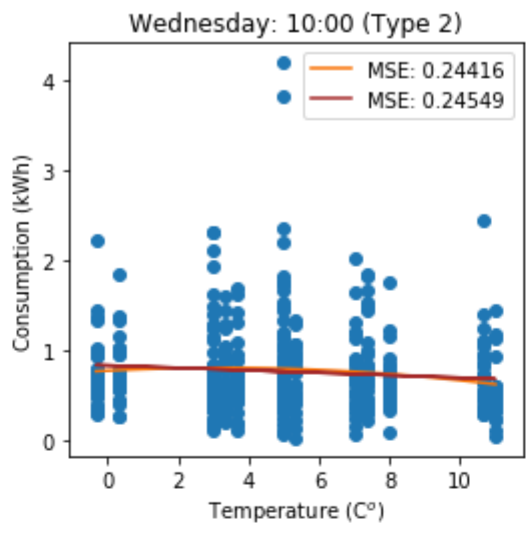

Wednesday: 08:00 (Type 3)
MSE: 0.10411
MSE: 0.10427
Consumption (kWh)
Temperature (C°)

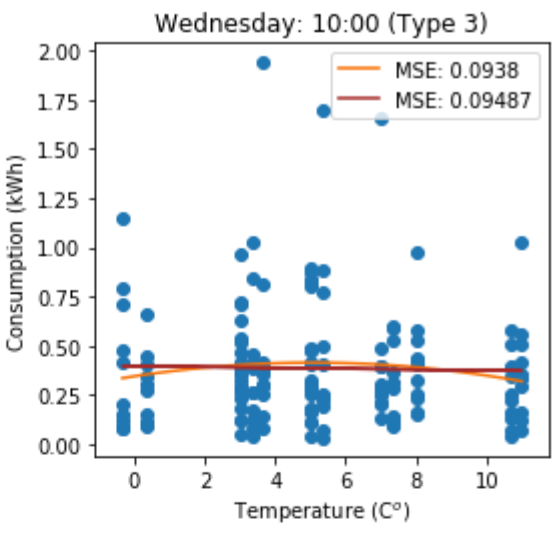

Wednesday: 09:00 (Type 1)
MSE: 0.0431
MSE: 0.0431
Consumption (kWh)
Temperature (C°)


Wednesday: 09:00 (Type 2)
MSE: 0.21639
MSE: 0.21665
Consumption (kWh)
Temperature (C°)


Wednesday: 09:00 (Type 3)
MSE: 0.20197
MSE: 0.2037
Consumption (kWh)
Temperature (C°)


Wednesday: 10:00 (Type 1)
MSE: 0.04813
MSE: 0.04815
Consumption (kWh)
Temperature (C°)

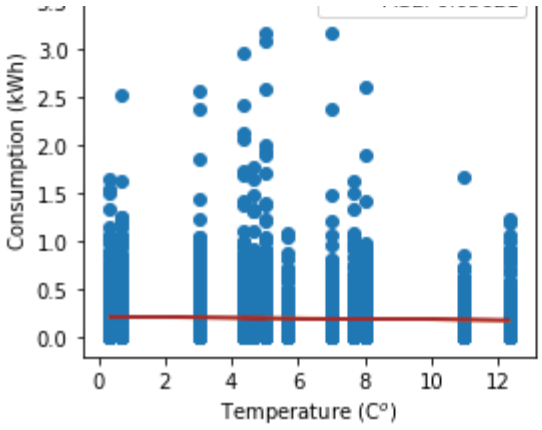

Wednesday: 10:00 (Type 2)
MSE: 0.24416
MSE: 0.24549
Consumption (kWh)
Temperature (C°)

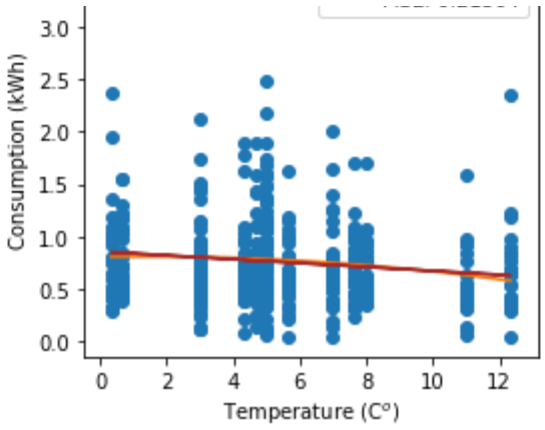

Wednesday: 10:00 (Type 3)
MSE: 0.0938
MSE: 0.09487
Consumption (kWh)
Temperature (C°)

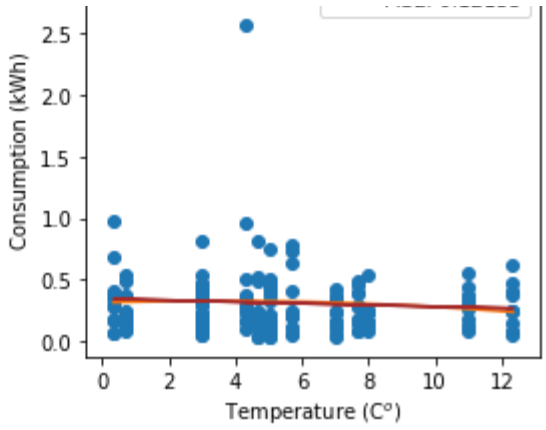

Wednesday: 11:00 (Type 1)
MSE: 0.0562
MSE: 0.05621

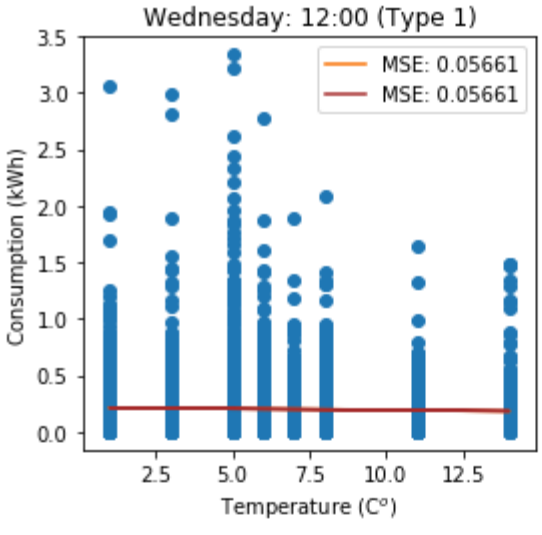

Wednesday: 11:00 (Type 2)
MSE: 0.21435
MSE: 0.21504

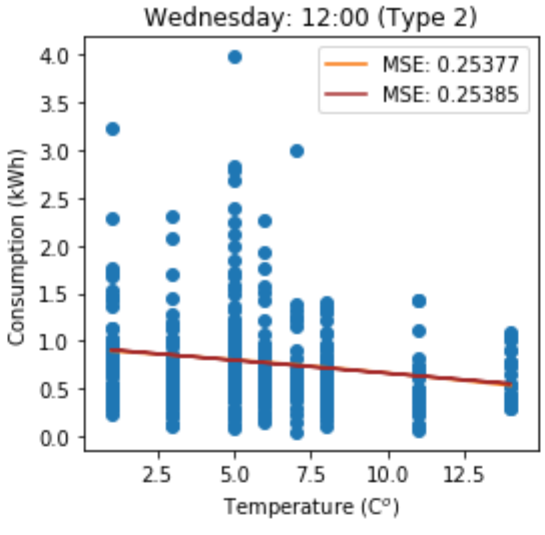

Wednesday: 11:00 (Type 3)
MSE: 0.12131
MSE: 0.12155

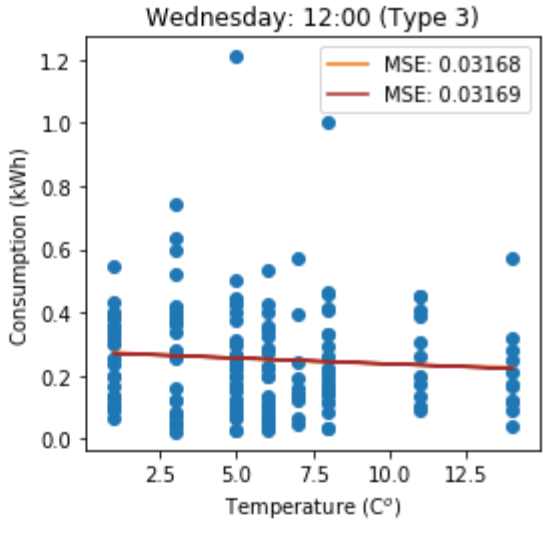

Consumption (kWh)
Temperature (C°)

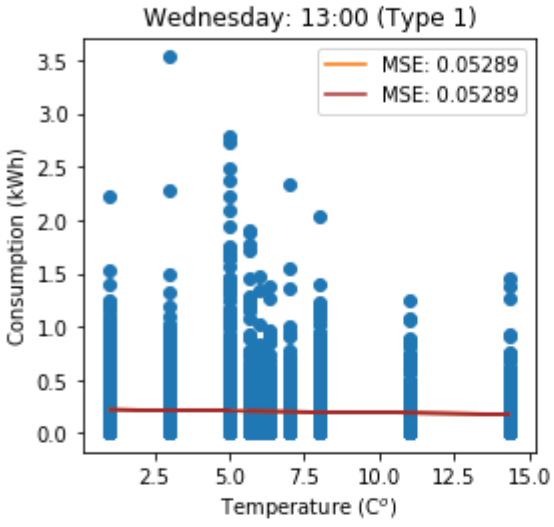

Consumption (kWh)
Temperature (C°)

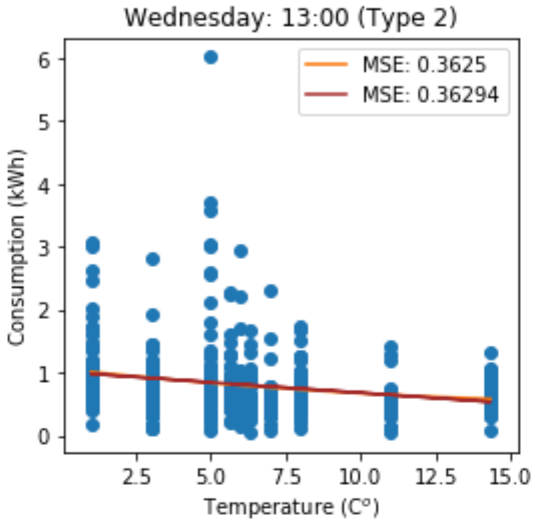

Consumption (kWh)
Temperature (C°)

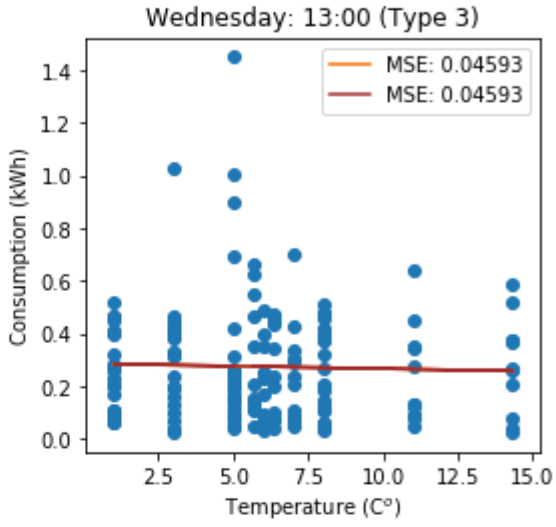

Wednesday: 12:00 (Type 1)
MSE: 0.05661
MSE: 0.05661
Consumption (kWh)
Temperature (C°)

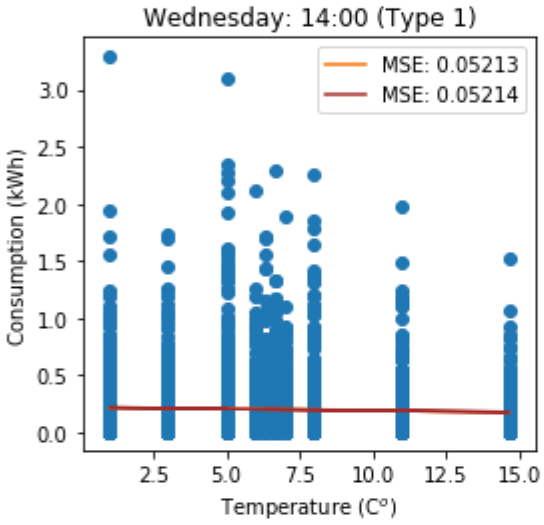

Wednesday: 12:00 (Type 2)
MSE: 0.25377
MSE: 0.25385
Consumption (kWh)
Temperature (C°)

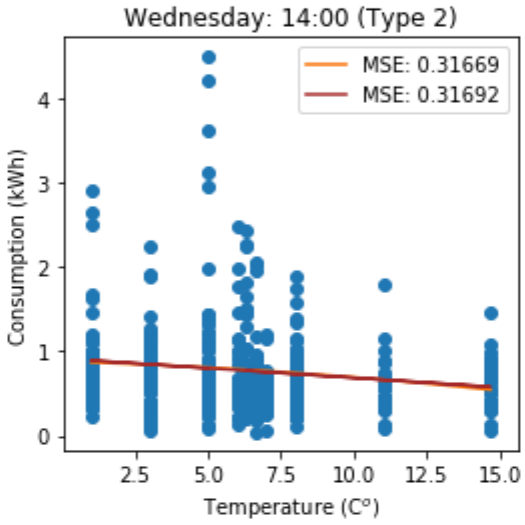

Wednesday: 12:00 (Type 3)
MSE: 0.03168
MSE: 0.03169
Consumption (kWh)
Temperature (C°)

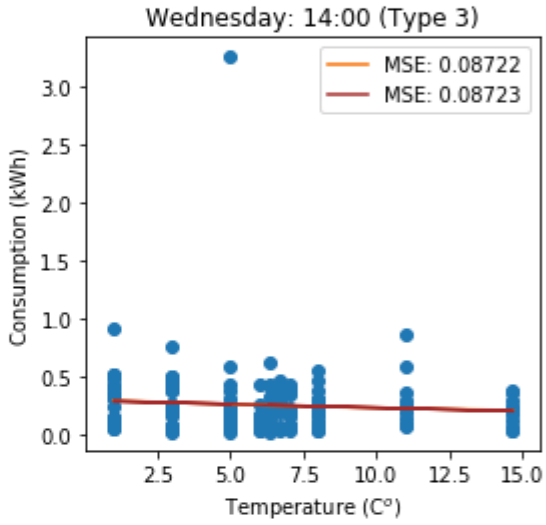

Wednesday: 13:00 (Type 1)
MSE: 0.05289
MSE: 0.05289
Consumption (kWh)
Temperature (C°)

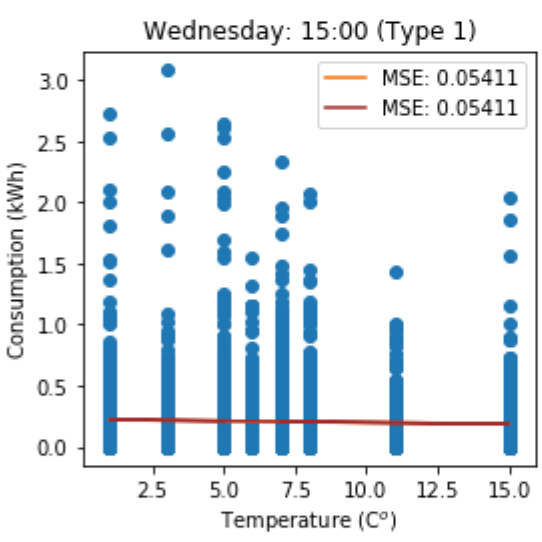

Wednesday: 13:00 (Type 2)
MSE: 0.3625
MSE: 0.36294
Consumption (kWh)
Temperature (C°)

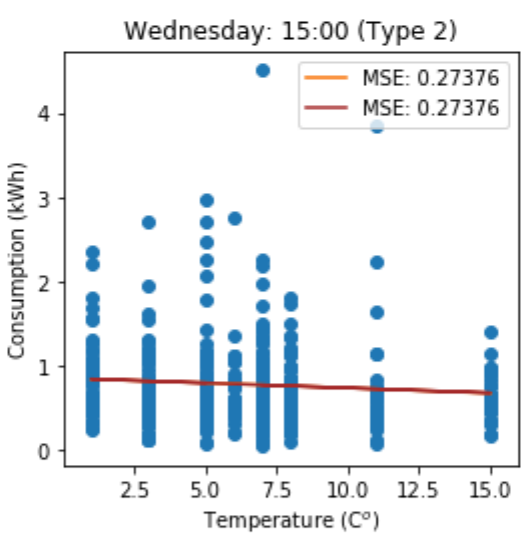

Wednesday: 13:00 (Type 3)
MSE: 0.04593
MSE: 0.04593
Consumption (kWh)
Temperature (C°)

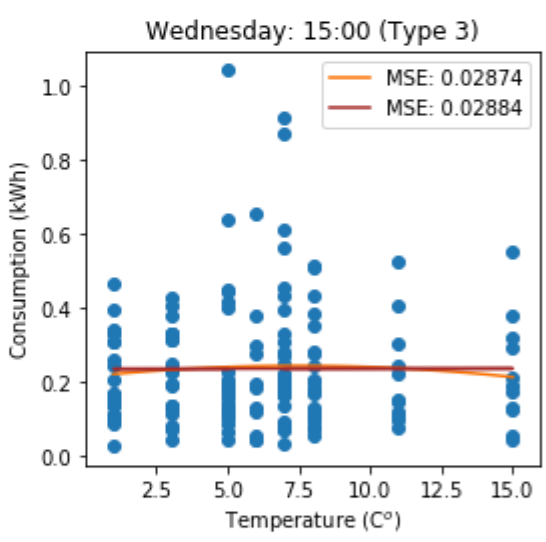

Wednesday: 14:00 (Type 1)
MSE: 0.05213
MSE: 0.05214
Consumption (kWh)
Temperature (C°)

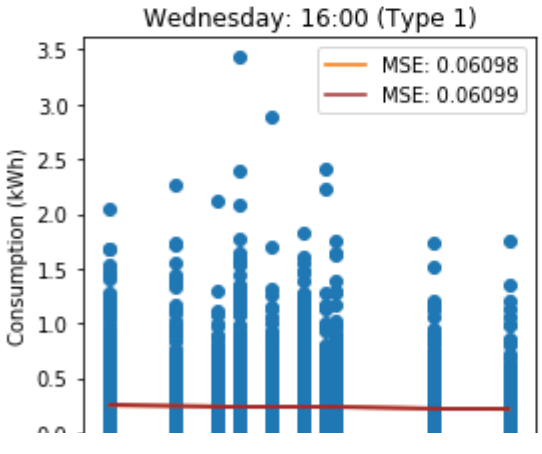

Wednesday: 14:00 (Type 2)
MSE: 0.31669
MSE: 0.31692
Consumption (kWh)
Temperature (C°)

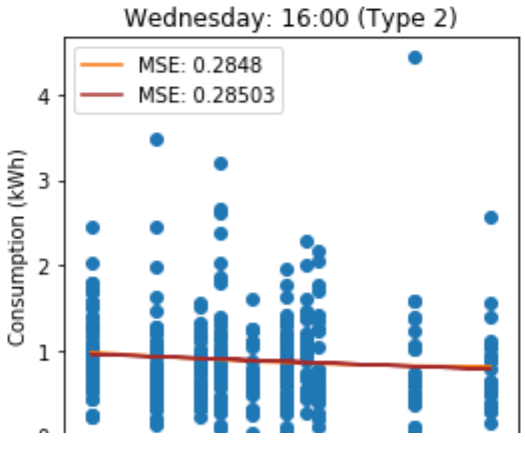

Wednesday: 14:00 (Type 3)
MSE: 0.08722
MSE: 0.08723
Consumption (kWh)
Temperature (C°)

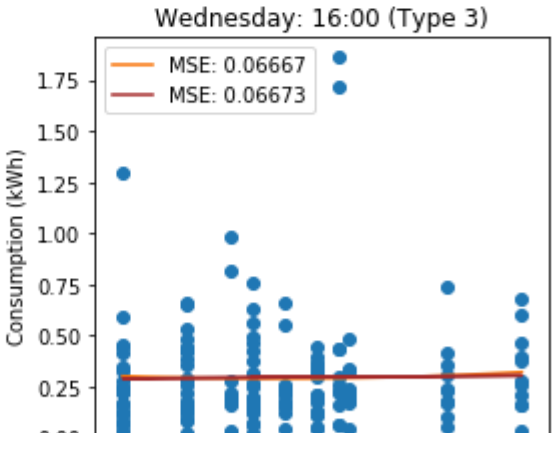

Wednesday: 15:00 (Type 1)
MSE: 0.05411
MSE: 0.05411
Consumption (kWh)
Temperature (C°)

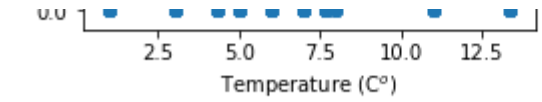

Wednesday: 15:00 (Type 2)
MSE: 0.27376
MSE: 0.27376
Consumption (kWh)
Temperature (C°)

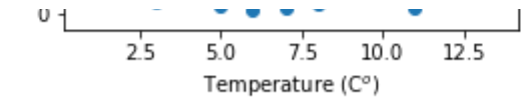

Wednesday: 15:00 (Type 3)
MSE: 0.02874
MSE: 0.02884
Consumption (kWh)
Temperature (C°)

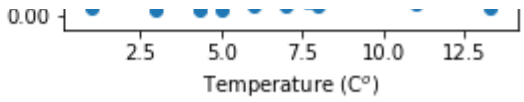

Wednesday: 16:00 (Type 1)
MSE: 0.06098
MSE: 0.06099
Consumption (kWh)

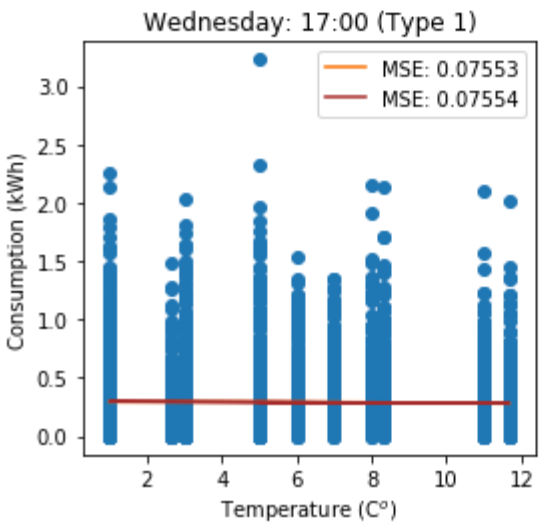

Wednesday: 16:00 (Type 2)
MSE: 0.2848
MSE: 0.28503
Consumption (kWh)

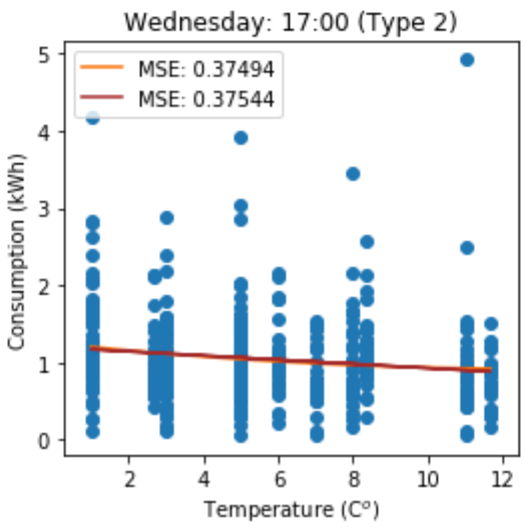

Wednesday: 16:00 (Type 3)
MSE: 0.06667
MSE: 0.06673
Consumption (kWh)

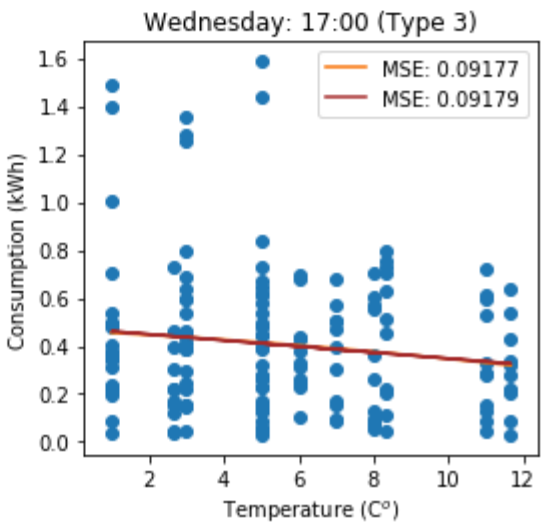

Temperature (C°)

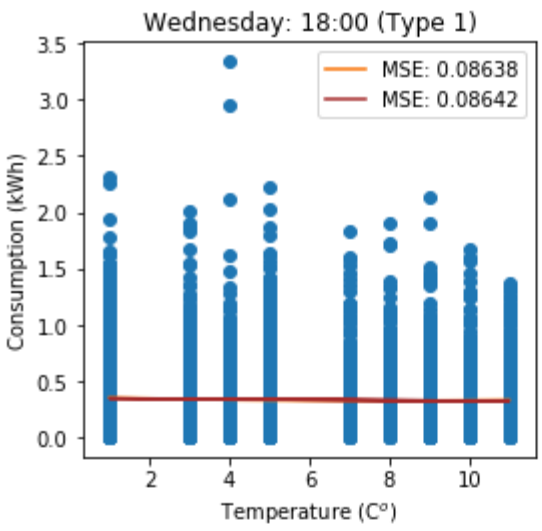

Temperature (C°)

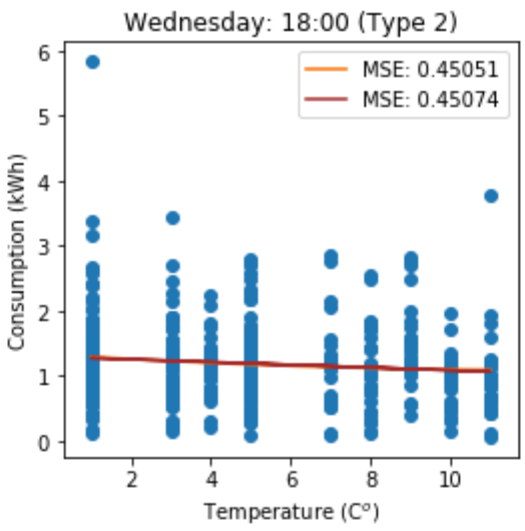

Temperature (C°)

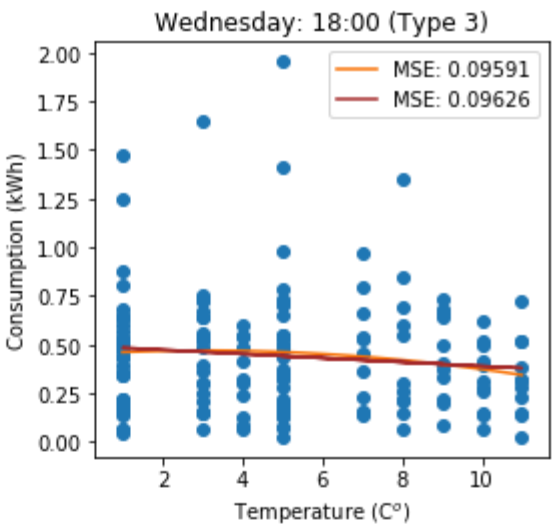

Wednesday: 17:00 (Type 1)
MSE: 0.07553
MSE: 0.07554
Consumption (kWh)
Temperature (C°)

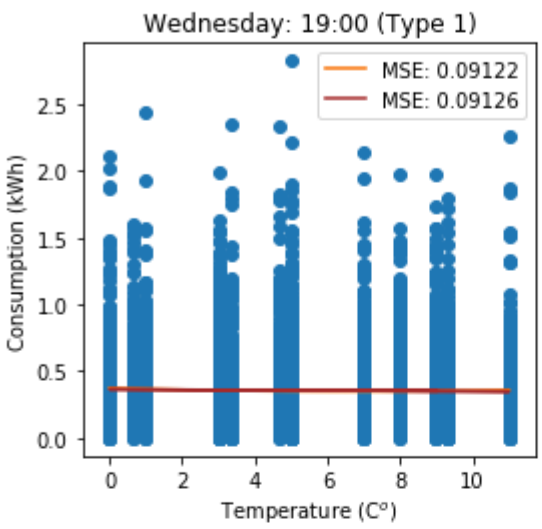

Wednesday: 17:00 (Type 2)
MSE: 0.37494
MSE: 0.37544
Consumption (kWh)
Temperature (C°)

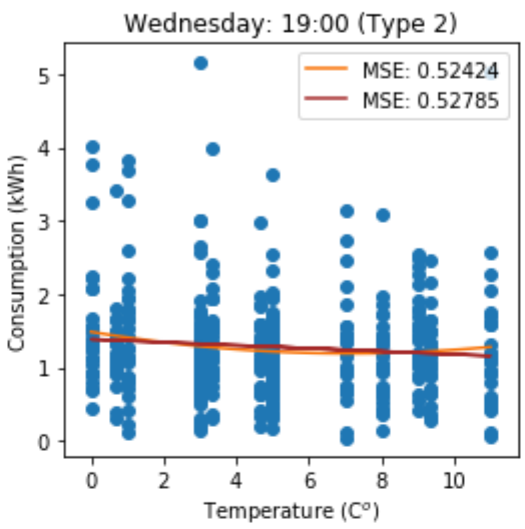

Wednesday: 17:00 (Type 3)
MSE: 0.09177
MSE: 0.09179
Consumption (kWh)
Temperature (C°)

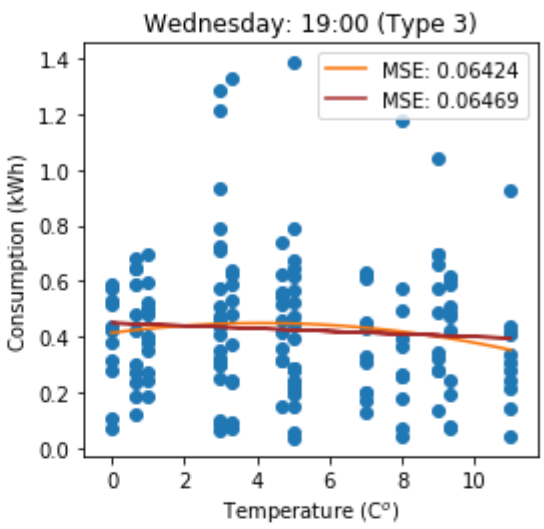

Wednesday: 18:00 (Type 1)
MSE: 0.08638
MSE: 0.08642
Consumption (kWh)
Temperature (C°)

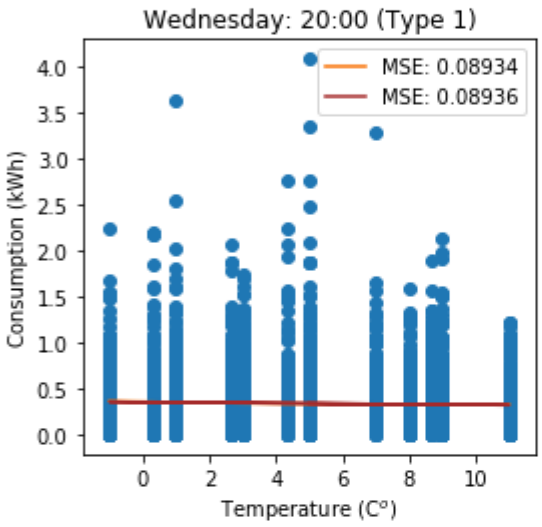

Wednesday: 18:00 (Type 2)
MSE: 0.45051
MSE: 0.45074
Consumption (kWh)
Temperature (C°)

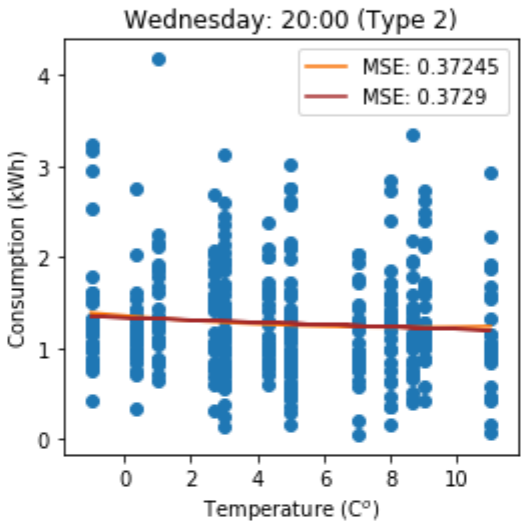

Wednesday: 18:00 (Type 3)
MSE: 0.09591
MSE: 0.09626
Consumption (kWh)
Temperature (C°)

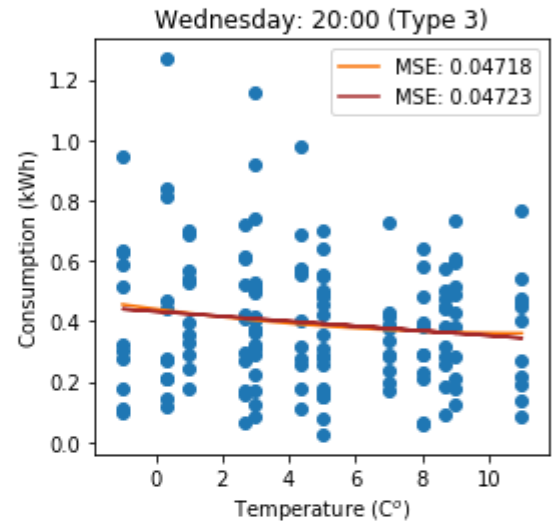

Wednesday: 19:00 (Type 1)
MSE: 0.09122
MSE: 0.09126
Consumption (kWh)
Temperature (C°)

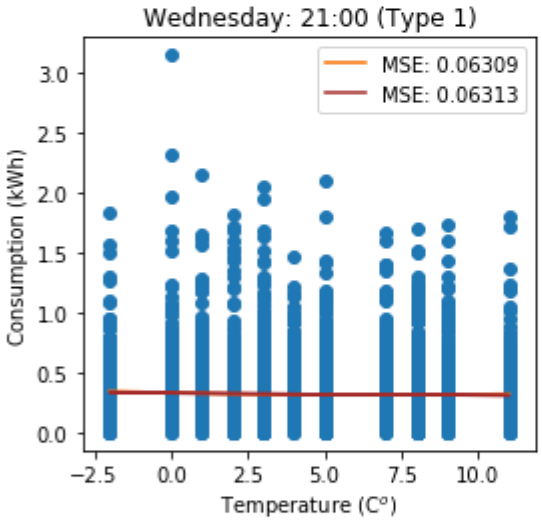

Wednesday: 19:00 (Type 2)
MSE: 0.52424
MSE: 0.52785
Consumption (kWh)
Temperature (C°)

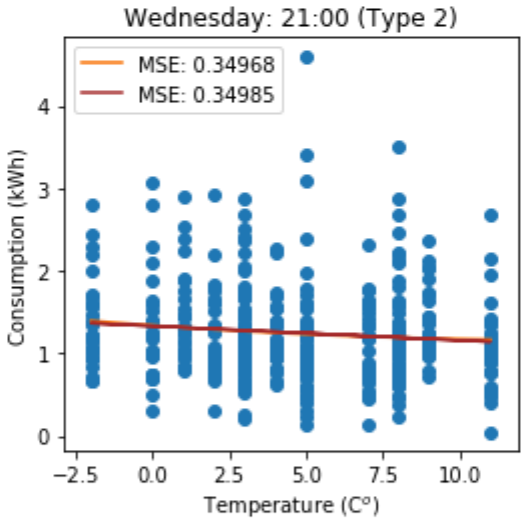

Wednesday: 19:00 (Type 3)
MSE: 0.06424
MSE: 0.06469
Consumption (kWh)
Temperature (C°)

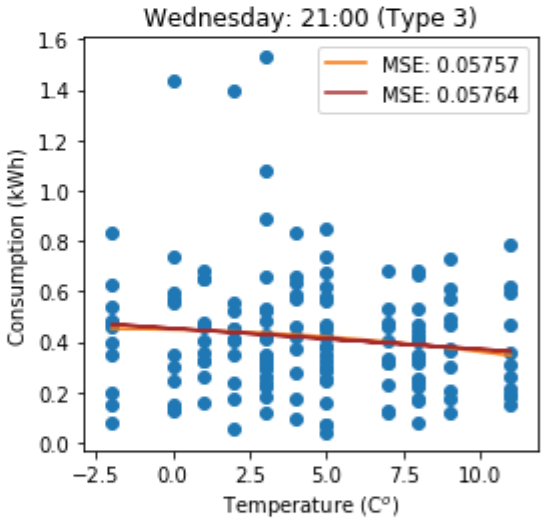

Wednesday: 20:00 (Type 1)
MSE: 0.08934
MSE: 0.08936
Consumption (kWh)
Temperature (C°)

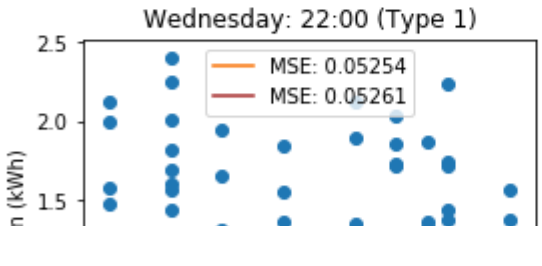

Wednesday: 20:00 (Type 2)
MSE: 0.37245
MSE: 0.3729
Consumption (kWh)
Temperature (C°)

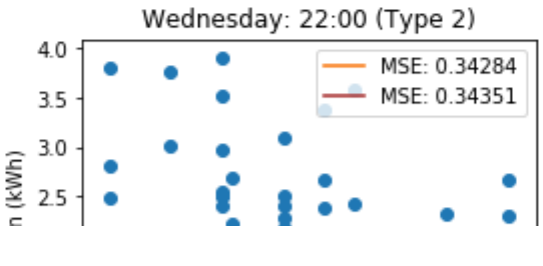

Wednesday: 20:00 (Type 3)
MSE: 0.04718
MSE: 0.04723
Consumption (kWh)
Temperature (C°)

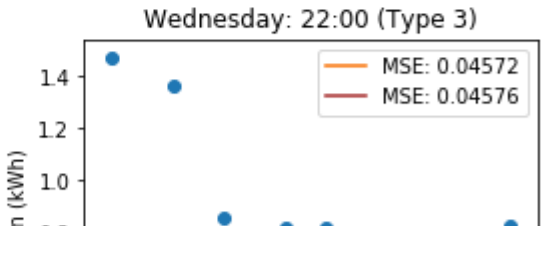

Wednesday: 21:00 (Type 1)
MSE: 0.06309
MSE: 0.06313
Consumption (kWh)
Temperature (C°)

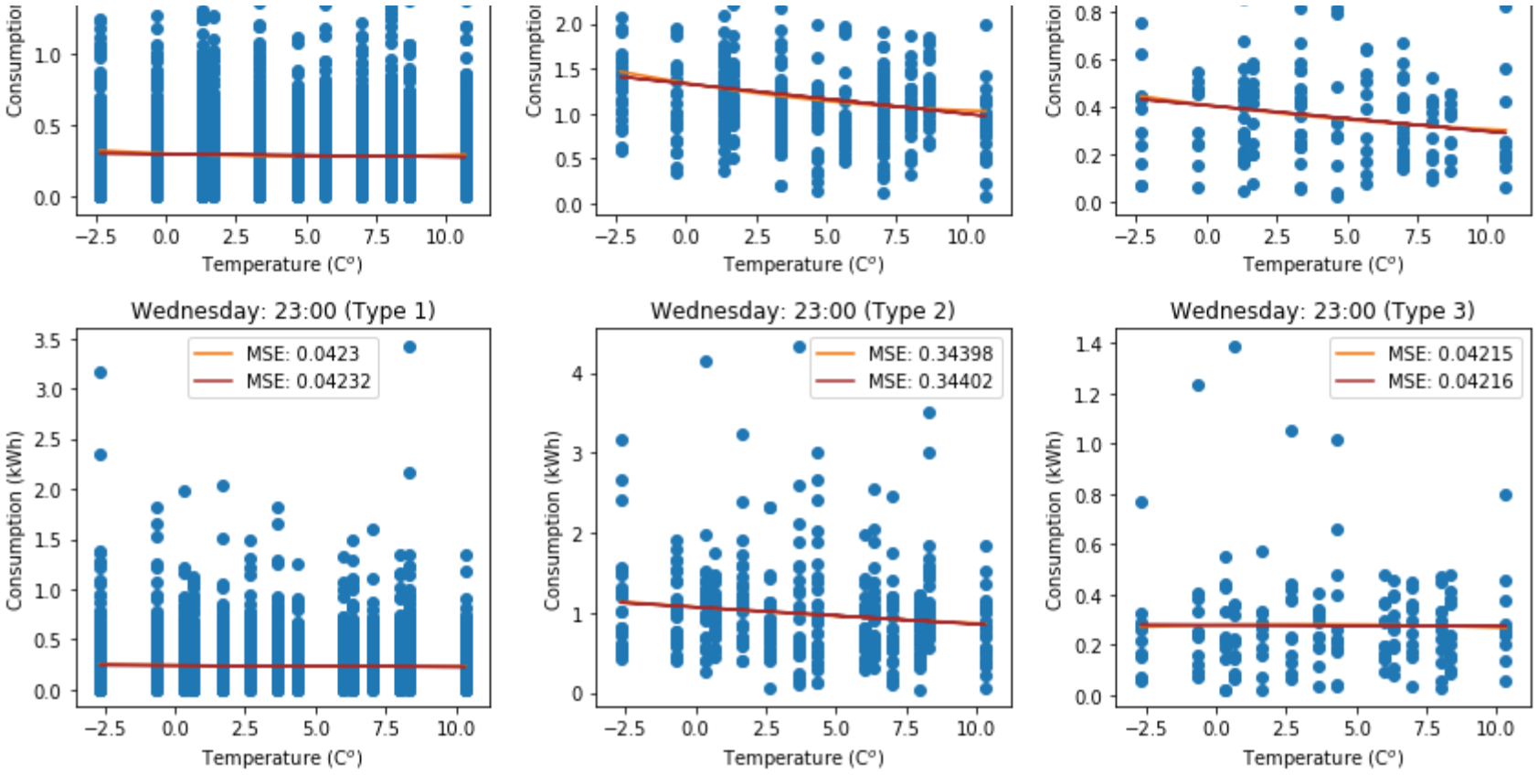

Wednesday: 21:00 (Type 2)
MSE: 0.34968
MSE: 0.34985
Consumption (kWh)
Temperature (C°)


Wednesday: 21:00 (Type 3)
MSE: 0.05757
MSE: 0.05764
Consumption (kWh)
Temperature (C°)

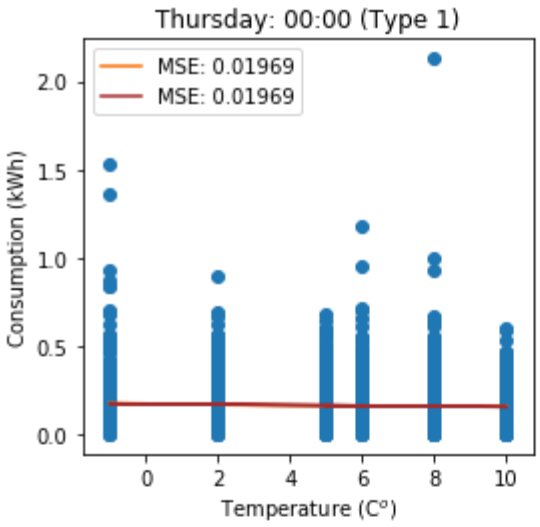

Wednesday: 22:00 (Type 1)
MSE: 0.05254
MSE: 0.05261

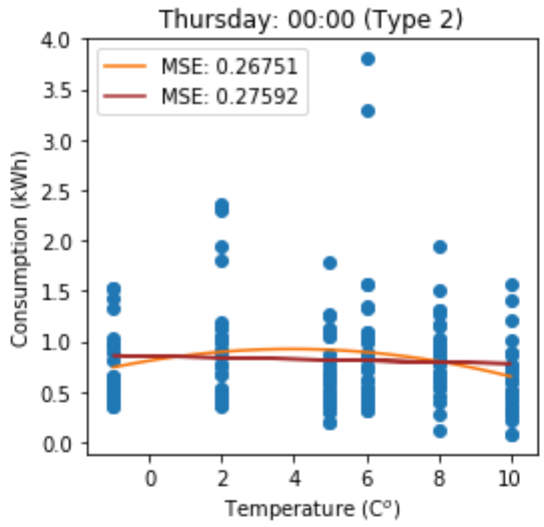

Wednesday: 22:00 (Type 2)
MSE: 0.34284
MSE: 0.34351

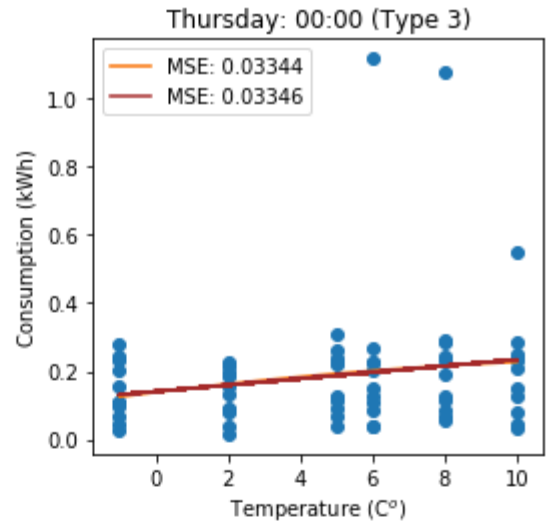

Wednesday: 22:00 (Type 3)
MSE: 0.04572
MSE: 0.04576

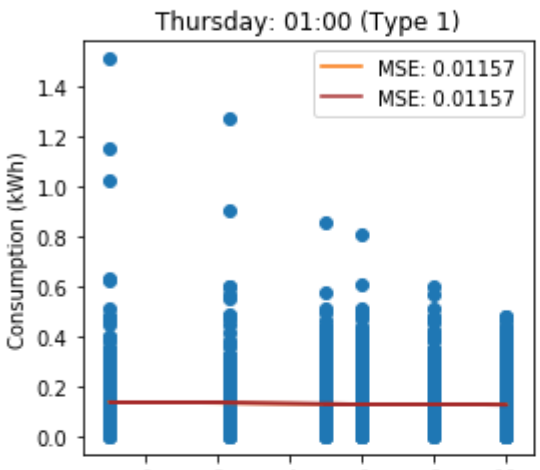

Wednesday: 23:00 (Type 1)
MSE: 0.0423
MSE: 0.04232
Wednesday: 23:00 (Type 2)
MSE: 0.34398
MSE: 0.34402
Wednesday: 23:00 (Type 3)
MSE: 0.04215
MSE: 0.04216
Consumption (kWh)
Temperature (C°)

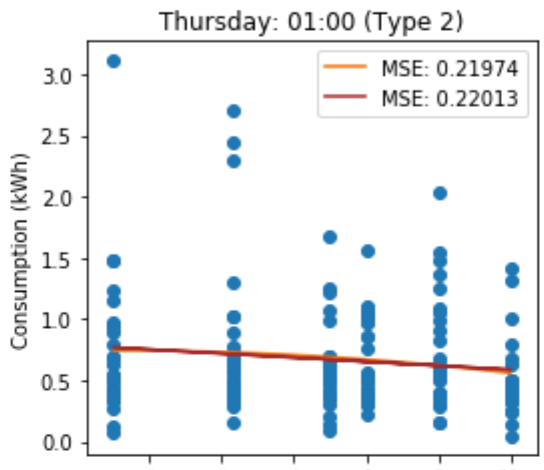
In [11]:
```
# Thursday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 3 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

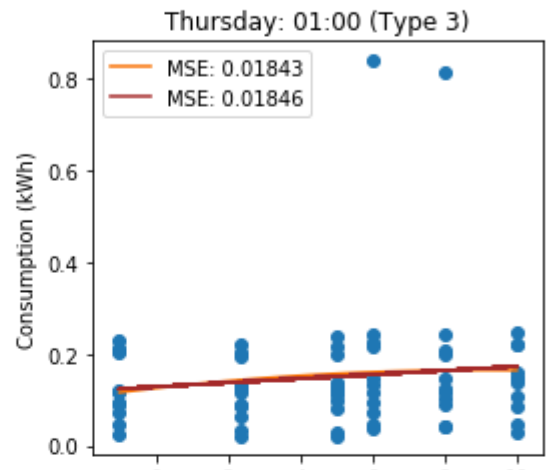

Thursday: 00:00 (Type 1)
MSE: 0.01969
MSE: 0.01969
Consumption (kWh)
Temperature (C°)

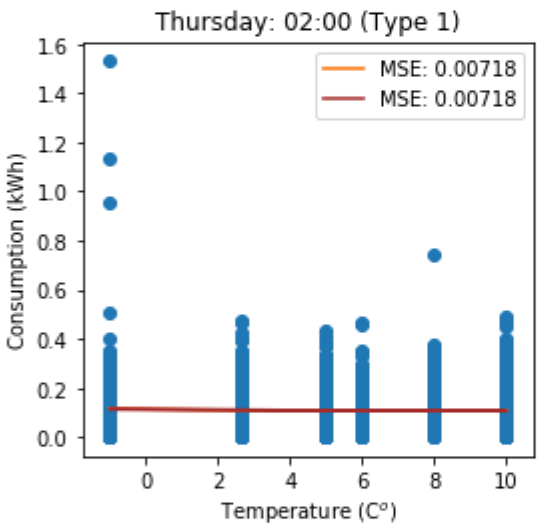

Thursday: 00:00 (Type 2)
MSE: 0.26751
MSE: 0.27592
Consumption (kWh)
Temperature (C°)

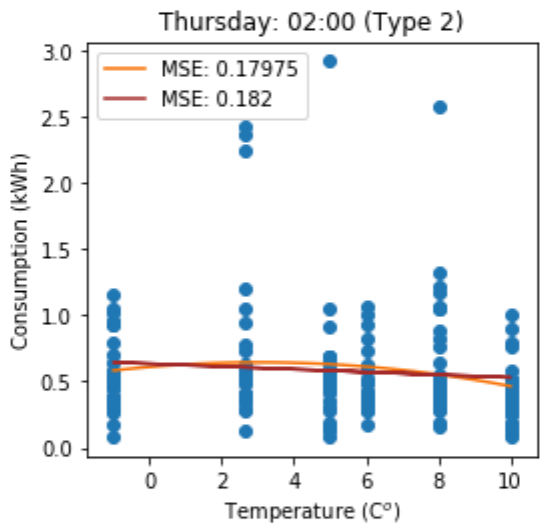

Thursday: 00:00 (Type 3)
MSE: 0.03344
MSE: 0.03346
Consumption (kWh)
Temperature (C°)

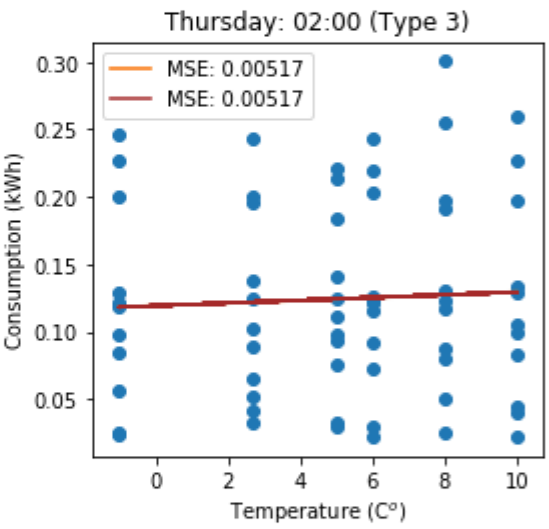

Thursday: 01:00 (Type 1)
MSE: 0.01157
MSE: 0.01157
Consumption (kWh)
Temperature (C°)

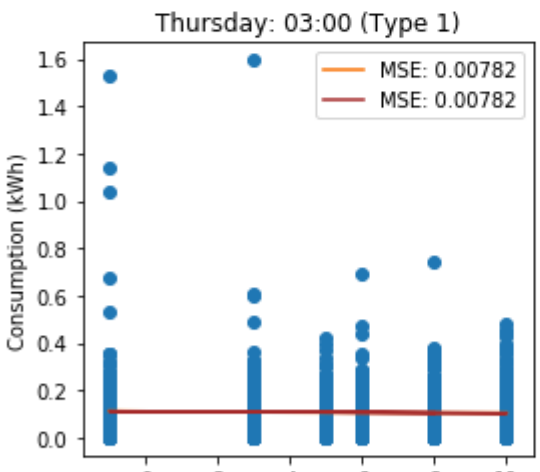

Thursday: 01:00 (Type 2)
MSE: 0.21974
MSE: 0.22013
Consumption (kWh)
Temperature (C°)

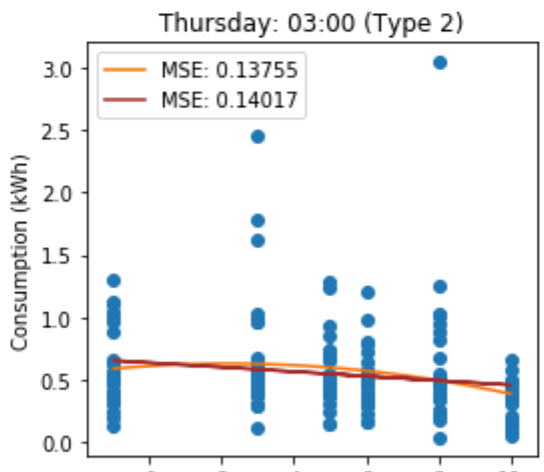

Thursday: 01:00 (Type 3)
MSE: 0.01843
MSE: 0.01846
Consumption (kWh)
Temperature (C°)

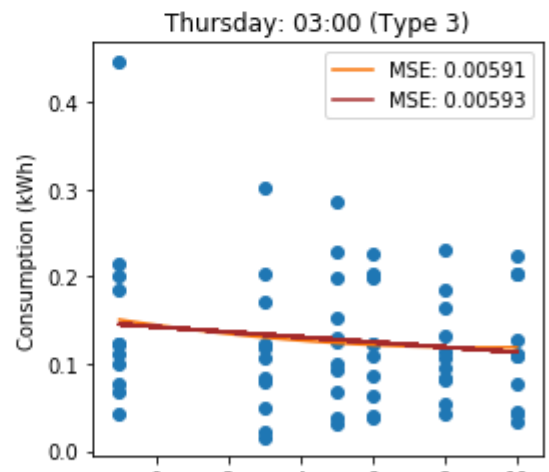

Thursday: 02:00 (Type 1)
MSE: 0.00718
MSE: 0.00718
Consumption (kWh)
Temperature (C°)

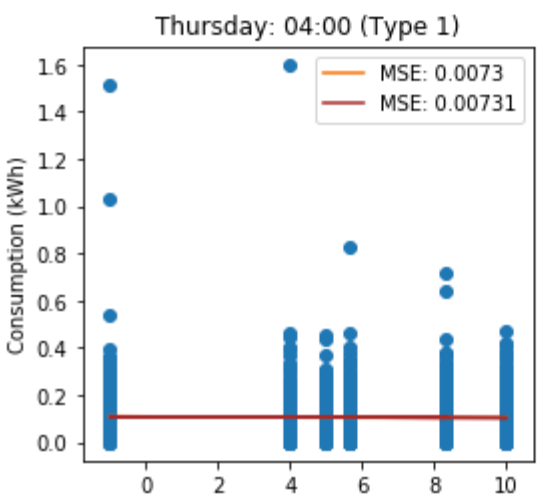

Thursday: 02:00 (Type 2)
MSE: 0.17975
MSE: 0.182
Consumption (kWh)
Temperature (C°)

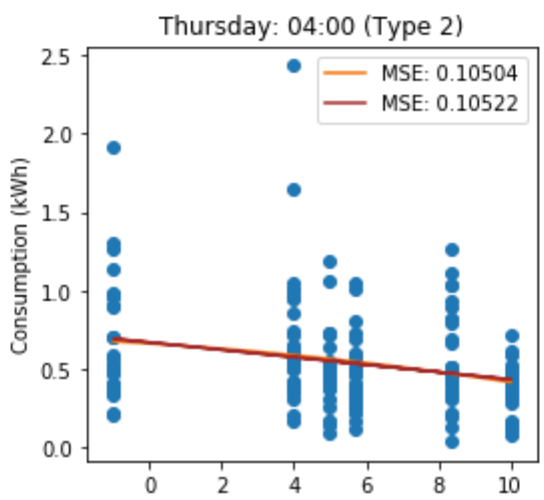

Thursday: 02:00 (Type 3)
MSE: 0.00517
MSE: 0.00517
Consumption (kWh)
Temperature (C°)

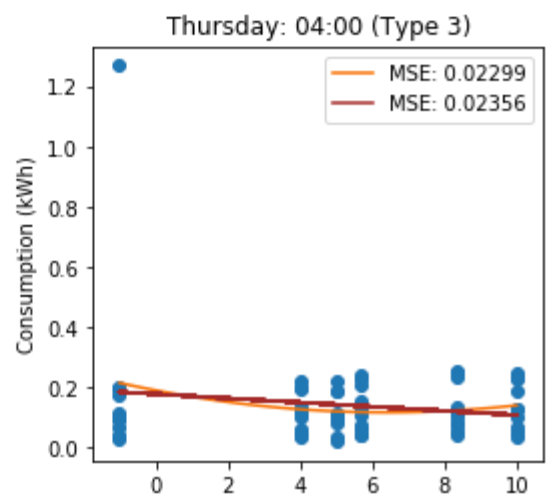

Thursday: 03:00 (Type 1)
MSE: 0.00782
MSE: 0.00782
Consumption (kWh)
Temperature (C°)

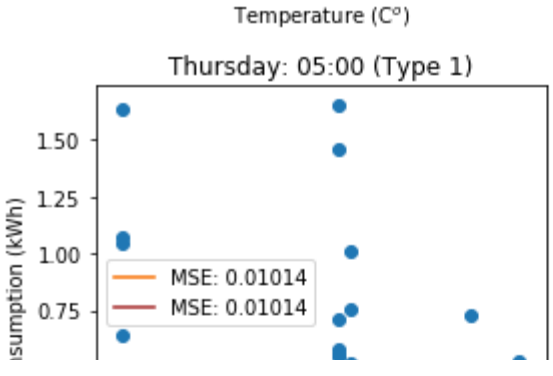

Thursday: 03:00 (Type 2)
MSE: 0.13755
MSE: 0.14017
Consumption (kWh)
Temperature (C°)

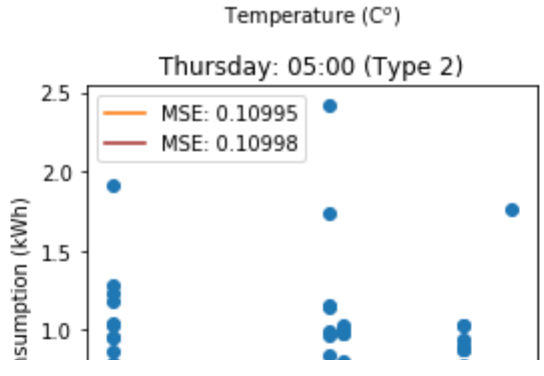

Thursday: 03:00 (Type 3)
MSE: 0.00591
MSE: 0.00593
Consumption (kWh)
Temperature (C°)

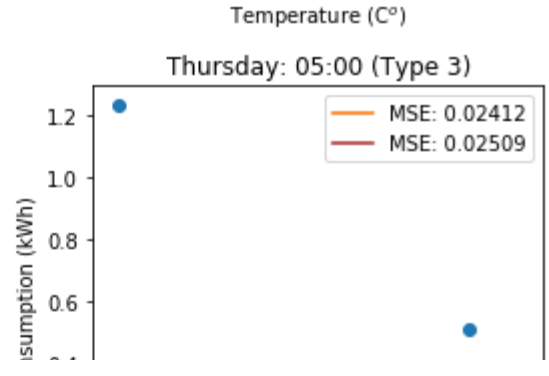

Thursday: 04:00 (Type 1)
MSE: 0.0073
MSE: 0.00731
Consumption (kWh)
Temperature (C°)

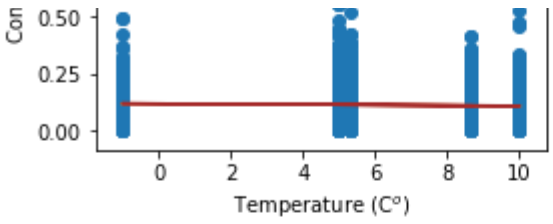

Thursday: 04:00 (Type 2)
MSE: 0.10504
MSE: 0.10522
Consumption (kWh)
Temperature (C°)

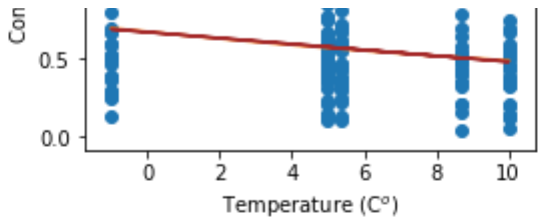

Thursday: 04:00 (Type 3)
MSE: 0.02299
MSE: 0.02356
Consumption (kWh)
Temperature (C°)

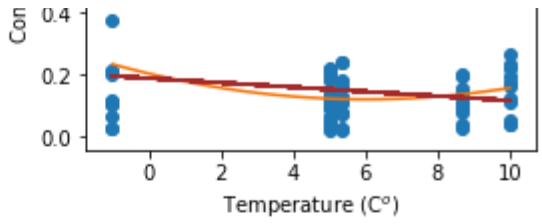

Thursday: 05:00 (Type 1)
MSE: 0.01014
MSE: 0.01014

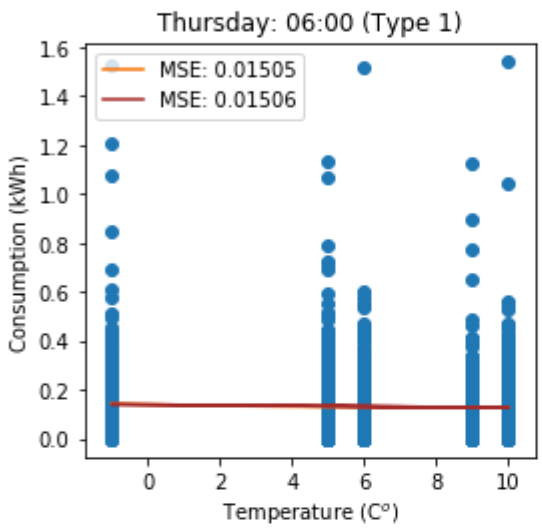

Thursday: 05:00 (Type 2)
MSE: 0.10995
MSE: 0.10998

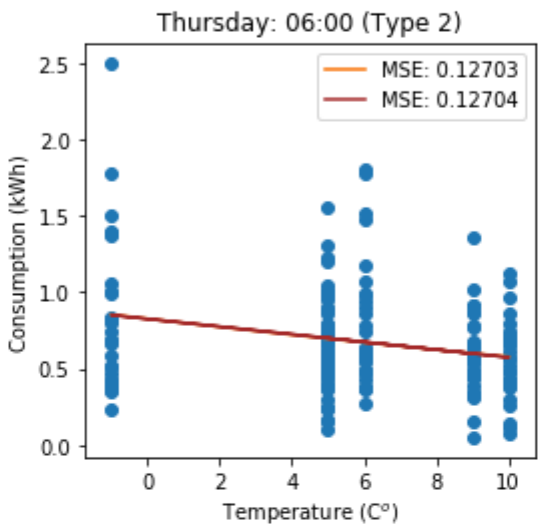

Thursday: 05:00 (Type 3)
MSE: 0.02412
MSE: 0.02509

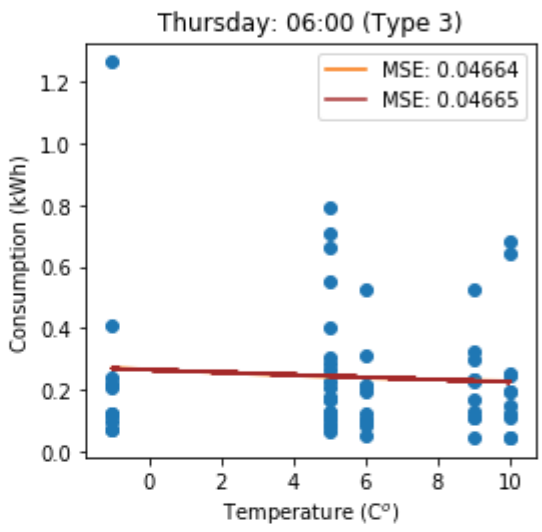

Con
0.50
0.25
0.00
0
2
4
6
8
10
Temperature (C°)

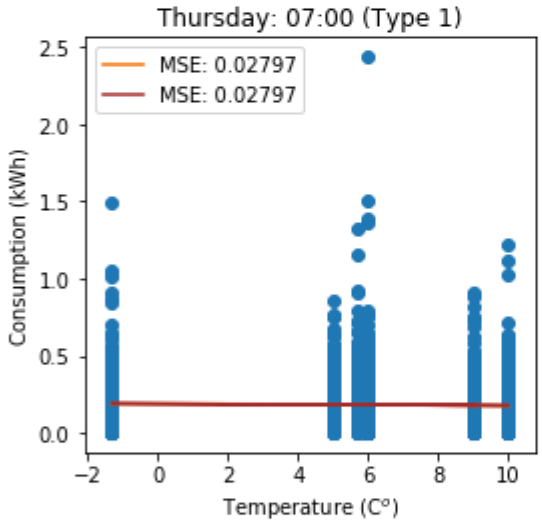

Con
0.5
0.0
0
2
4
6
8
10
Temperature (C°)

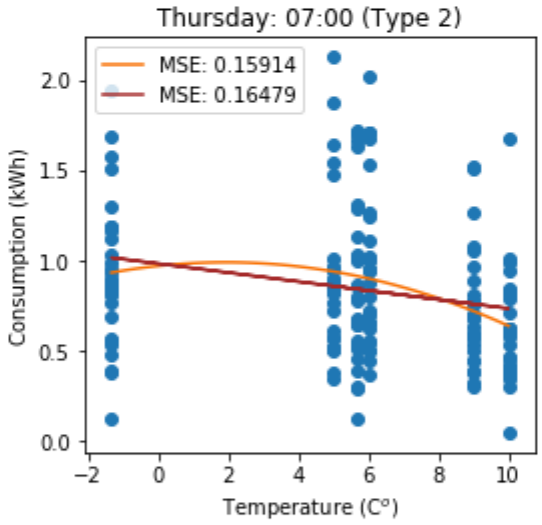

Con
0.4
0.2
0.0
0
2
4
6
8
10
Temperature (C°)

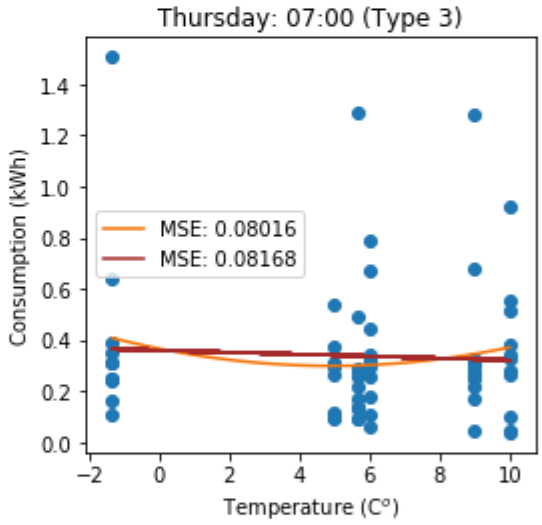

Thursday: 06:00 (Type 1)
MSE: 0.01505
MSE: 0.01506
Consumption (kWh)
Temperature (C°)

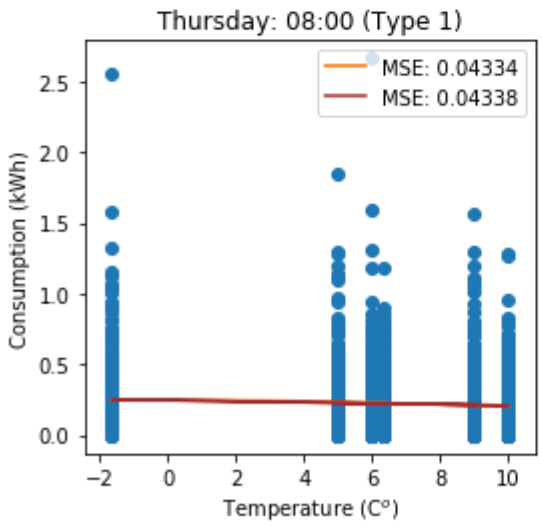

Thursday: 06:00 (Type 2)
MSE: 0.12703
MSE: 0.12704
Consumption (kWh)
Temperature (C°)

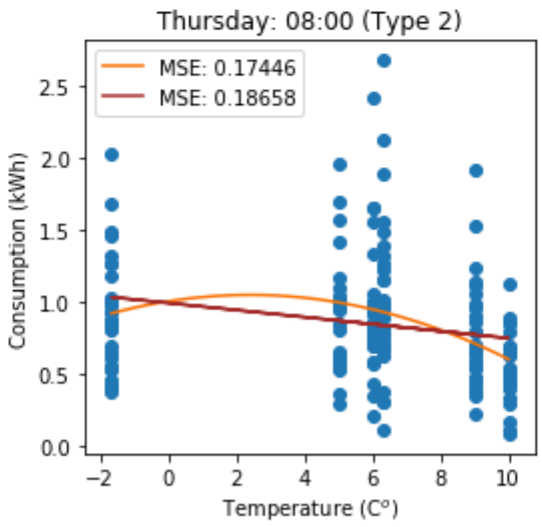

Thursday: 06:00 (Type 3)
MSE: 0.04664
MSE: 0.04665
Consumption (kWh)
Temperature (C°)

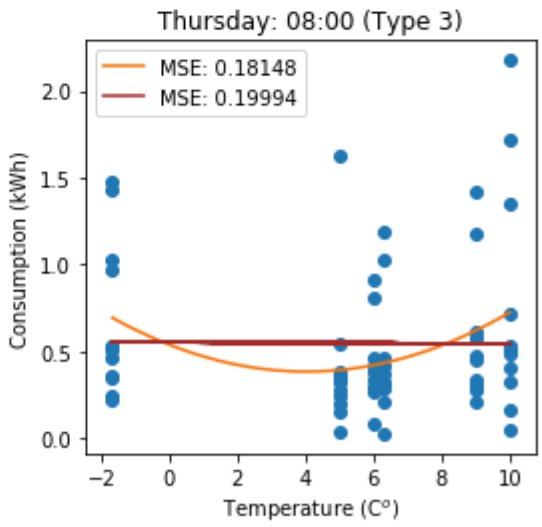

Thursday: 07:00 (Type 1)
MSE: 0.02797
MSE: 0.02797
Consumption (kWh)
Temperature (C°)

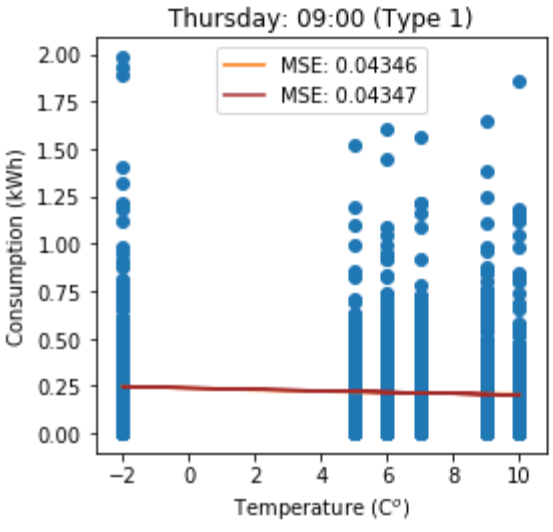

Thursday: 07:00 (Type 2)
MSE: 0.15914
MSE: 0.16479
Consumption (kWh)
Temperature (C°)

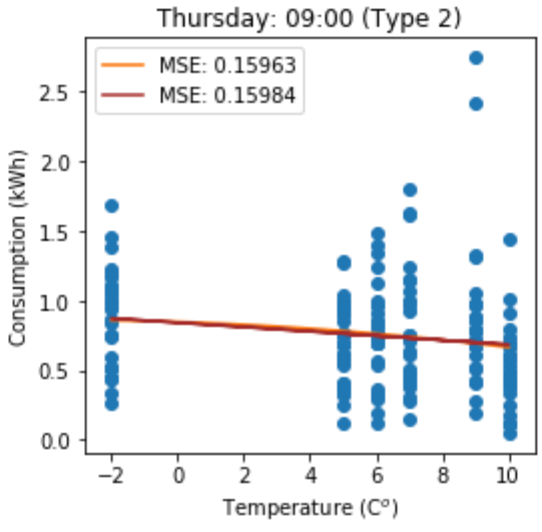

Thursday: 07:00 (Type 3)
MSE: 0.08016
MSE: 0.08168
Consumption (kWh)
Temperature (C°)

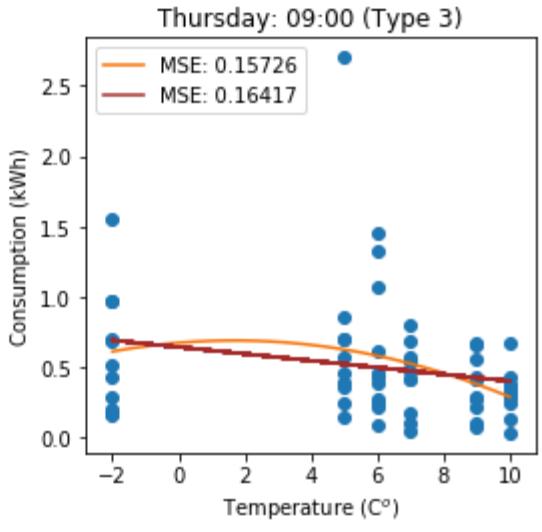

Thursday: 08:00 (Type 1)
MSE: 0.04334
MSE: 0.04338
Consumption (kWh)
Temperature (C°)

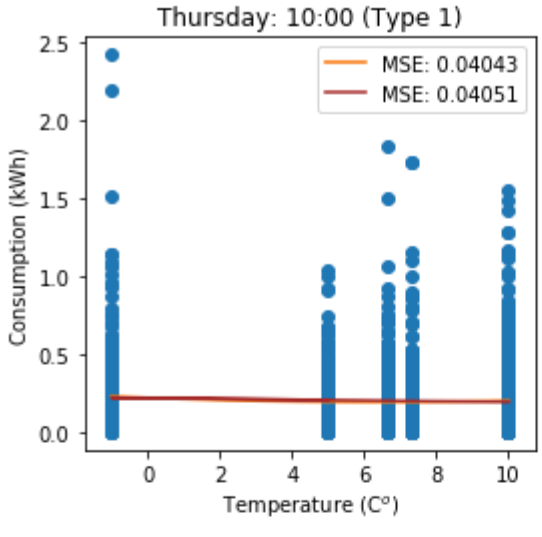

Thursday: 08:00 (Type 2)
MSE: 0.17446
MSE: 0.18658
Consumption (kWh)
Temperature (C°)

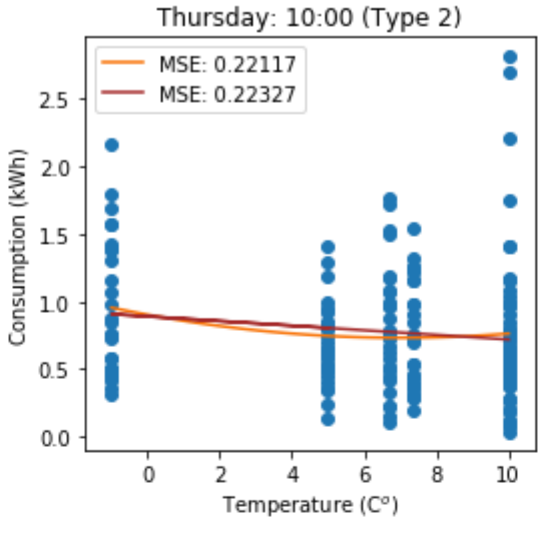

Thursday: 08:00 (Type 3)
MSE: 0.18148
MSE: 0.19994
Consumption (kWh)
Temperature (C°)

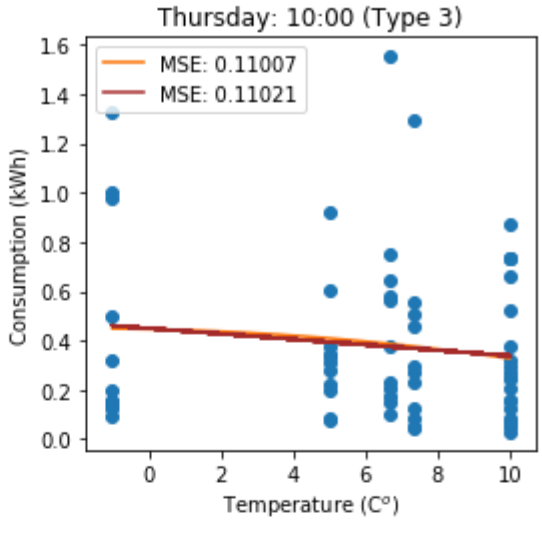

Thursday: 09:00 (Type 1)
MSE: 0.04346
MSE: 0.04347
Consumption (kWh)
Temperature (C°)

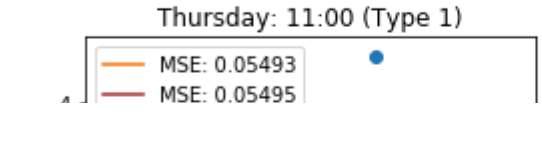

Thursday: 09:00 (Type 2)
MSE: 0.15963
MSE: 0.15984
Consumption (kWh)
Temperature (C°)

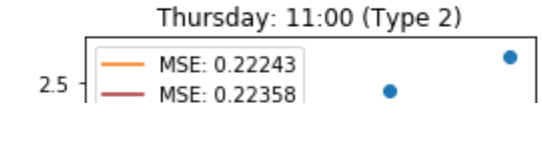

Thursday: 09:00 (Type 3)
MSE: 0.15726
MSE: 0.16417
Consumption (kWh)
Temperature (C°)

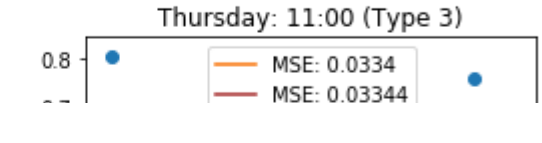

Thursday: 10:00 (Type 1)
MSE: 0.04043
MSE: 0.04051
Consumption (kWh)
Temperature (C°)

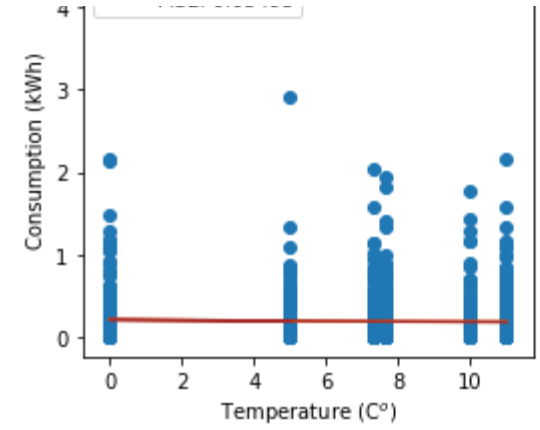

Thursday: 10:00 (Type 2)
MSE: 0.22117
MSE: 0.22327
Consumption (kWh)
Temperature (C°)

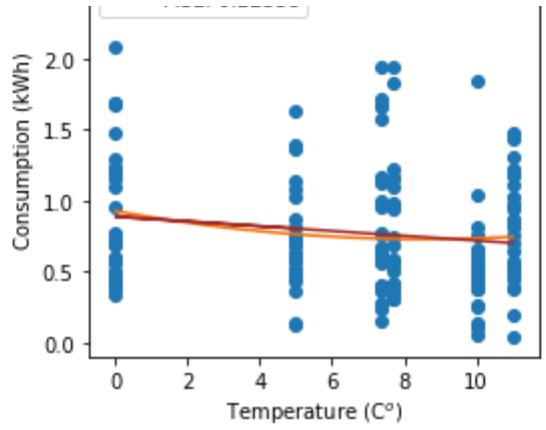

Thursday: 10:00 (Type 3)
MSE: 0.11007
MSE: 0.11021
Consumption (kWh)
Temperature (C°)

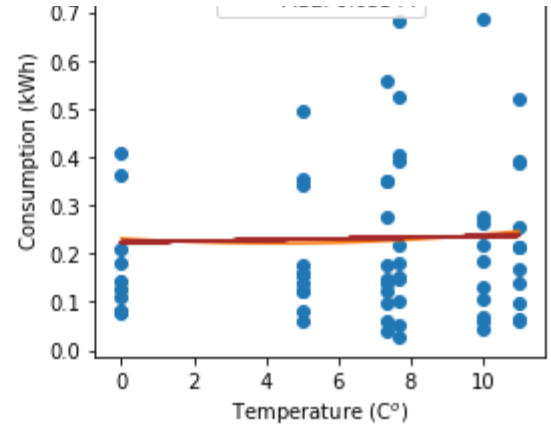

Thursday: 11:00 (Type 1)
MSE: 0.05493
MSE: 0.05495

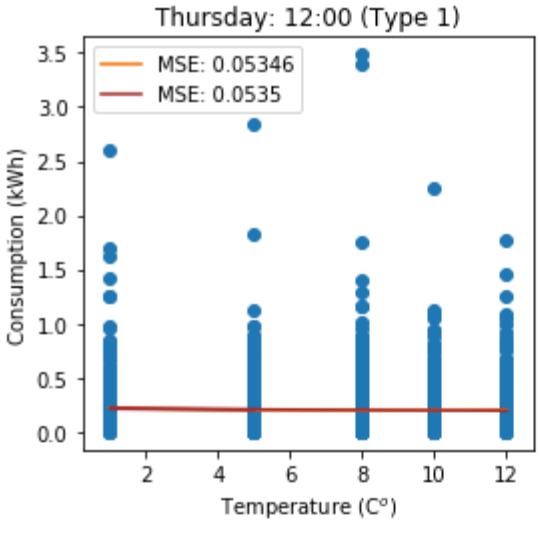

Thursday: 11:00 (Type 2)
MSE: 0.22243
MSE: 0.22358

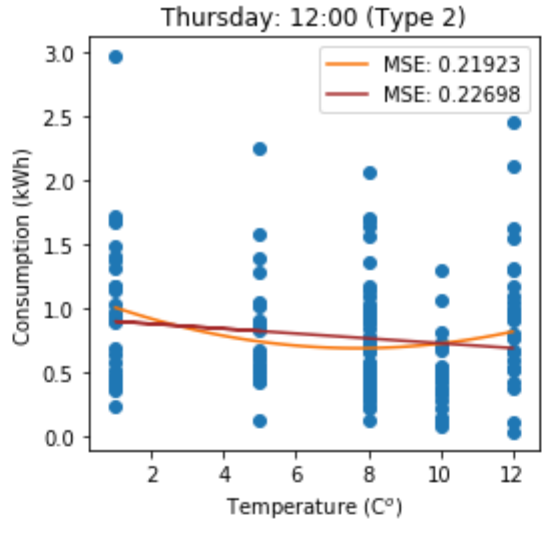

Thursday: 11:00 (Type 3)
MSE: 0.0334
MSE: 0.03344

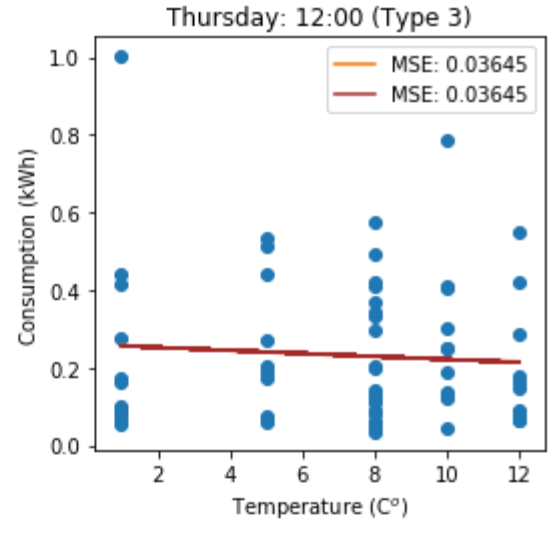

Consumption (kWh)
Temperature (C°)

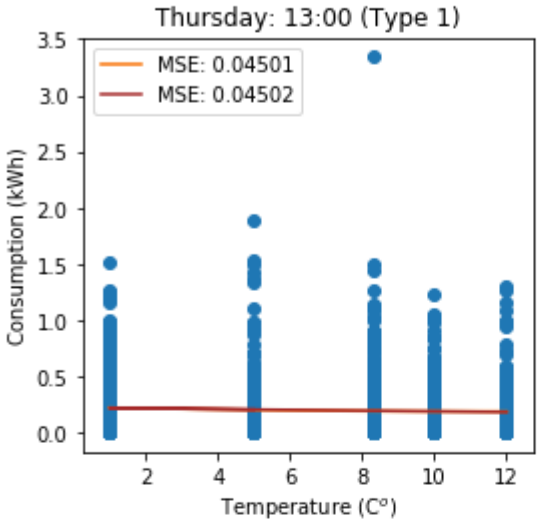

Consumption (kWh)
Temperature (C°)

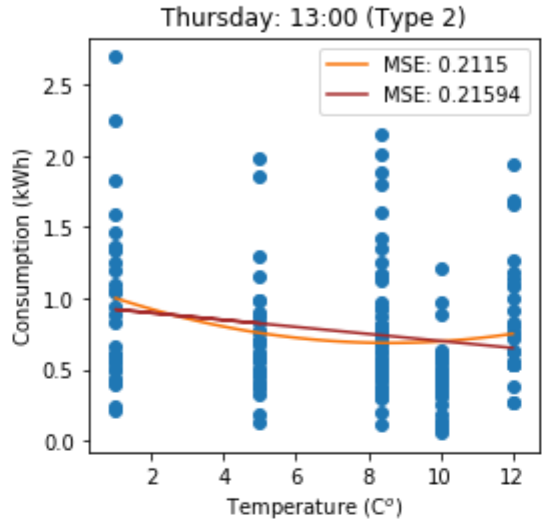

Consumption (kWh)
Temperature (C°)

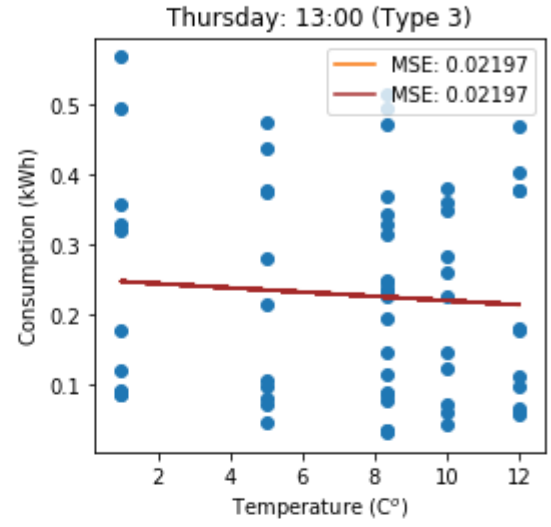

Thursday: 12:00 (Type 1)
MSE: 0.05346
MSE: 0.0535
Consumption (kWh)
Temperature (C°)

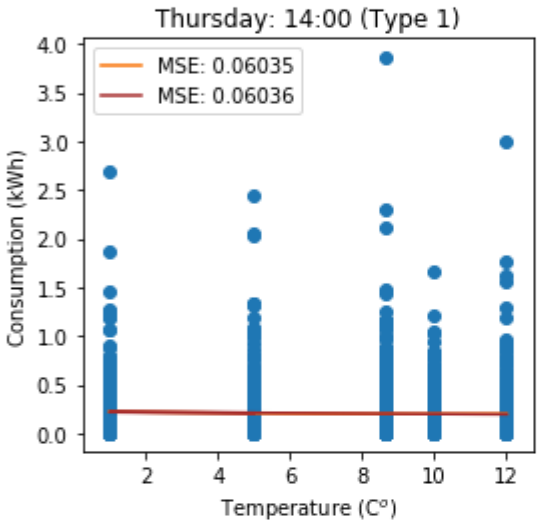

Thursday: 12:00 (Type 2)
MSE: 0.21923
MSE: 0.22698
Consumption (kWh)
Temperature (C°)

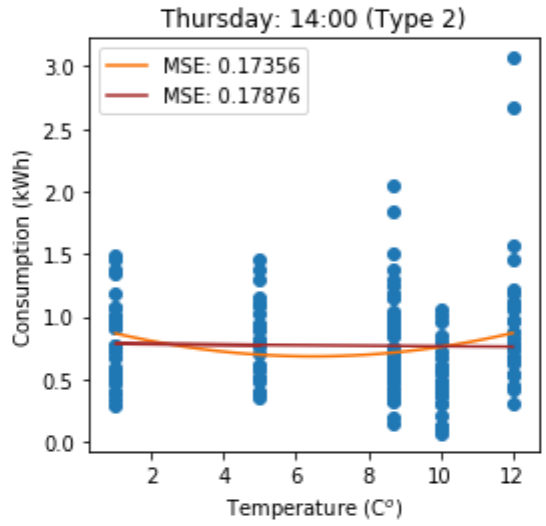

Thursday: 12:00 (Type 3)
MSE: 0.03645
MSE: 0.03645
Consumption (kWh)
Temperature (C°)

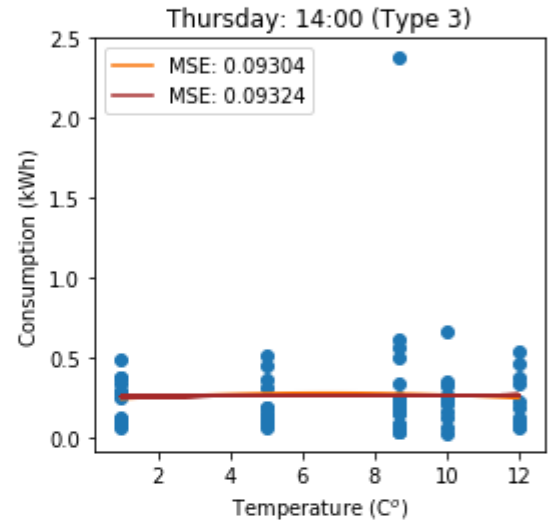

Thursday: 13:00 (Type 1)
MSE: 0.04501
MSE: 0.04502
Consumption (kWh)
Temperature (C°)

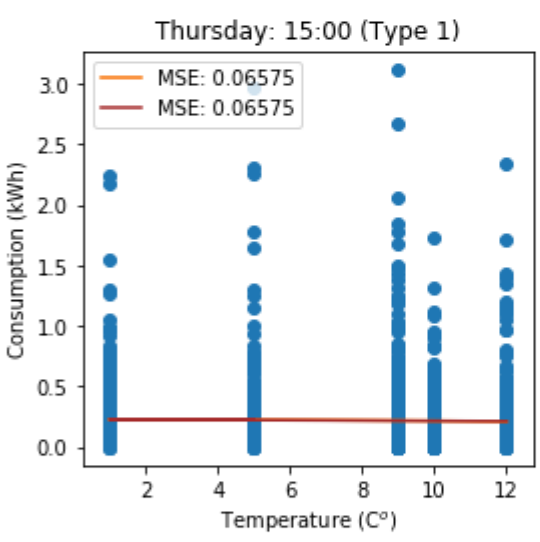

Thursday: 13:00 (Type 2)
MSE: 0.2115
MSE: 0.21594
Consumption (kWh)
Temperature (C°)

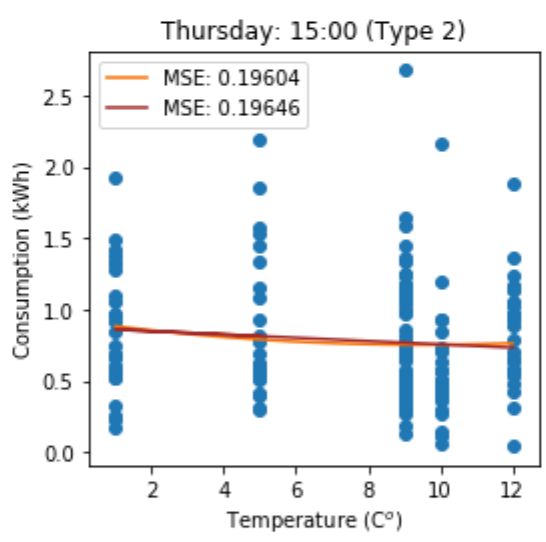

Thursday: 13:00 (Type 3)
MSE: 0.02197
MSE: 0.02197
Consumption (kWh)
Temperature (C°)

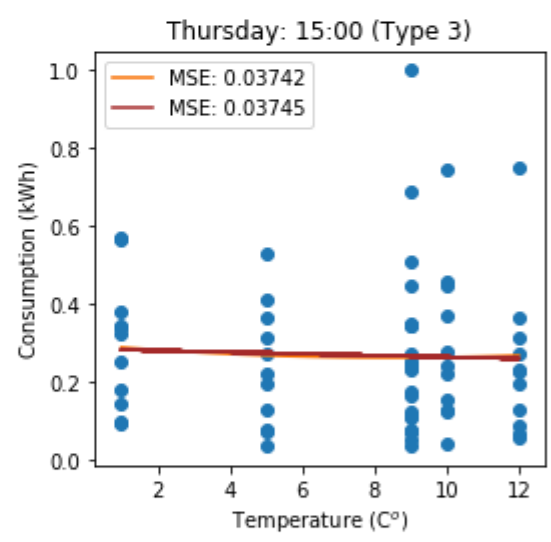

Thursday: 14:00 (Type 1)
MSE: 0.06035
MSE: 0.06036
Consumption (kWh)
Temperature (C°)

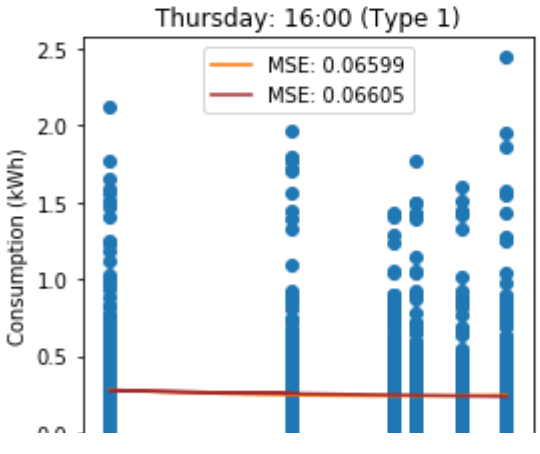

Thursday: 14:00 (Type 2)
MSE: 0.17356
MSE: 0.17876
Consumption (kWh)
Temperature (C°)

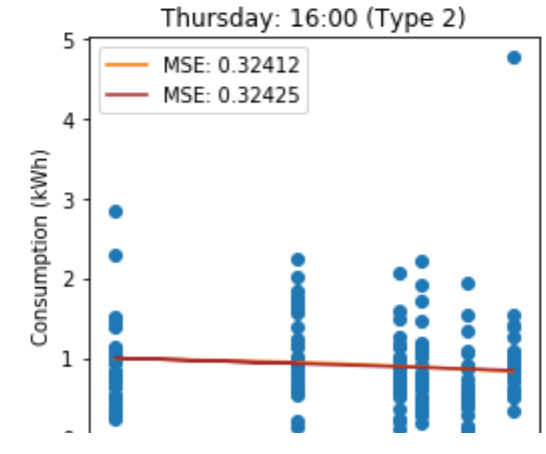

Thursday: 14:00 (Type 3)
MSE: 0.09304
MSE: 0.09324
Consumption (kWh)
Temperature (C°)

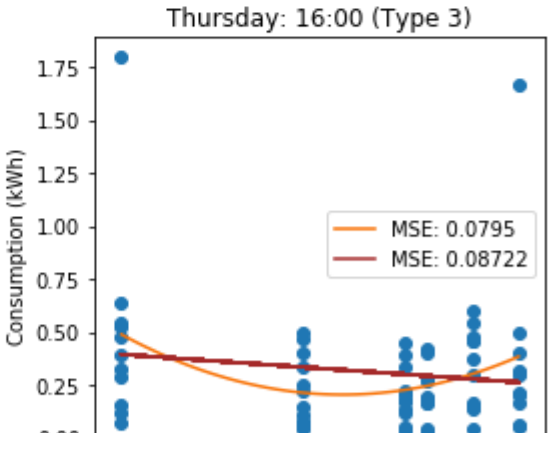

Thursday: 15:00 (Type 1)
MSE: 0.06575
MSE: 0.06575
Consumption (kWh)
Temperature (C°)

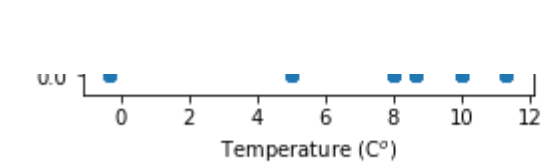

Thursday: 15:00 (Type 2)
MSE: 0.19604
MSE: 0.19646
Consumption (kWh)
Temperature (C°)

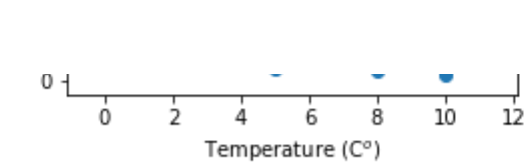

Thursday: 15:00 (Type 3)
MSE: 0.03742
MSE: 0.03745
Consumption (kWh)
Temperature (C°)

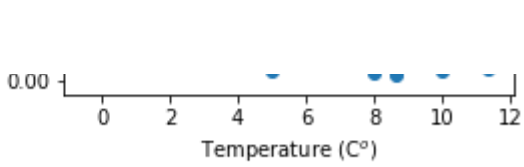

Thursday: 16:00 (Type 1)
MSE: 0.06599
MSE: 0.06605
Consumption (kWh)

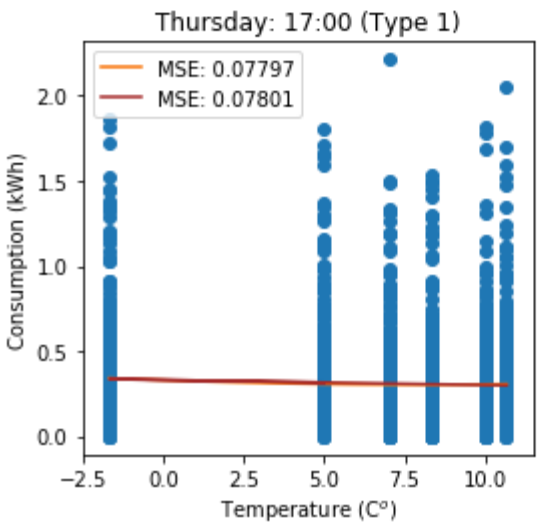

Thursday: 16:00 (Type 2)
MSE: 0.32412
MSE: 0.32425
Consumption (kWh)

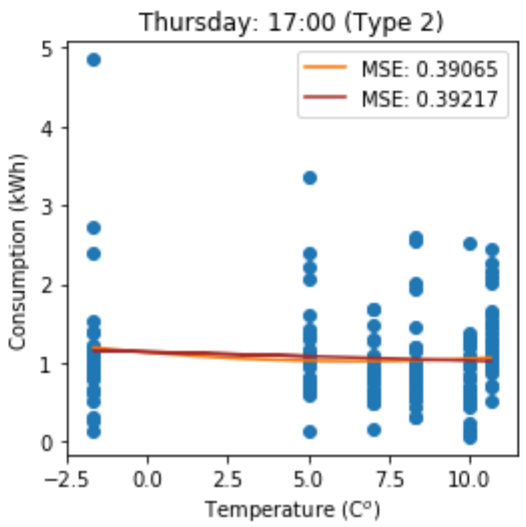

Thursday: 16:00 (Type 3)
MSE: 0.0795
MSE: 0.08722
Consumption (kWh)

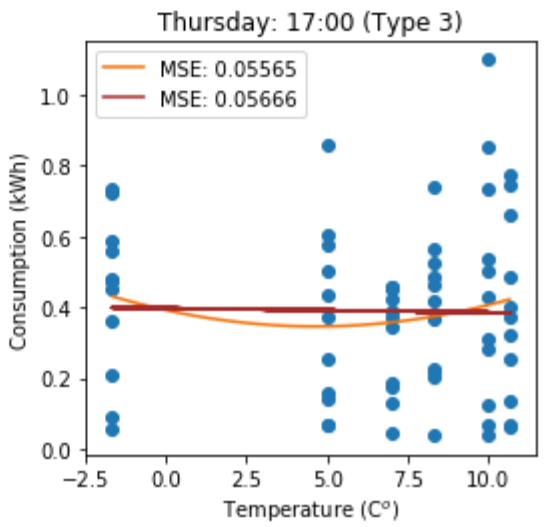

Temperature (C°)

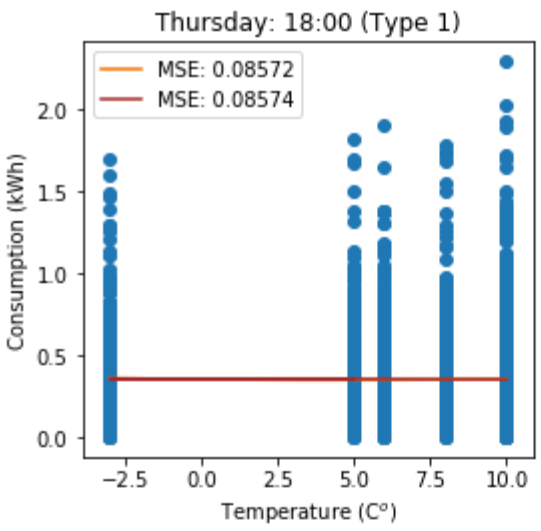

Temperature (C°)

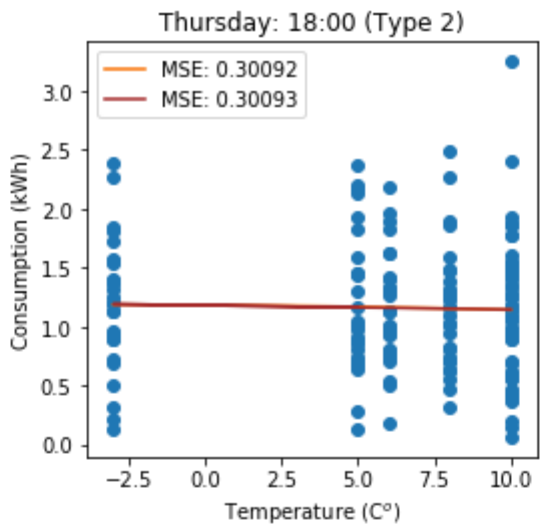

Temperature (C°)

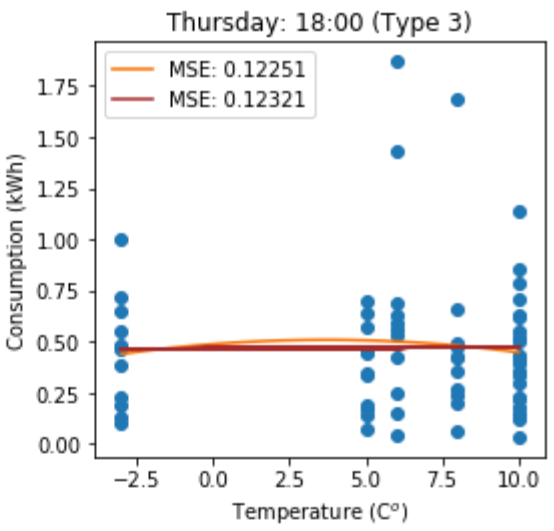

Thursday: 17:00 (Type 1)
MSE: 0.07797
MSE: 0.07801
Consumption (kWh)
Temperature (C°)

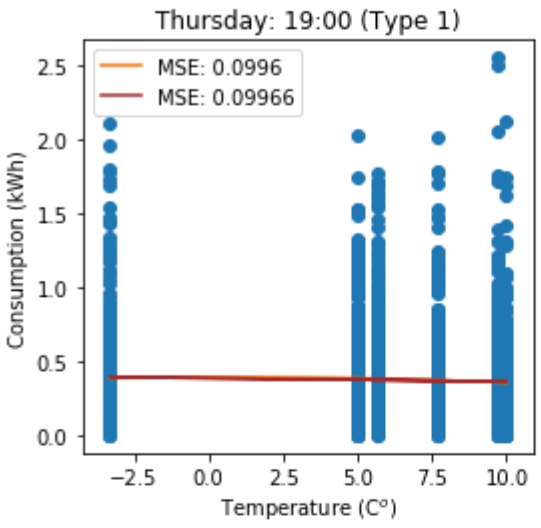

Thursday: 17:00 (Type 2)
MSE: 0.39065
MSE: 0.39217
Consumption (kWh)
Temperature (C°)

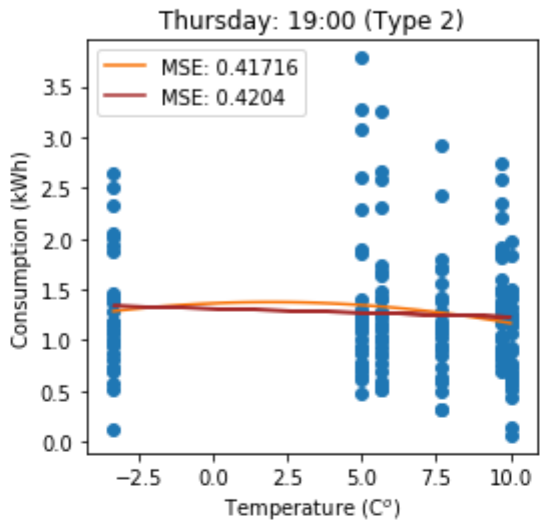

Thursday: 17:00 (Type 3)
MSE: 0.05565
MSE: 0.05666
Consumption (kWh)
Temperature (C°)

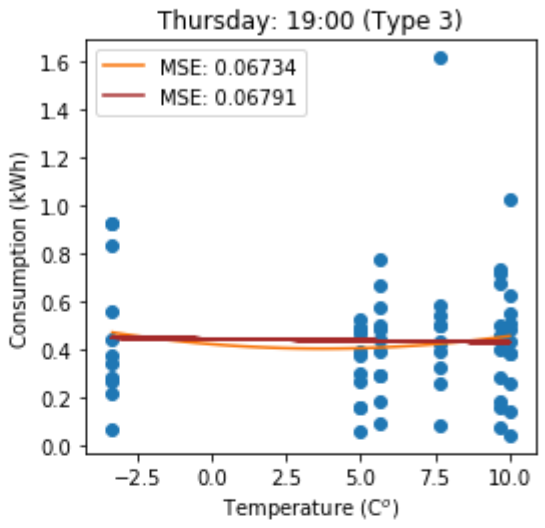

Thursday: 18:00 (Type 1)
MSE: 0.08572
MSE: 0.08574
Consumption (kWh)
Temperature (C°)

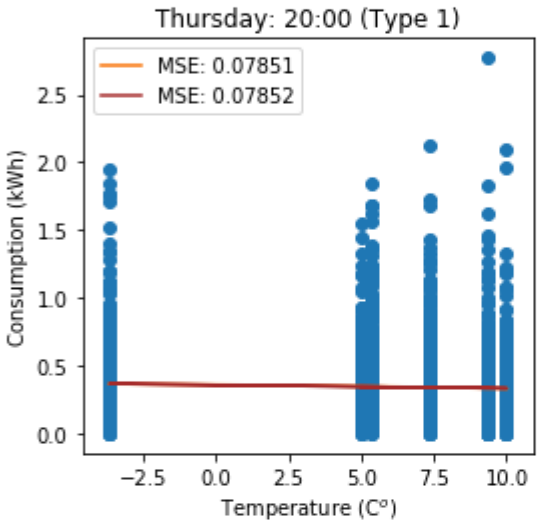

Thursday: 18:00 (Type 2)
MSE: 0.30092
MSE: 0.30093
Consumption (kWh)
Temperature (C°)

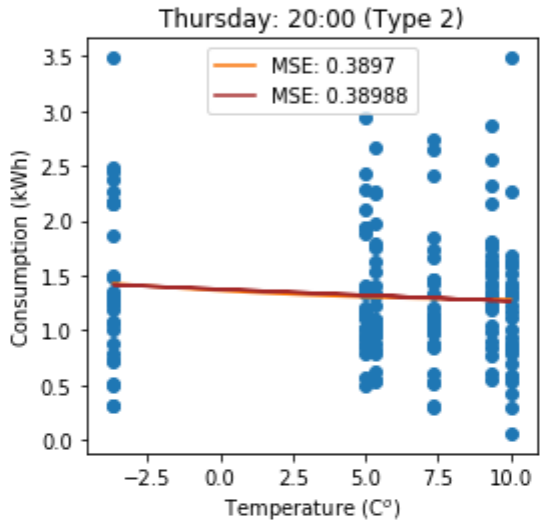

Thursday: 18:00 (Type 3)
MSE: 0.12251
MSE: 0.12321
Consumption (kWh)
Temperature (C°)

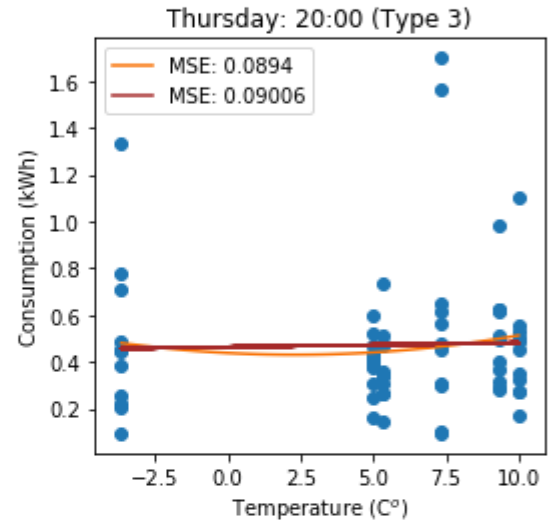

Thursday: 19:00 (Type 1)
MSE: 0.0996
MSE: 0.09966
Consumption (kWh)
Temperature (C°)

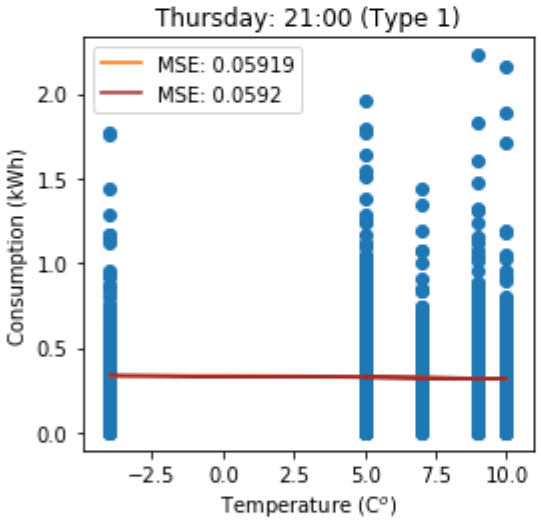

Thursday: 19:00 (Type 2)
MSE: 0.41716
MSE: 0.4204
Consumption (kWh)
Temperature (C°)

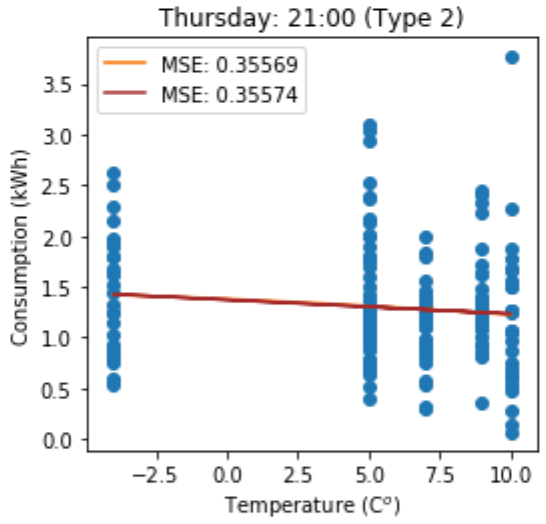

Thursday: 19:00 (Type 3)
MSE: 0.06734
MSE: 0.06791
Consumption (kWh)
Temperature (C°)

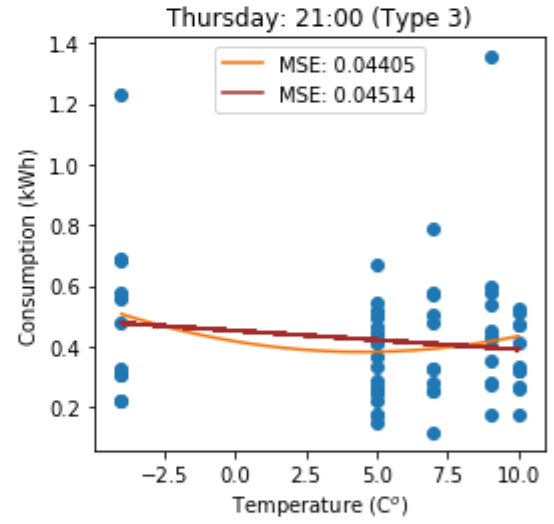

Thursday: 20:00 (Type 1)
MSE: 0.07851
MSE: 0.07852
Consumption (kWh)
Temperature (C°)

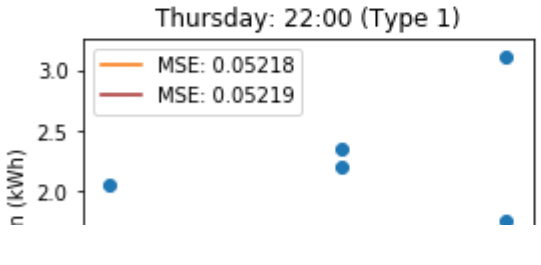

Thursday: 20:00 (Type 2)
MSE: 0.3897
MSE: 0.38988
Consumption (kWh)
Temperature (C°)

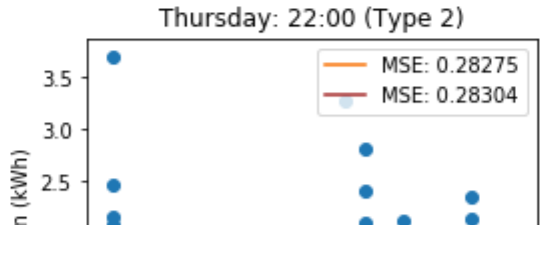

Thursday: 20:00 (Type 3)
MSE: 0.0894
MSE: 0.09006
Consumption (kWh)
Temperature (C°)

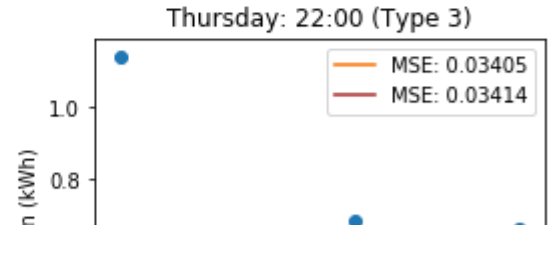

Thursday: 21:00 (Type 1)
MSE: 0.05919
MSE: 0.0592
Consumption (kWh)
Temperature (C°)

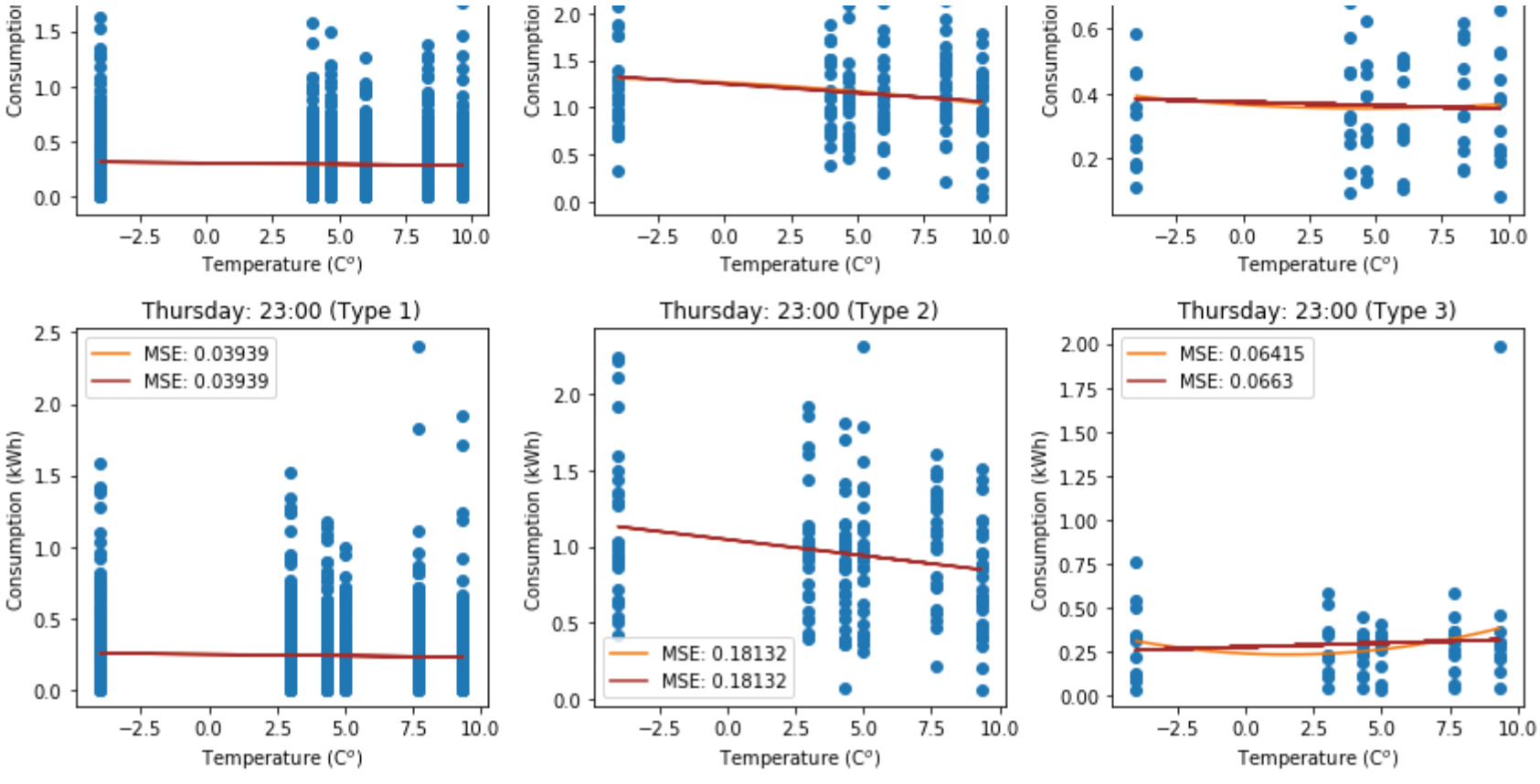

Thursday: 21:00 (Type 2)
MSE: 0.35569
MSE: 0.35574
Consumption (kWh)
Temperature (C°)


Thursday: 21:00 (Type 3)
MSE: 0.04405
MSE: 0.04514
Consumption (kWh)
Temperature (C°)

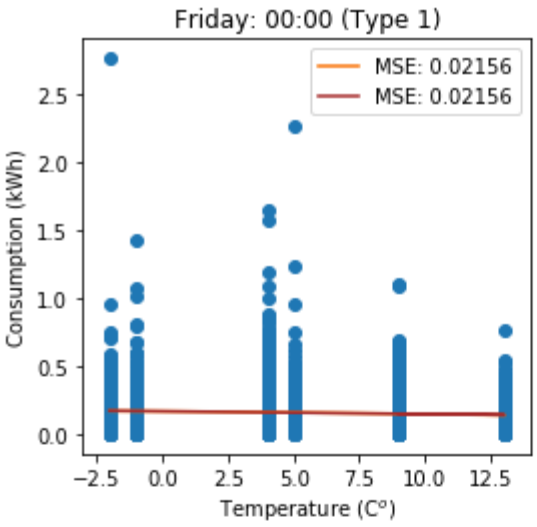

Thursday: 22:00 (Type 1)
MSE: 0.05218
MSE: 0.05219

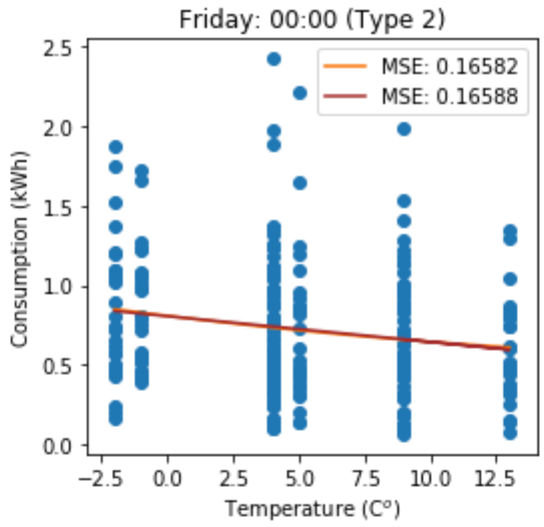

Thursday: 22:00 (Type 2)
MSE: 0.28275
MSE: 0.28304

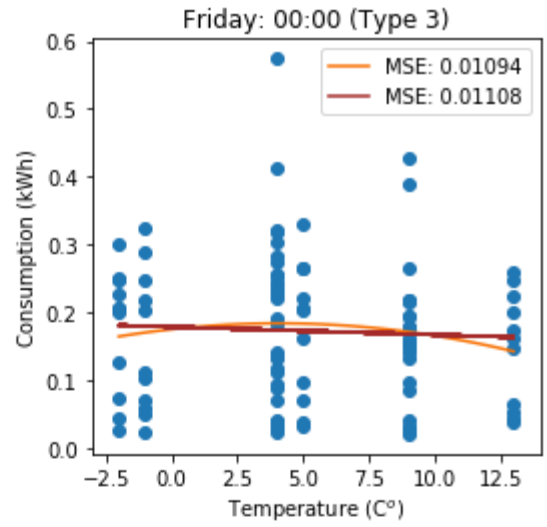

Thursday: 22:00 (Type 3)
MSE: 0.03405
MSE: 0.03414

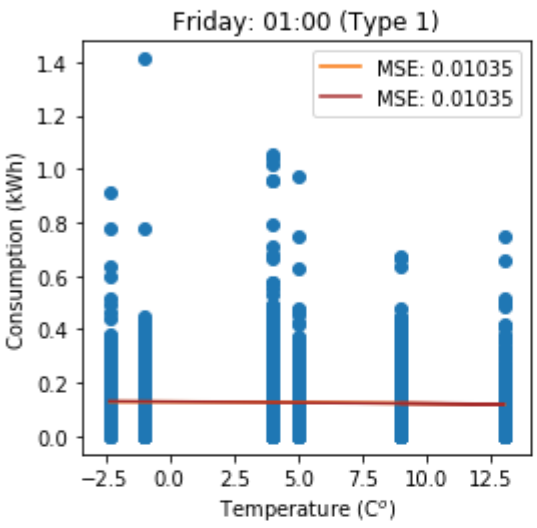

Thursday: 23:00 (Type 1)
Thursday: 23:00 (Type 2)
Thursday: 23:00 (Type 3)
Consumption (kWh)
Temperature (C°)
MSE: 0.03939
MSE: 0.03939
MSE: 0.18132
MSE: 0.18132
MSE: 0.06415
MSE: 0.0663

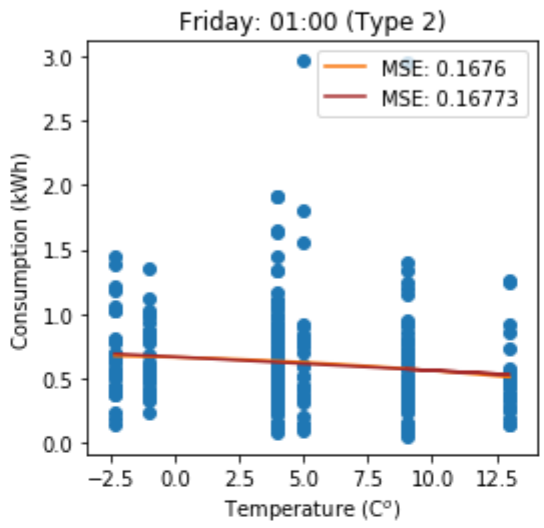
In [12]:
```
# Friday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 4 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

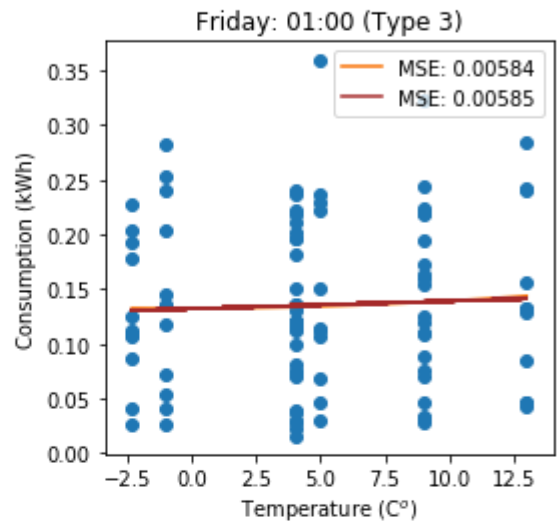

Friday: 00:00 (Type 1)
MSE: 0.02156
MSE: 0.02156
Consumption (kWh)
Temperature (C°)

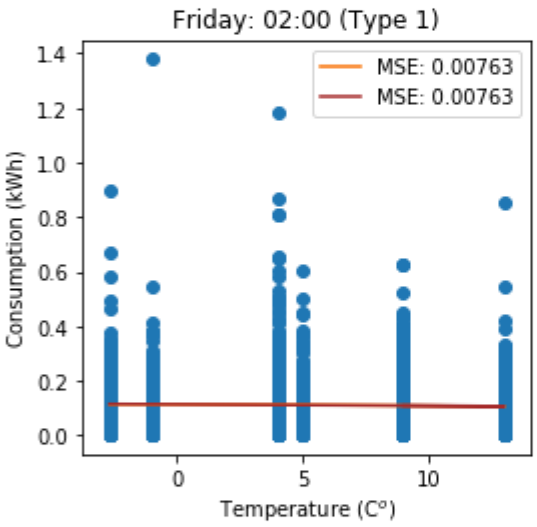

Friday: 00:00 (Type 2)
MSE: 0.16582
MSE: 0.16588
Consumption (kWh)
Temperature (C°)

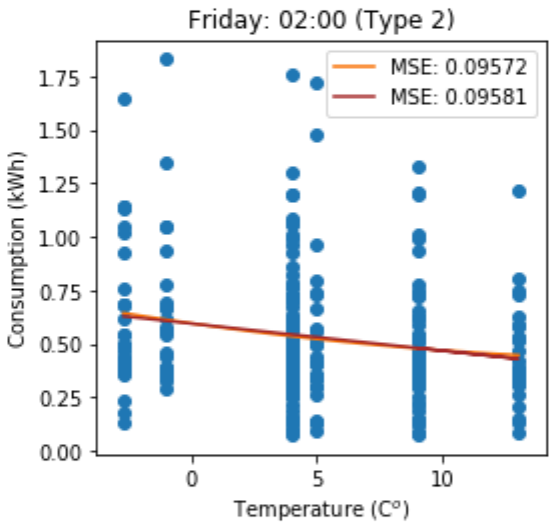

Friday: 00:00 (Type 3)
MSE: 0.01094
MSE: 0.01108
Consumption (kWh)
Temperature (C°)

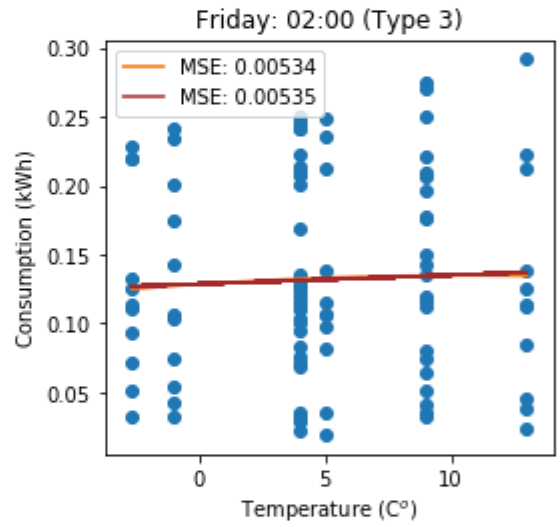

Friday: 01:00 (Type 1)
MSE: 0.01035
MSE: 0.01035
Consumption (kWh)
Temperature (C°)

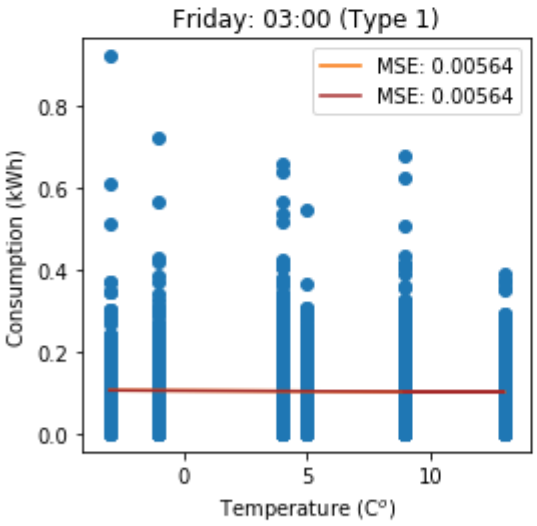

Friday: 01:00 (Type 2)
MSE: 0.1676
MSE: 0.16773
Consumption (kWh)
Temperature (C°)

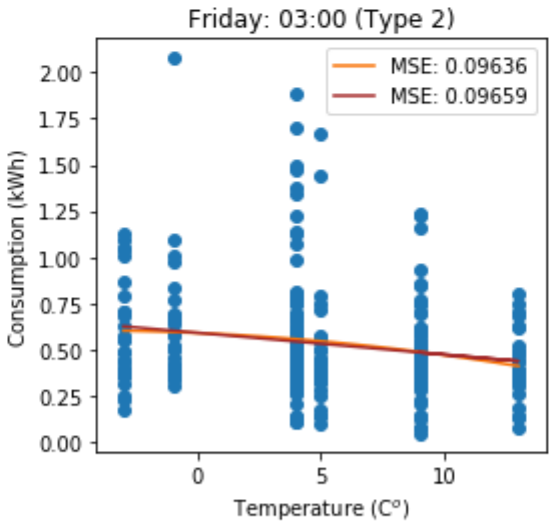

Friday: 01:00 (Type 3)
MSE: 0.00584
MSE: 0.00585
Consumption (kWh)
Temperature (C°)

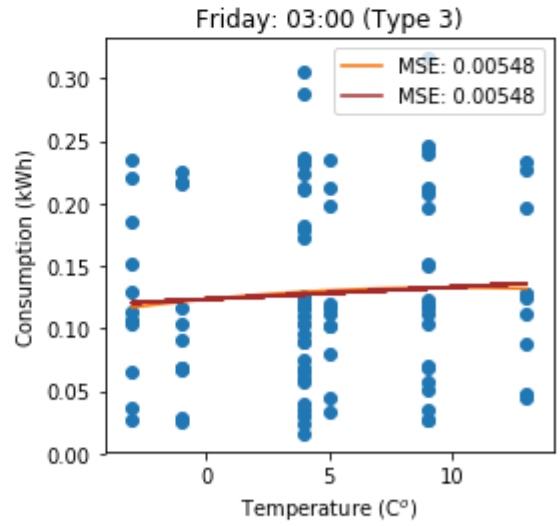

Friday: 02:00 (Type 1)
MSE: 0.00763
MSE: 0.00763
Consumption (kWh)
Temperature (C°)

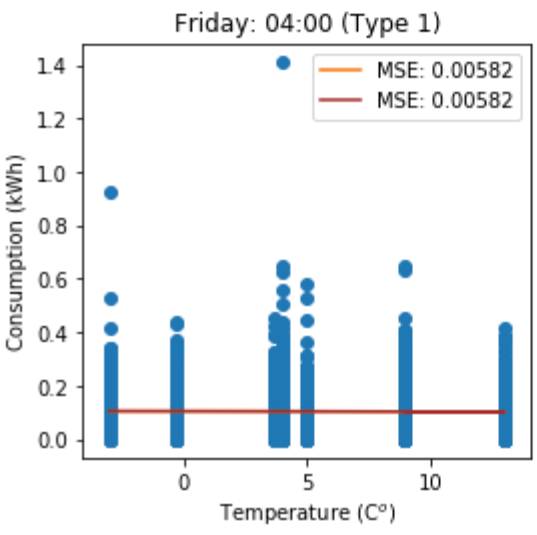

Friday: 02:00 (Type 2)
MSE: 0.09572
MSE: 0.09581
Consumption (kWh)
Temperature (C°)

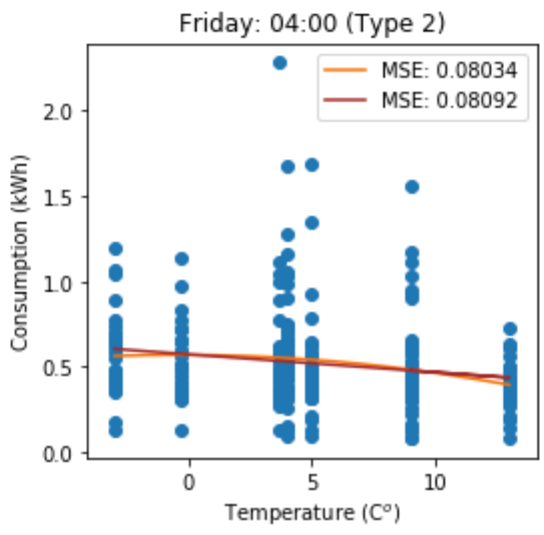

Friday: 02:00 (Type 3)
MSE: 0.00534
MSE: 0.00535
Consumption (kWh)
Temperature (C°)

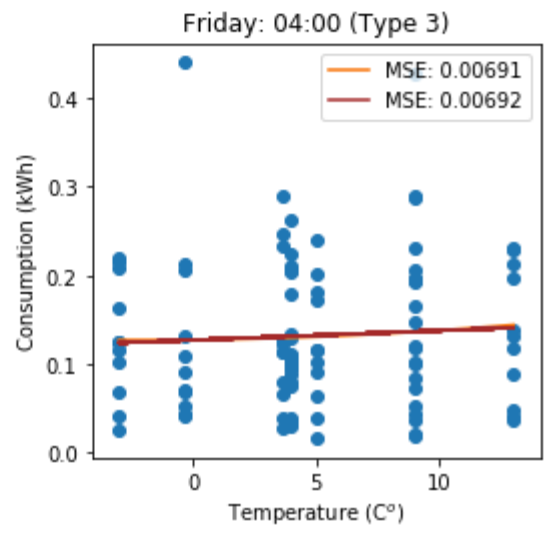

Friday: 03:00 (Type 1)
MSE: 0.00564
MSE: 0.00564
Consumption (kWh)
Temperature (C°)

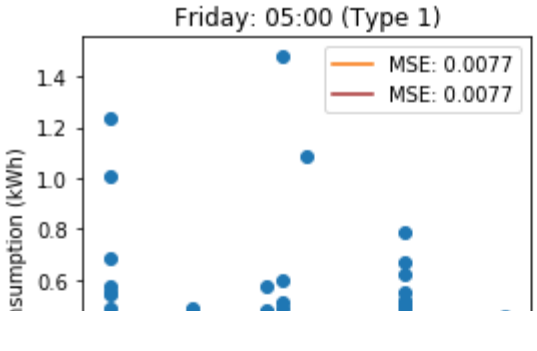

Friday: 03:00 (Type 2)
MSE: 0.09636
MSE: 0.09659
Consumption (kWh)
Temperature (C°)

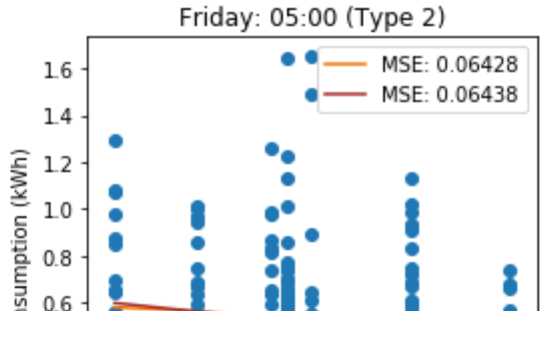

Friday: 03:00 (Type 3)
MSE: 0.00548
MSE: 0.00548
Consumption (kWh)
Temperature (C°)

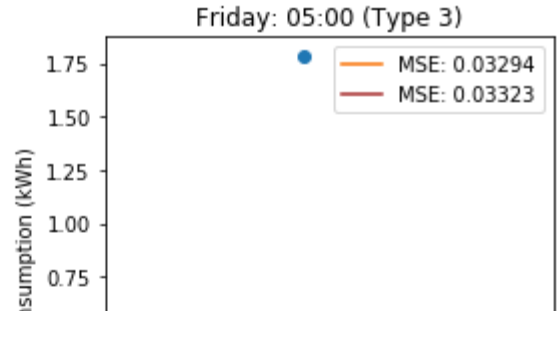

Friday: 04:00 (Type 1)
MSE: 0.00582
MSE: 0.00582
Consumption (kWh)
Temperature (C°)

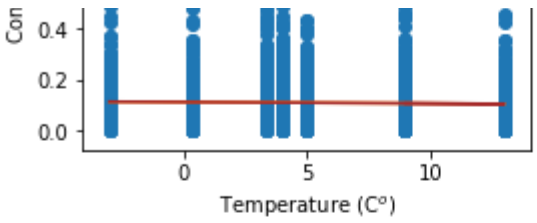

Friday: 04:00 (Type 2)
MSE: 0.08034
MSE: 0.08092
Consumption (kWh)
Temperature (C°)

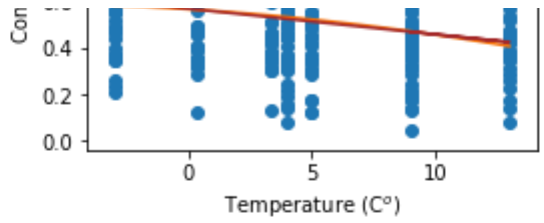

Friday: 04:00 (Type 3)
MSE: 0.00691
MSE: 0.00692
Consumption (kWh)
Temperature (C°)

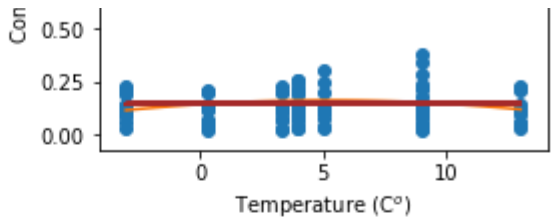

Friday: 05:00 (Type 1)
MSE: 0.0077
MSE: 0.0077

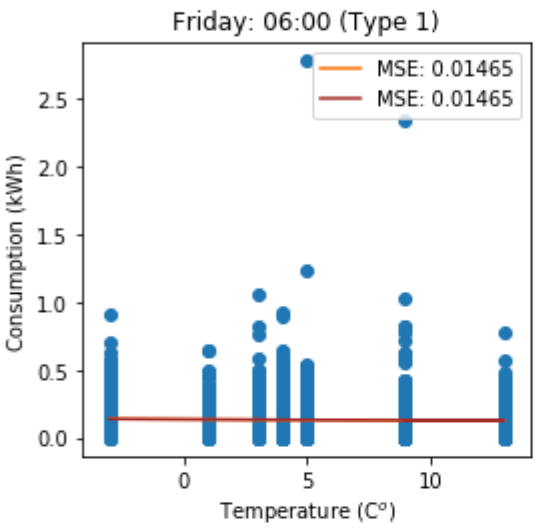

Friday: 05:00 (Type 2)
MSE: 0.06428
MSE: 0.06438

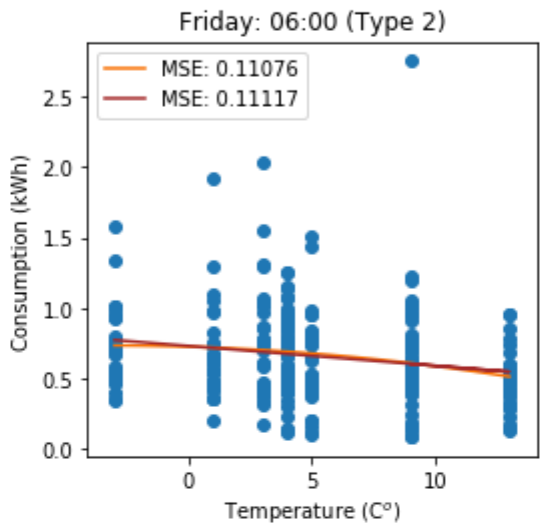

Friday: 05:00 (Type 3)
MSE: 0.03294
MSE: 0.03323

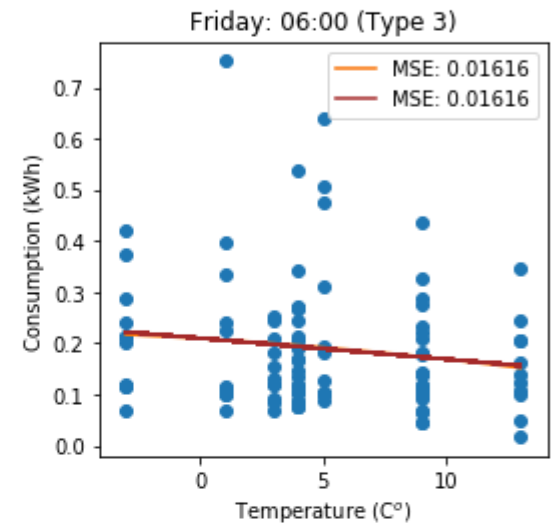

Con
0.4
0.2
0.0
0
5
10
Temperature (C°)

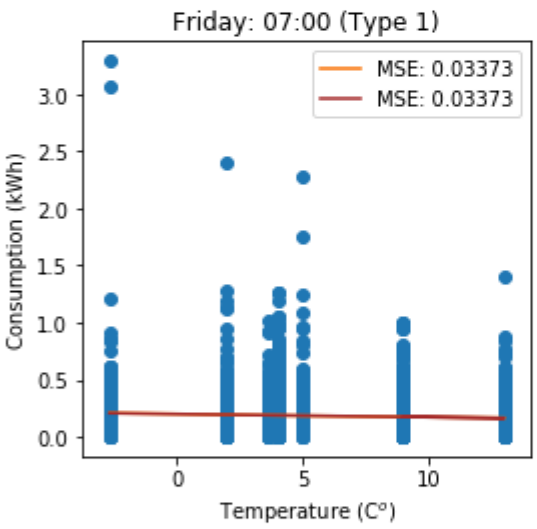

Con
0.4
0.2
0.0
0
5
10
Temperature (C°)

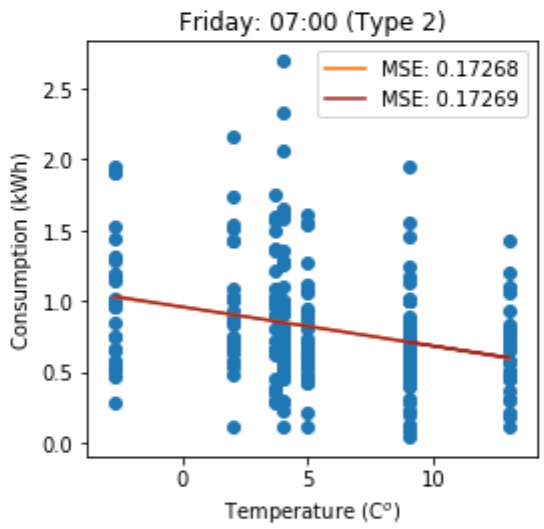

Con
0.50
0.25
0.00
0
5
10
Temperature (C°)

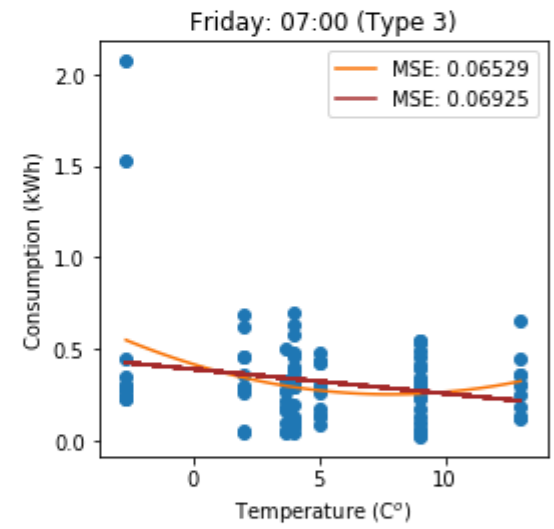

Friday: 06:00 (Type 1)
MSE: 0.01465
MSE: 0.01465
Consumption (kWh)
Temperature (C°)

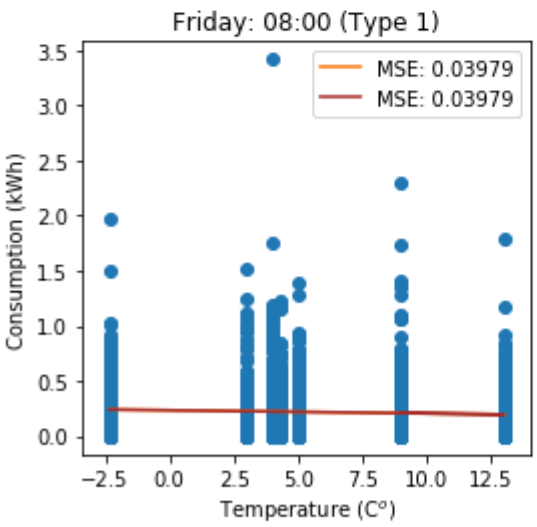

Friday: 06:00 (Type 2)
MSE: 0.11076
MSE: 0.11117
Consumption (kWh)
Temperature (C°)

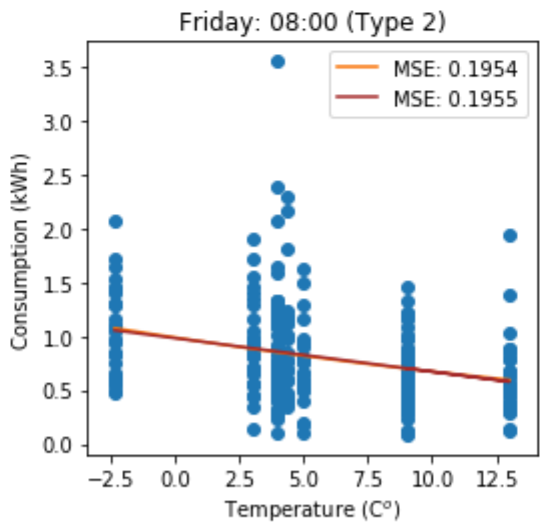

Friday: 06:00 (Type 3)
MSE: 0.01616
MSE: 0.01616
Consumption (kWh)
Temperature (C°)

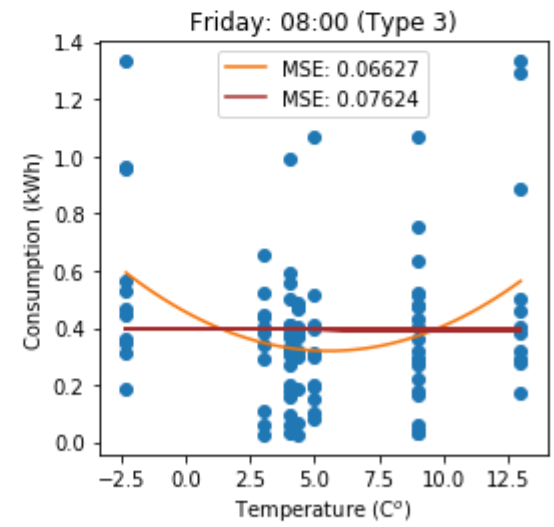

Friday: 07:00 (Type 1)
MSE: 0.03373
MSE: 0.03373
Consumption (kWh)
Temperature (C°)

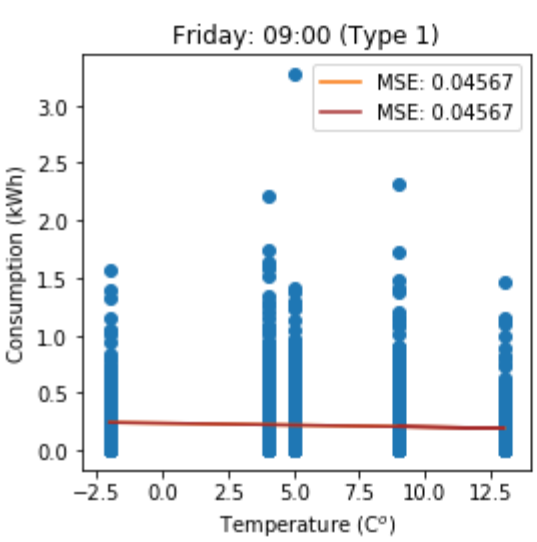

Friday: 07:00 (Type 2)
MSE: 0.17268
MSE: 0.17269
Consumption (kWh)
Temperature (C°)

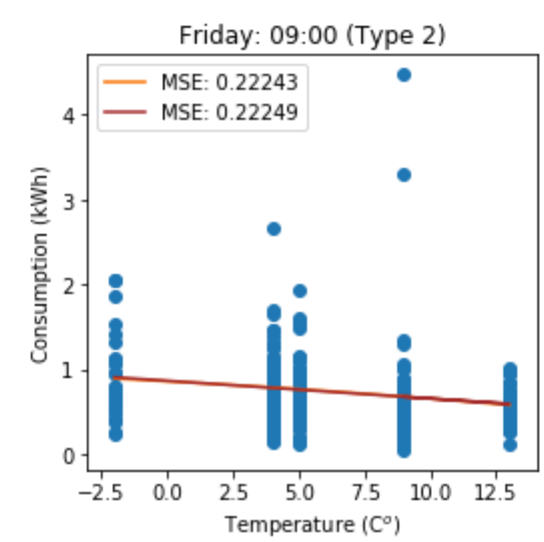

Friday: 07:00 (Type 3)
MSE: 0.06529
MSE: 0.06925
Consumption (kWh)
Temperature (C°)

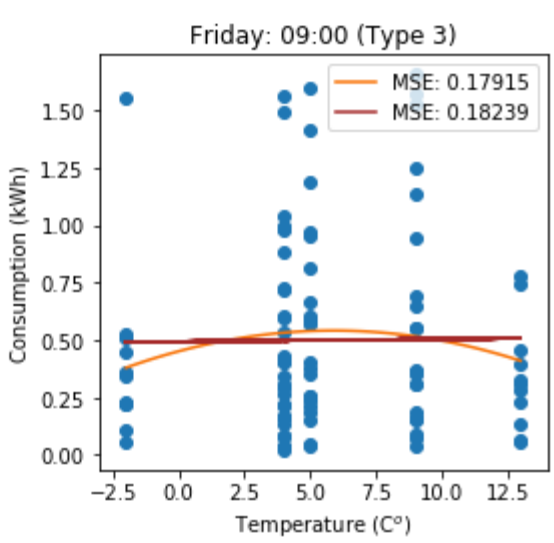

Friday: 08:00 (Type 1)
MSE: 0.03979
MSE: 0.03979
Consumption (kWh)
Temperature (C°)

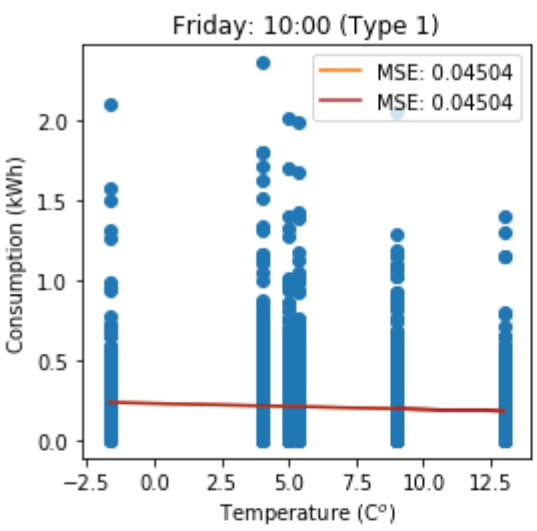

Friday: 08:00 (Type 2)
MSE: 0.1954
MSE: 0.1955
Consumption (kWh)
Temperature (C°)

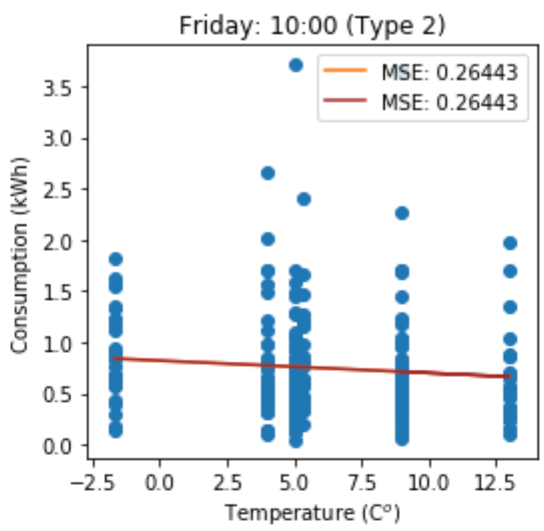

Friday: 08:00 (Type 3)
MSE: 0.06627
MSE: 0.07624
Consumption (kWh)
Temperature (C°)

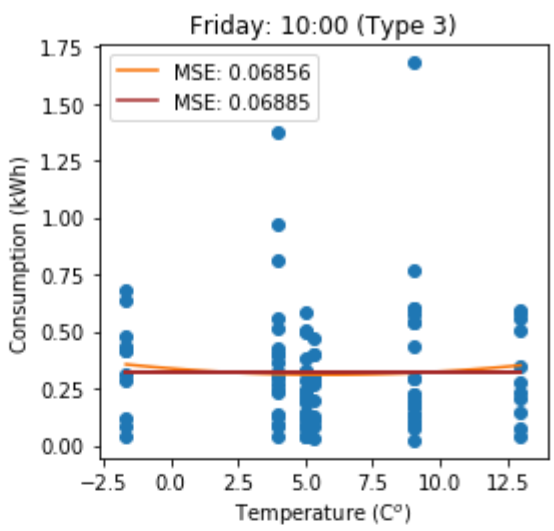

Friday: 09:00 (Type 1)
MSE: 0.04567
MSE: 0.04567
Consumption (kWh)
Temperature (C°)

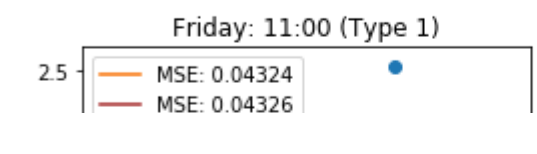

Friday: 09:00 (Type 2)
MSE: 0.22243
MSE: 0.22249
Consumption (kWh)
Temperature (C°)


Friday: 09:00 (Type 3)
MSE: 0.17915
MSE: 0.18239
Consumption (kWh)
Temperature (C°)


Friday: 10:00 (Type 1)
MSE: 0.04504
MSE: 0.04504
Consumption (kWh)
Temperature (C°)

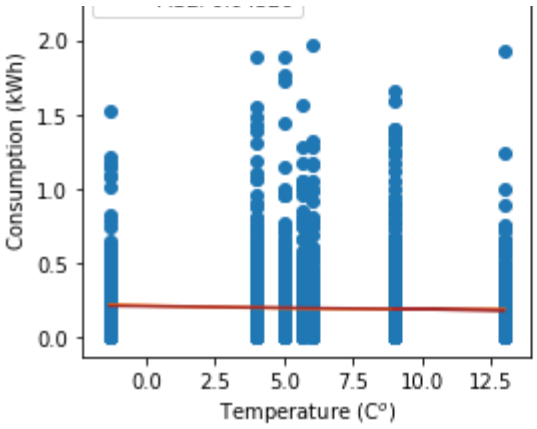

Friday: 10:00 (Type 2)
MSE: 0.26443
MSE: 0.26443
Consumption (kWh)
Temperature (C°)

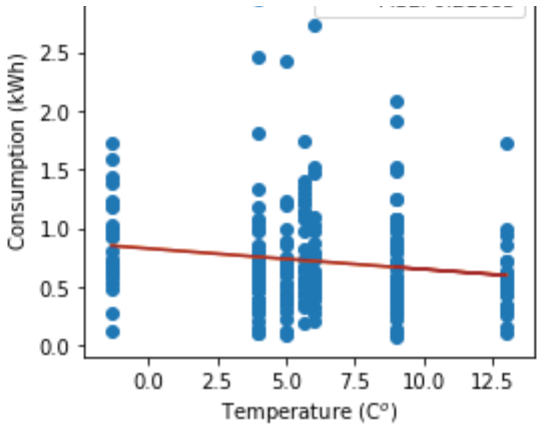

Friday: 10:00 (Type 3)
MSE: 0.06856
MSE: 0.06885
Consumption (kWh)
Temperature (C°)

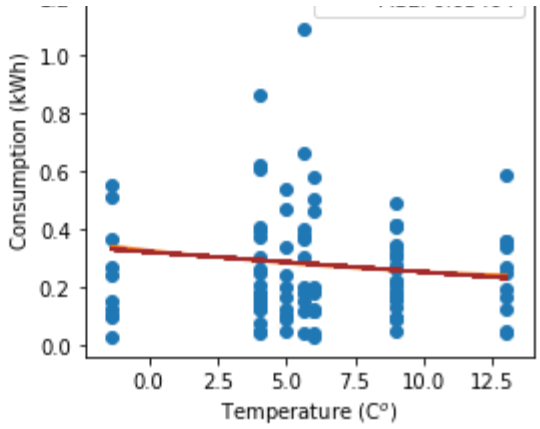

Friday: 11:00 (Type 1)
MSE: 0.04324
MSE: 0.04326

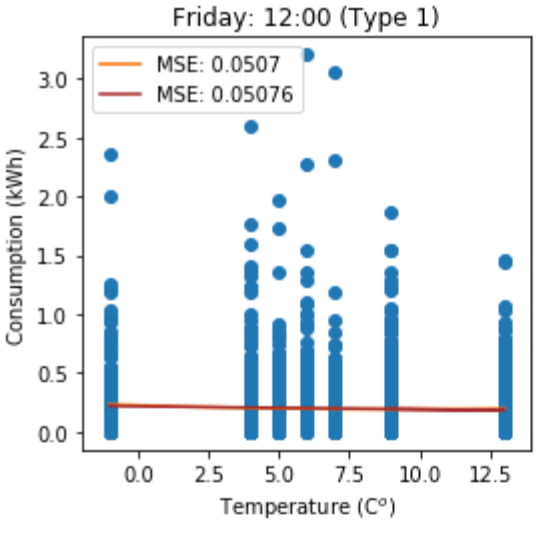

Friday: 11:00 (Type 2)
MSE: 0.21885
MSE: 0.21885

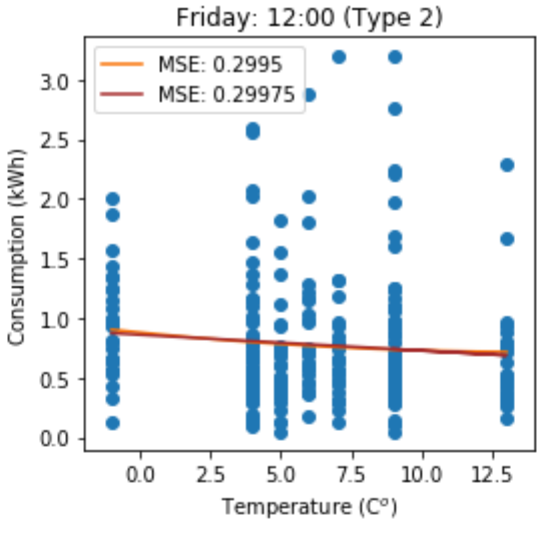

Friday: 11:00 (Type 3)
MSE: 0.054
MSE: 0.05404

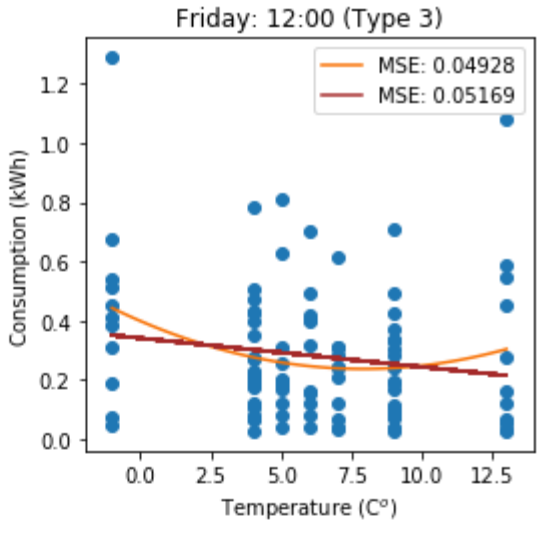

Consumption (kWh)
Temperature (C°)

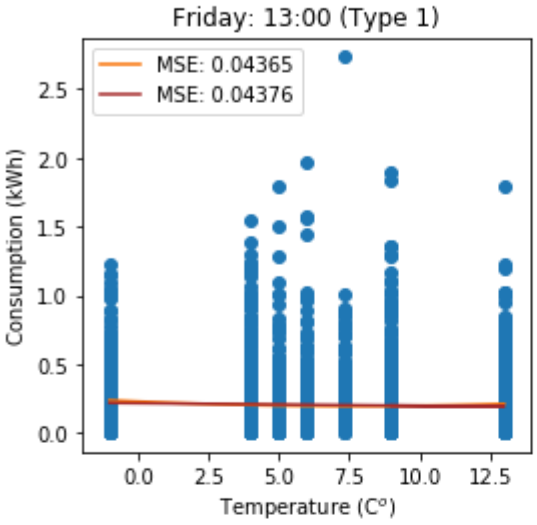

Consumption (kWh)
Temperature (C°)

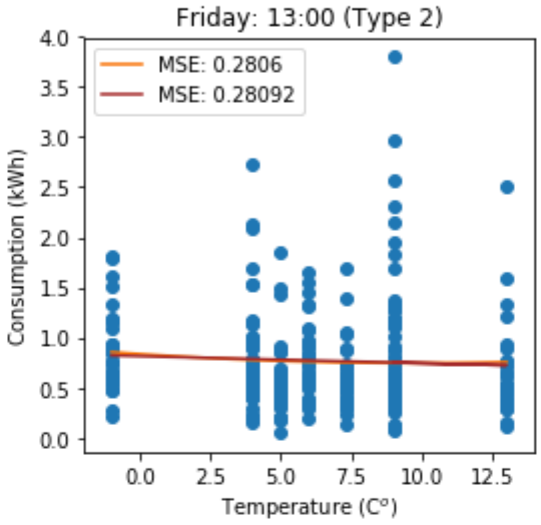

Consumption (kWh)
Temperature (C°)

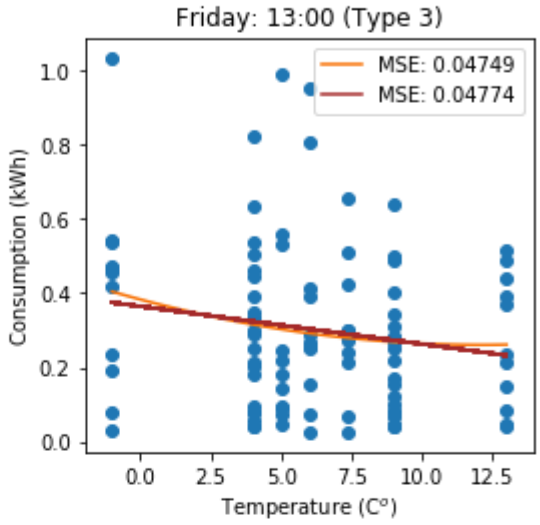

Friday: 12:00 (Type 1)
MSE: 0.0507
MSE: 0.05076
Consumption (kWh)
Temperature (C°)

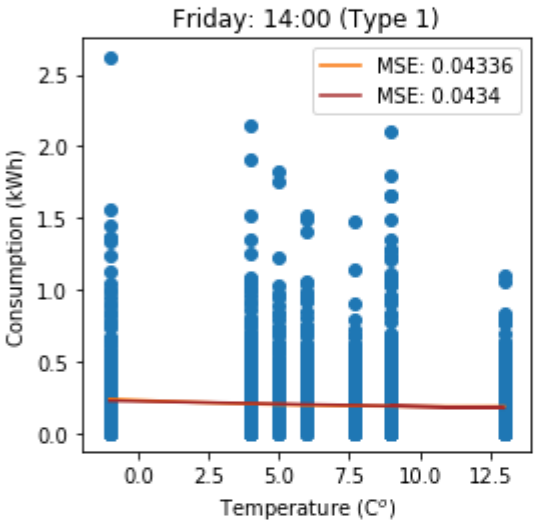

Friday: 12:00 (Type 2)
MSE: 0.2995
MSE: 0.29975
Consumption (kWh)
Temperature (C°)

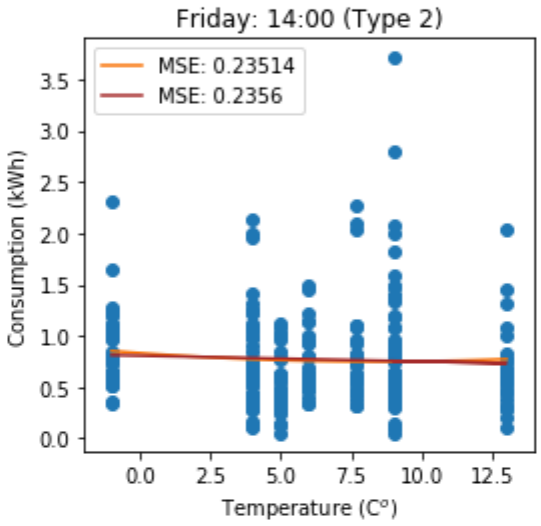

Friday: 12:00 (Type 3)
MSE: 0.04928
MSE: 0.05169
Consumption (kWh)
Temperature (C°)

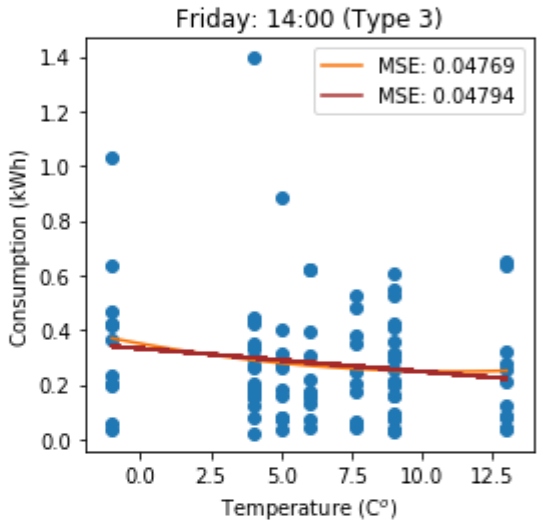

Friday: 13:00 (Type 1)
MSE: 0.04365
MSE: 0.04376
Consumption (kWh)
Temperature (C°)

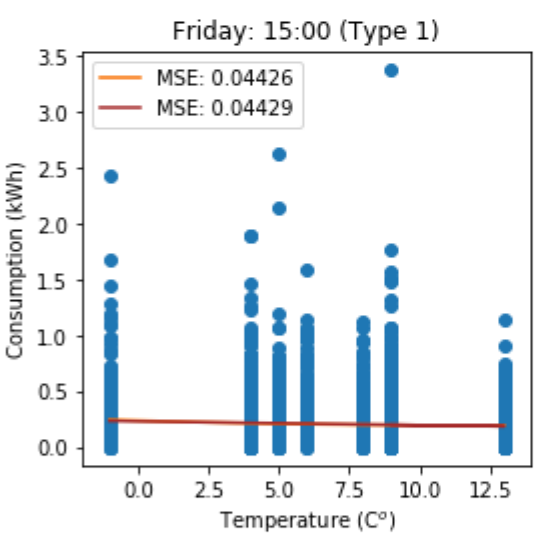

Friday: 13:00 (Type 2)
MSE: 0.2806
MSE: 0.28092
Consumption (kWh)
Temperature (C°)

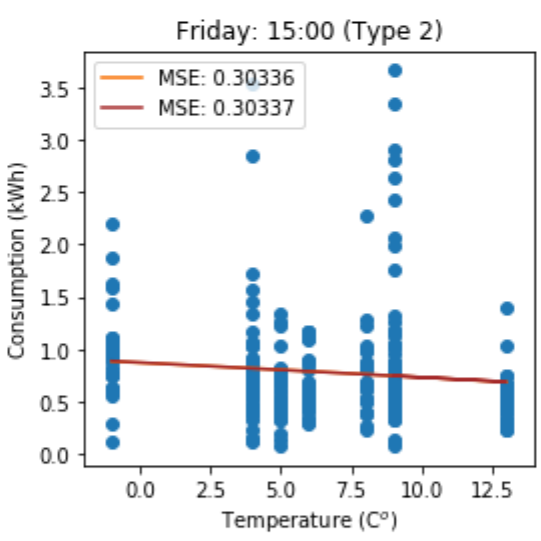

Friday: 13:00 (Type 3)
MSE: 0.04749
MSE: 0.04774
Consumption (kWh)
Temperature (C°)

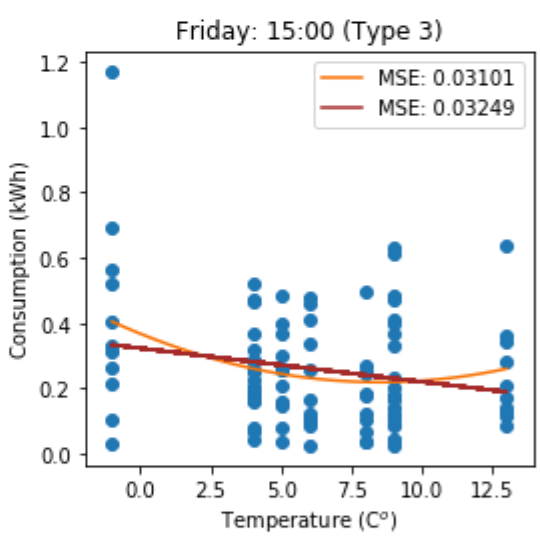

Friday: 14:00 (Type 1)
MSE: 0.04336
MSE: 0.0434
Consumption (kWh)
Temperature (C°)

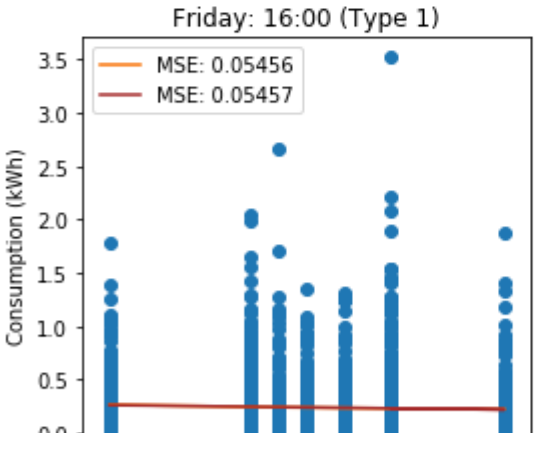

Friday: 14:00 (Type 2)
MSE: 0.23514
MSE: 0.2356
Consumption (kWh)
Temperature (C°)

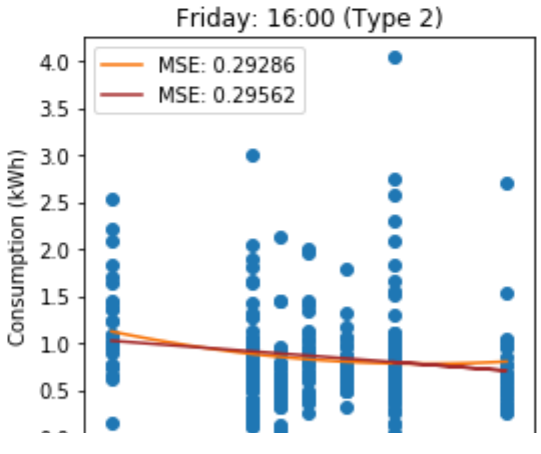

Friday: 14:00 (Type 3)
MSE: 0.04769
MSE: 0.04794
Consumption (kWh)
Temperature (C°)

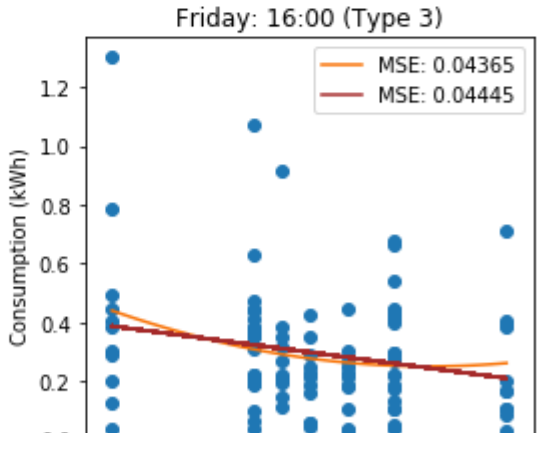

Friday: 15:00 (Type 1)
MSE: 0.04426
MSE: 0.04429
Consumption (kWh)
Temperature (C°)

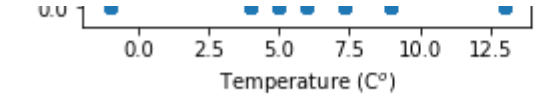

Friday: 15:00 (Type 2)
MSE: 0.30336
MSE: 0.30337
Consumption (kWh)
Temperature (C°)

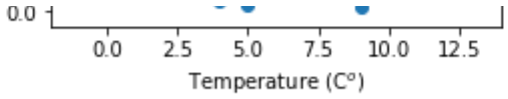

Friday: 15:00 (Type 3)
MSE: 0.03101
MSE: 0.03249
Consumption (kWh)
Temperature (C°)

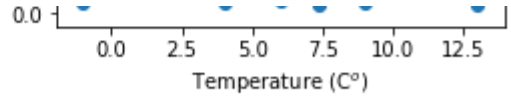

Friday: 16:00 (Type 1)
MSE: 0.05456
MSE: 0.05457
Consumption (kWh)

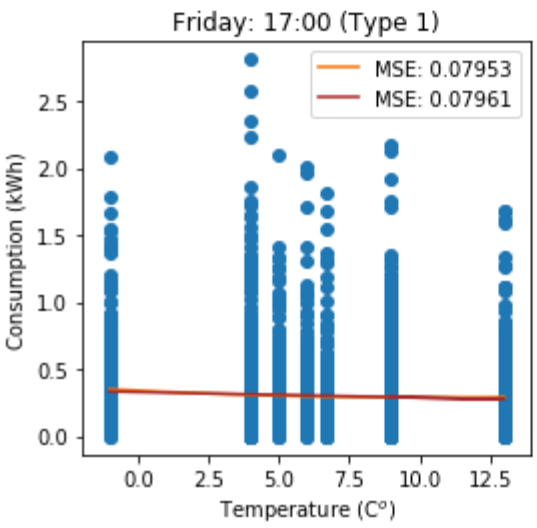

Friday: 16:00 (Type 2)
MSE: 0.29286
MSE: 0.29562
Consumption (kWh)

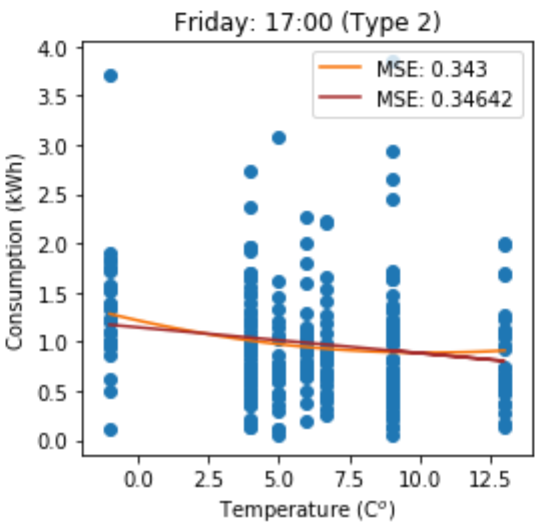

Friday: 16:00 (Type 3)
MSE: 0.04365
MSE: 0.04445
Consumption (kWh)

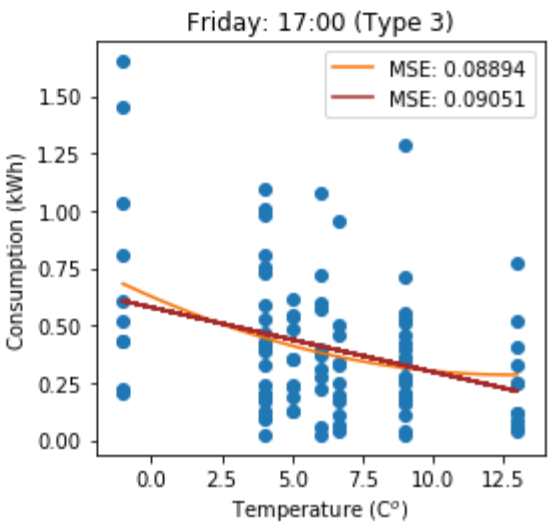

0.0
0.0 2.5 5.0 7.5 10.0 12.5
Temperature (C°)

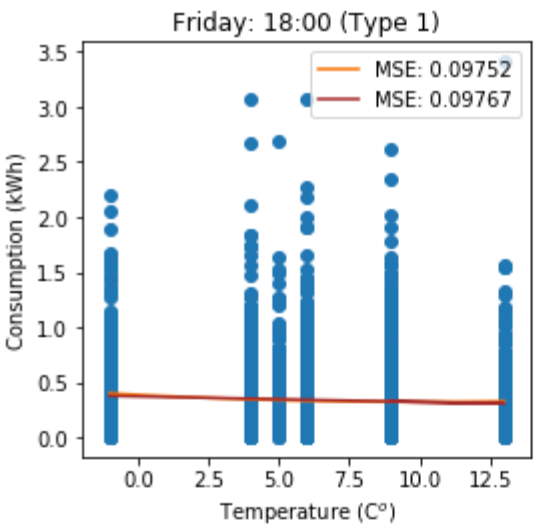

0.0
0.0 2.5 5.0 7.5 10.0 12.5
Temperature (C°)

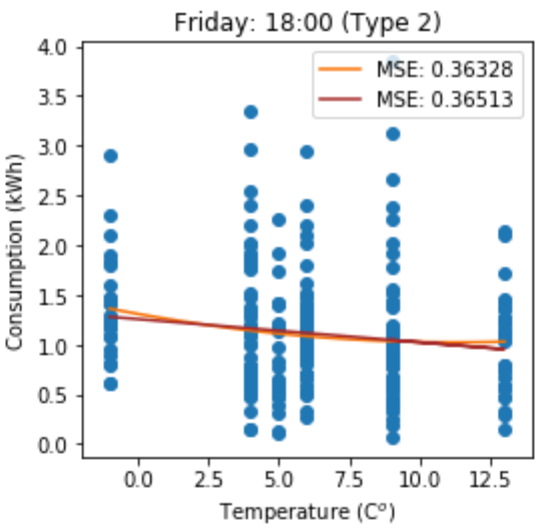

0.0
0.0 2.5 5.0 7.5 10.0 12.5
Temperature (C°)

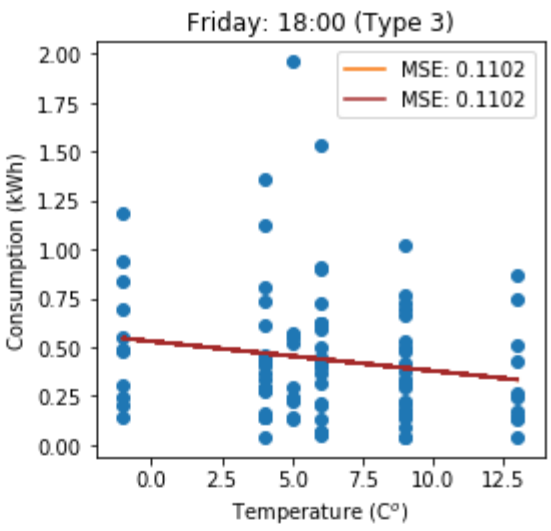

Friday: 17:00 (Type 1)
MSE: 0.07953
MSE: 0.07961
Consumption (kWh)
Temperature (C°)

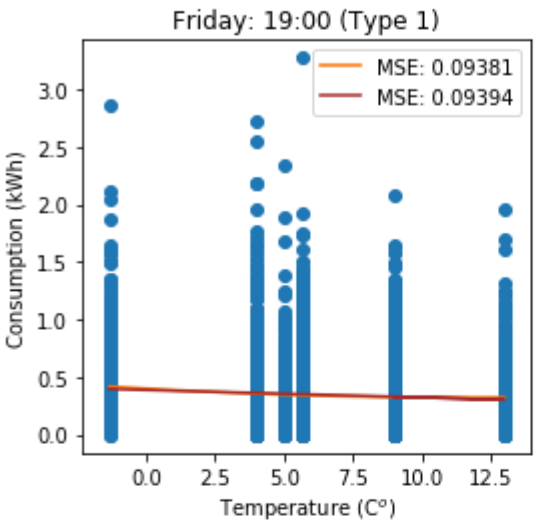

Friday: 17:00 (Type 2)
MSE: 0.343
MSE: 0.34642
Consumption (kWh)
Temperature (C°)

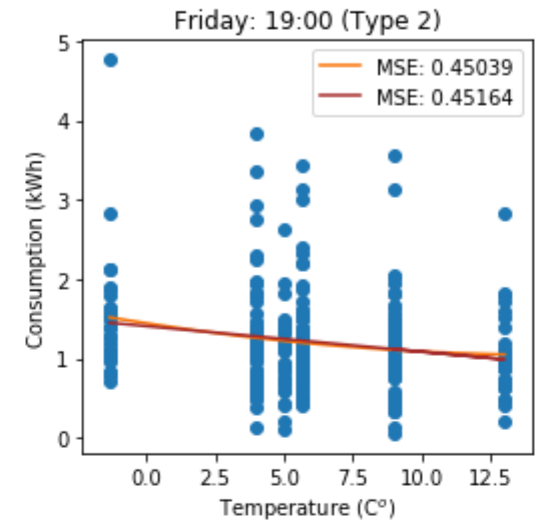

Friday: 17:00 (Type 3)
MSE: 0.08894
MSE: 0.09051
Consumption (kWh)
Temperature (C°)

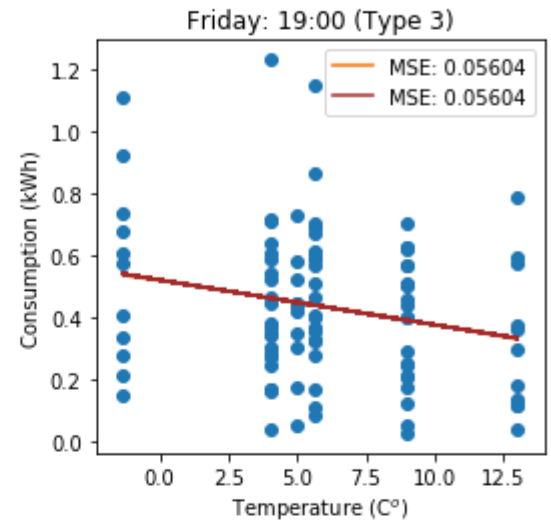

Friday: 18:00 (Type 1)
MSE: 0.09752
MSE: 0.09767
Consumption (kWh)
Temperature (C°)

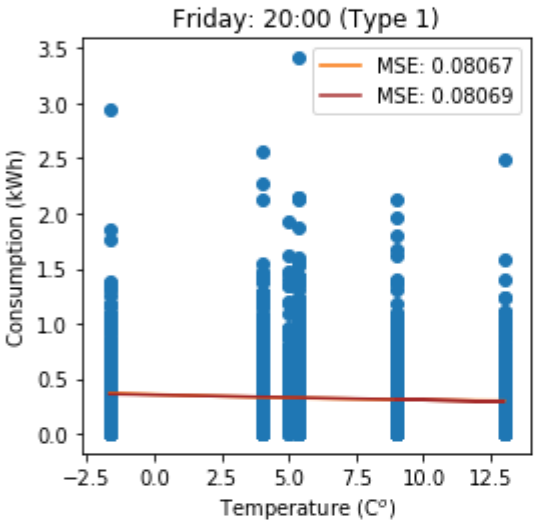

Friday: 18:00 (Type 2)
MSE: 0.36328
MSE: 0.36513
Consumption (kWh)
Temperature (C°)

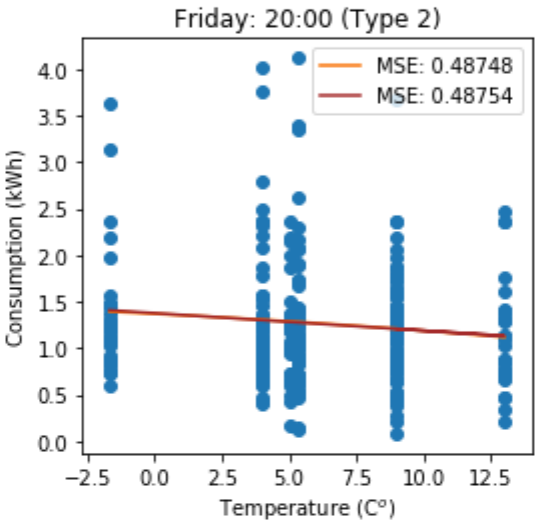

Friday: 18:00 (Type 3)
MSE: 0.1102
MSE: 0.1102
Consumption (kWh)
Temperature (C°)

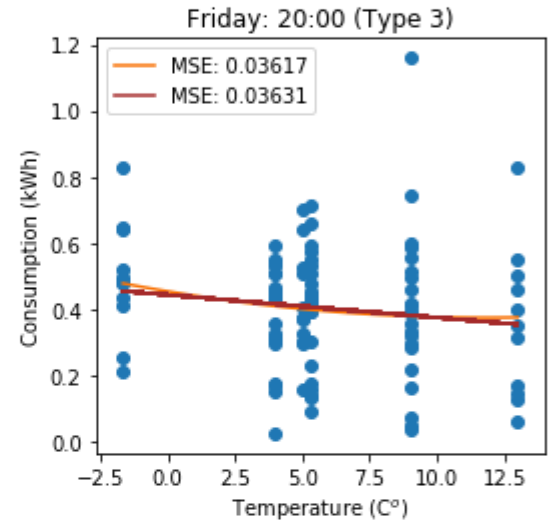

Friday: 19:00 (Type 1)
MSE: 0.09381
MSE: 0.09394
Consumption (kWh)
Temperature (C°)

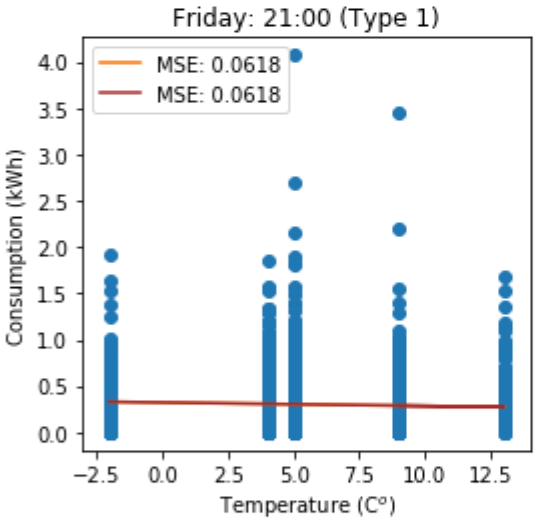

Friday: 19:00 (Type 2)
MSE: 0.45039
MSE: 0.45164
Consumption (kWh)
Temperature (C°)

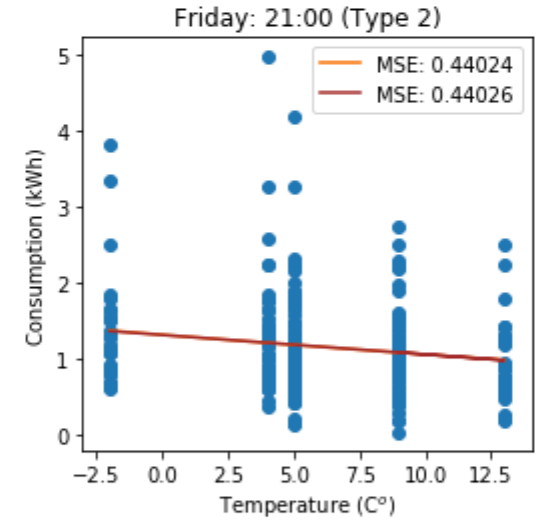

Friday: 19:00 (Type 3)
MSE: 0.05604
MSE: 0.05604
Consumption (kWh)
Temperature (C°)

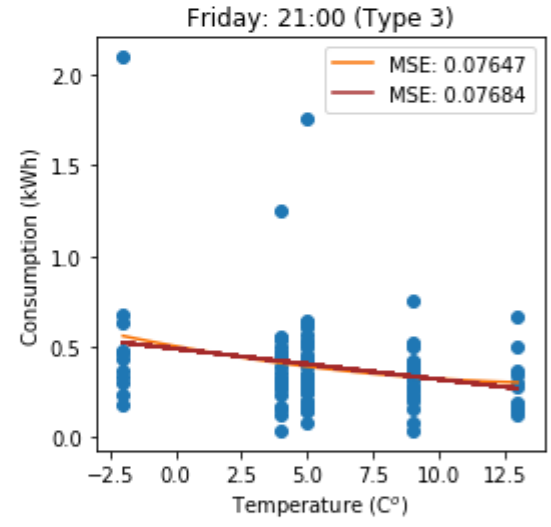

Friday: 20:00 (Type 1)
MSE: 0.08067
MSE: 0.08069
Consumption (kWh)
Temperature (C°)

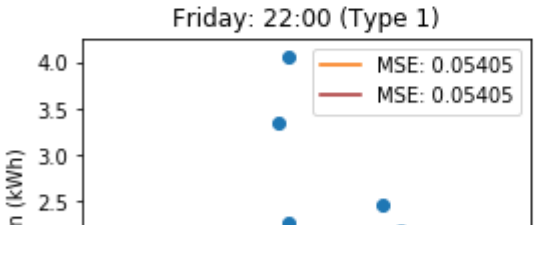

Friday: 20:00 (Type 2)
MSE: 0.48748
MSE: 0.48754
Consumption (kWh)
Temperature (C°)

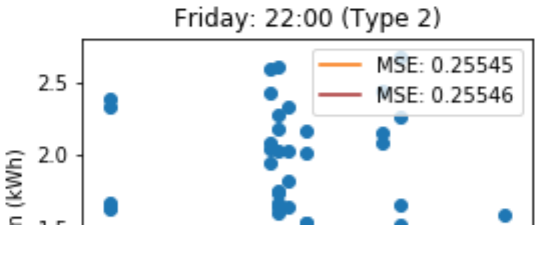

Friday: 20:00 (Type 3)
MSE: 0.03617
MSE: 0.03631
Consumption (kWh)
Temperature (C°)

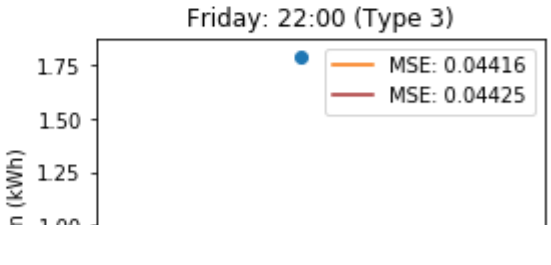

Friday: 21:00 (Type 1)
MSE: 0.0618
MSE: 0.0618
Consumption (kWh)
Temperature (C°)

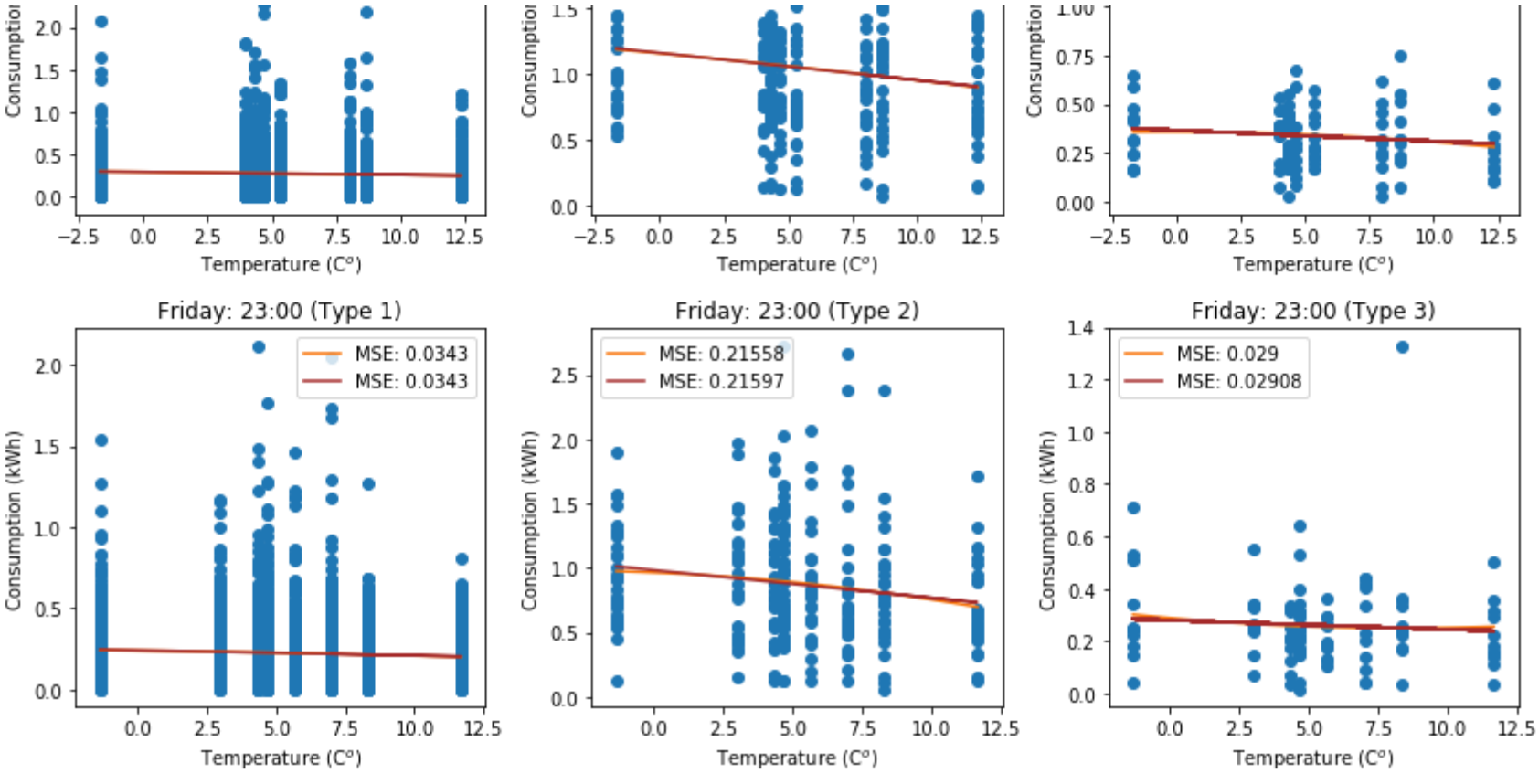

Friday: 21:00 (Type 2)
MSE: 0.44024
MSE: 0.44026
Consumption (kWh)
Temperature (C°)


Friday: 21:00 (Type 3)
MSE: 0.07647
MSE: 0.07684
Consumption (kWh)
Temperature (C°)

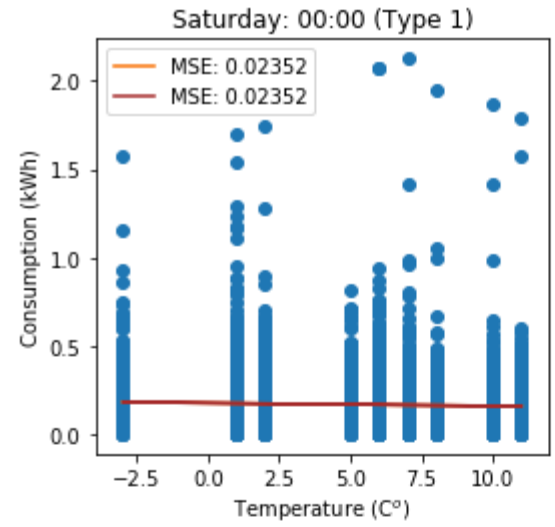

Friday: 22:00 (Type 1)
MSE: 0.05405
MSE: 0.05405

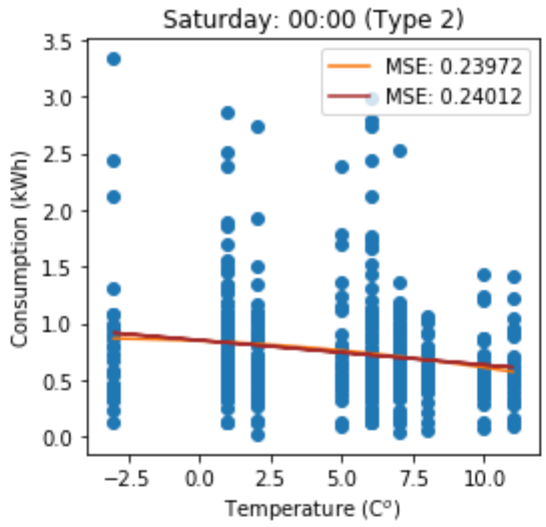

Friday: 22:00 (Type 2)
MSE: 0.25545
MSE: 0.25546

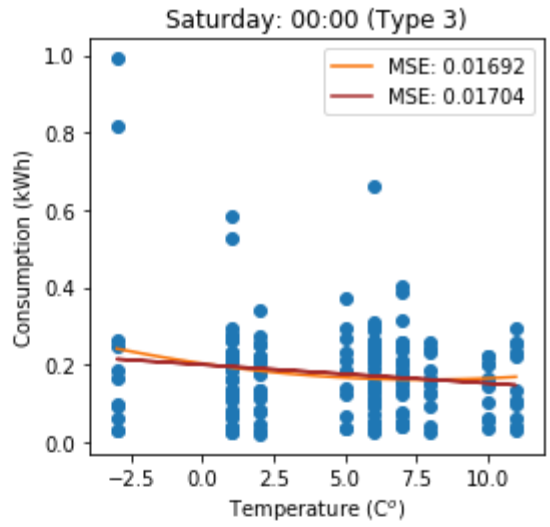

Friday: 22:00 (Type 3)
MSE: 0.04416
MSE: 0.04425

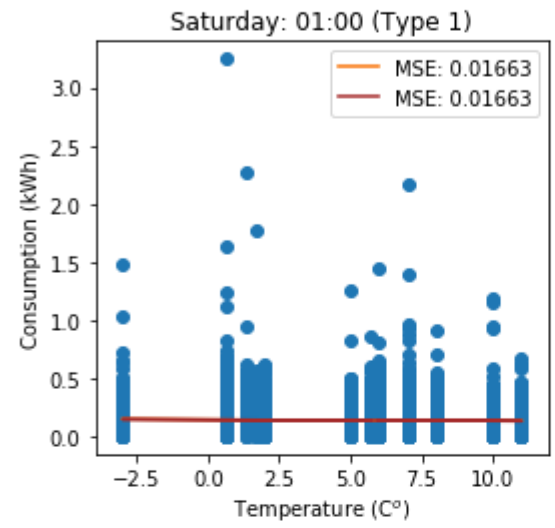

Friday: 23:00 (Type 1)
MSE: 0.0343
MSE: 0.0343
Friday: 23:00 (Type 2)
MSE: 0.21558
MSE: 0.21597
Friday: 23:00 (Type 3)
MSE: 0.029
MSE: 0.02908
Consumption (kWh)
Temperature (C°)

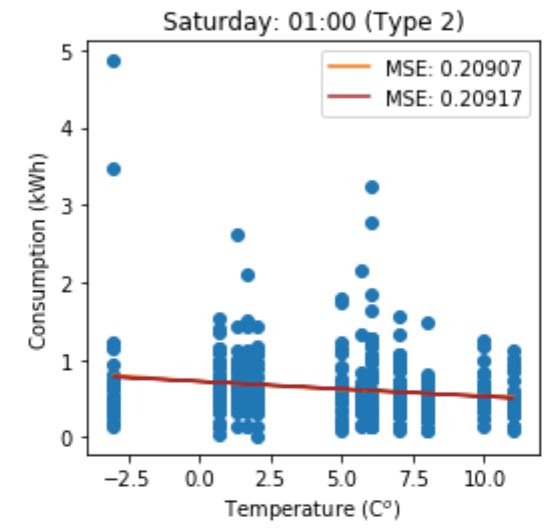
In [13]:
```
# Saturday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 5 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

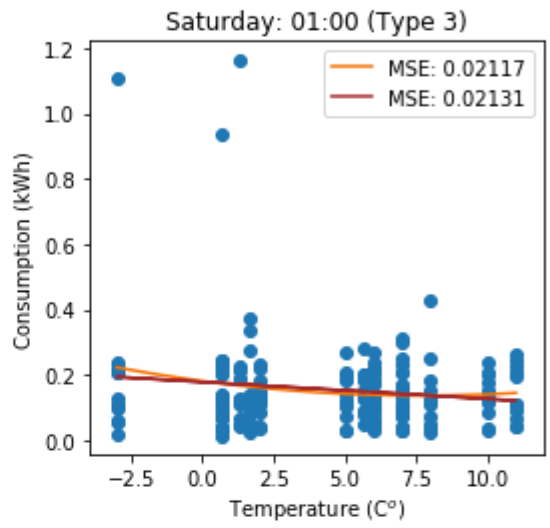

Saturday: 00:00 (Type 1)
MSE: 0.02352
MSE: 0.02352
Consumption (kWh)
Temperature (C°)

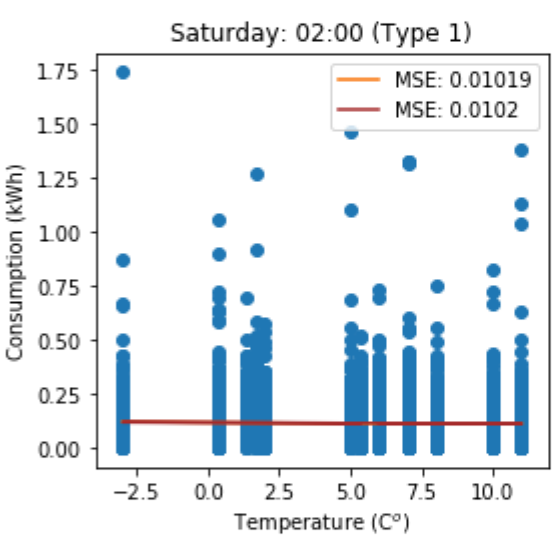

Saturday: 00:00 (Type 2)
MSE: 0.23972
MSE: 0.24012
Consumption (kWh)
Temperature (C°)

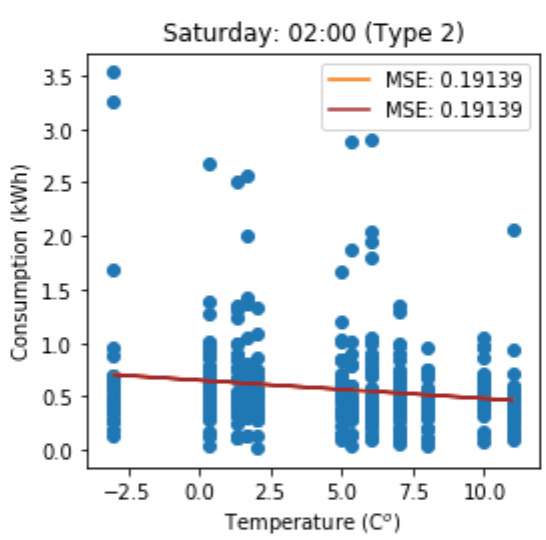

Saturday: 00:00 (Type 3)
MSE: 0.01692
MSE: 0.01704
Consumption (kWh)
Temperature (C°)

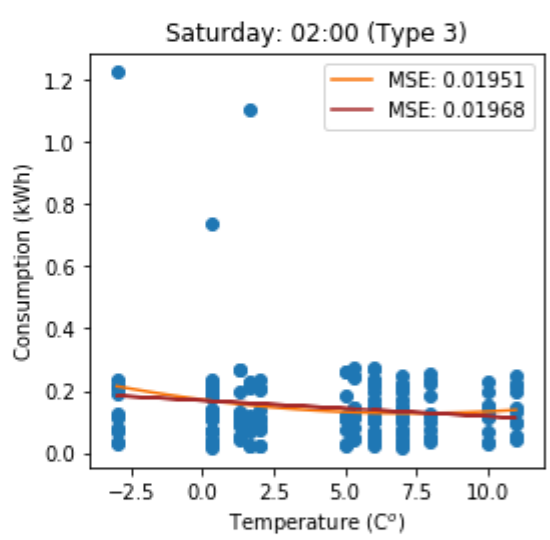

Saturday: 01:00 (Type 1)
MSE: 0.01663
MSE: 0.01663
Consumption (kWh)
Temperature (C°)

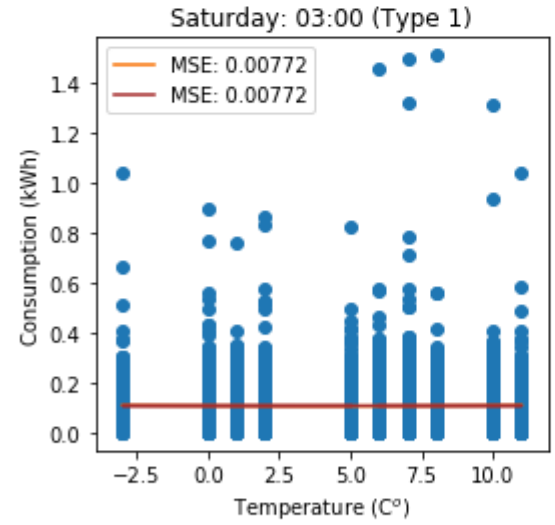

Saturday: 01:00 (Type 2)
MSE: 0.20907
MSE: 0.20917
Consumption (kWh)
Temperature (C°)

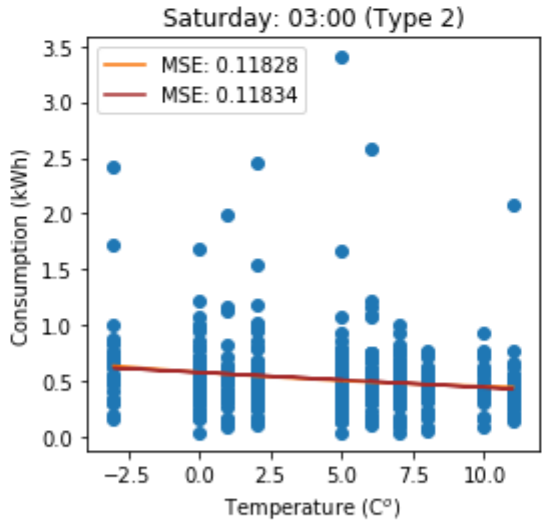

Saturday: 01:00 (Type 3)
MSE: 0.02117
MSE: 0.02131
Consumption (kWh)
Temperature (C°)

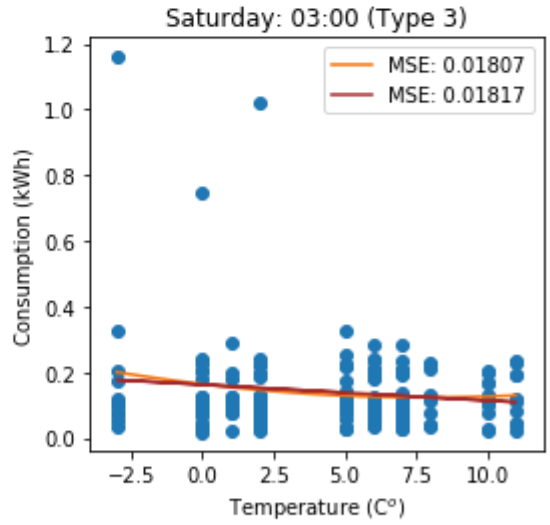

Saturday: 02:00 (Type 1)
MSE: 0.01019
MSE: 0.0102
Consumption (kWh)
Temperature (C°)

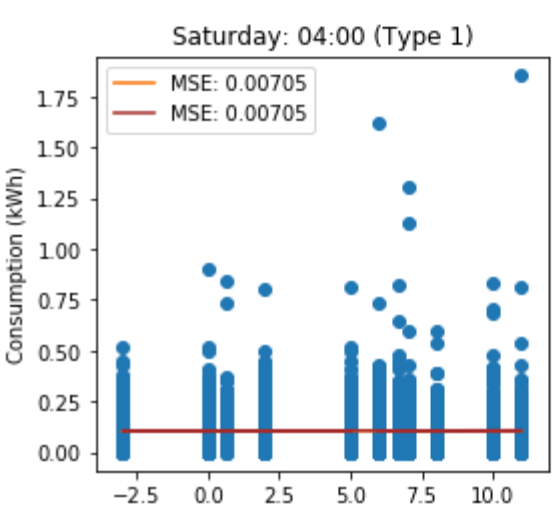

Saturday: 02:00 (Type 2)
MSE: 0.19139
MSE: 0.19139
Consumption (kWh)
Temperature (C°)

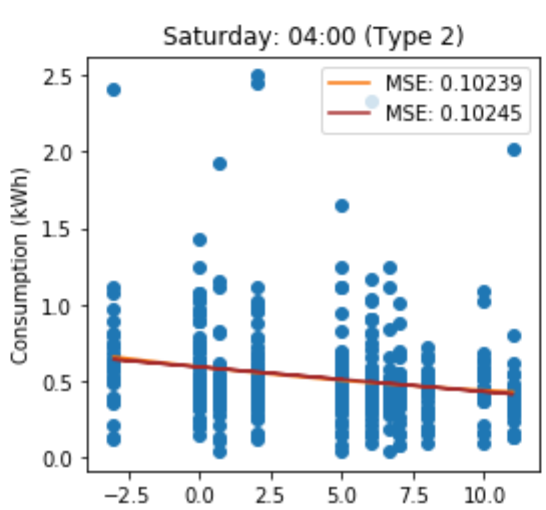

Saturday: 02:00 (Type 3)
MSE: 0.01951
MSE: 0.01968
Consumption (kWh)
Temperature (C°)

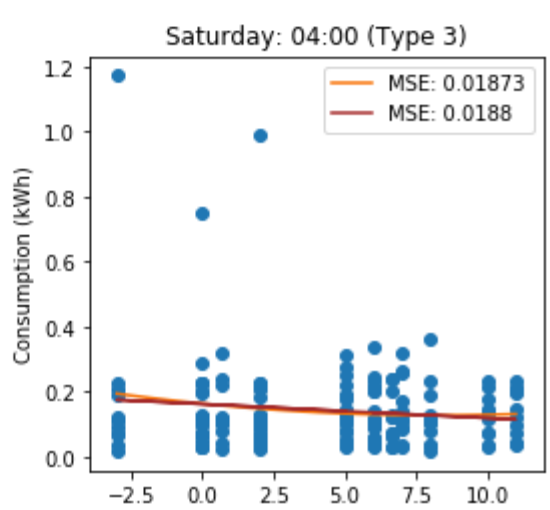

Saturday: 03:00 (Type 1)
MSE: 0.00772
MSE: 0.00772
Consumption (kWh)
Temperature (C°)

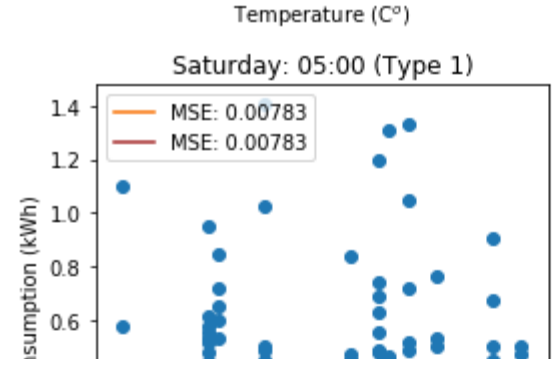

Saturday: 03:00 (Type 2)
MSE: 0.11828
MSE: 0.11834
Consumption (kWh)
Temperature (C°)

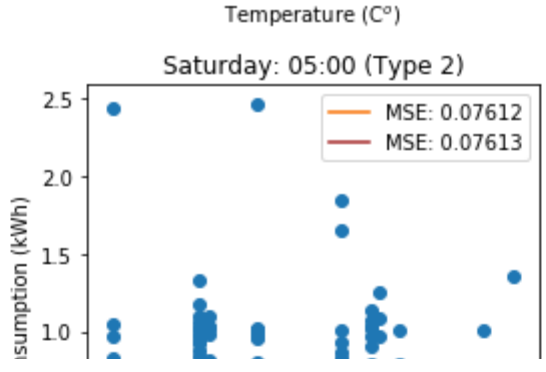

Saturday: 03:00 (Type 3)
MSE: 0.01807
MSE: 0.01817
Consumption (kWh)
Temperature (C°)

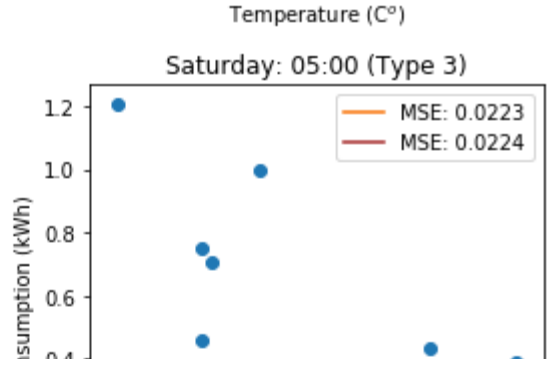

Saturday: 04:00 (Type 1)
MSE: 0.00705
MSE: 0.00705
Consumption (kWh)
Temperature (C°)

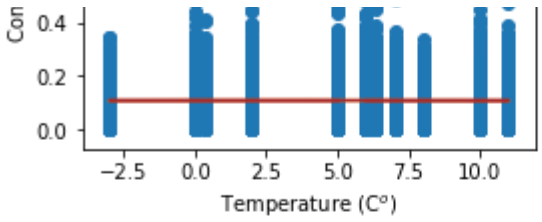

Saturday: 04:00 (Type 2)
MSE: 0.10239
MSE: 0.10245
Consumption (kWh)
Temperature (C°)

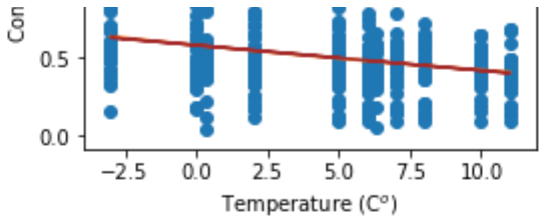

Saturday: 04:00 (Type 3)
MSE: 0.01873
MSE: 0.0188
Consumption (kWh)
Temperature (C°)

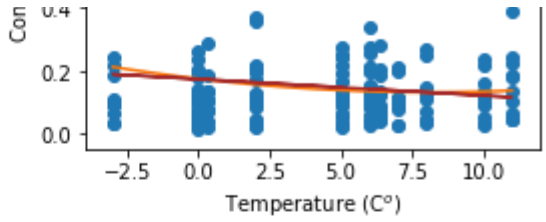

Saturday: 05:00 (Type 1)
MSE: 0.00783
MSE: 0.00783

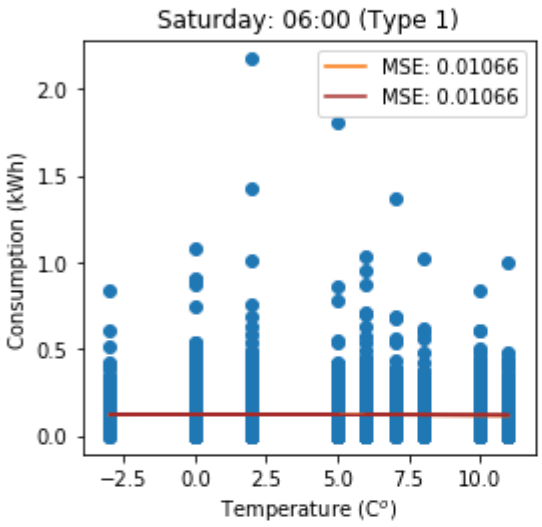

Saturday: 05:00 (Type 2)
MSE: 0.07612
MSE: 0.07613

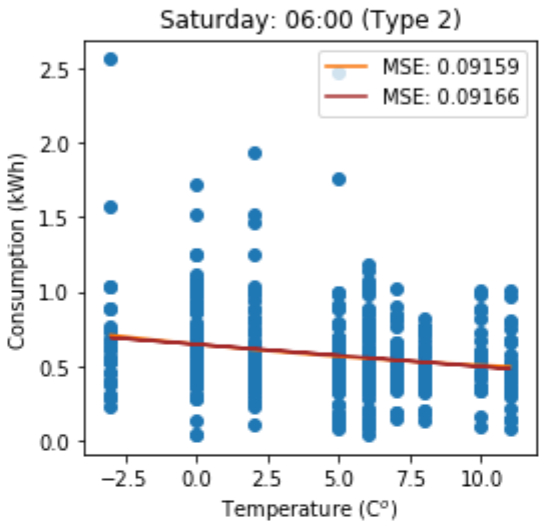

Saturday: 05:00 (Type 3)
MSE: 0.0223
MSE: 0.0224

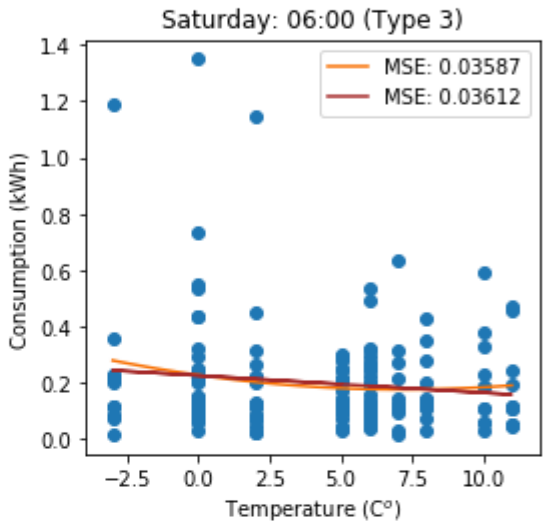

Temperature (C°)

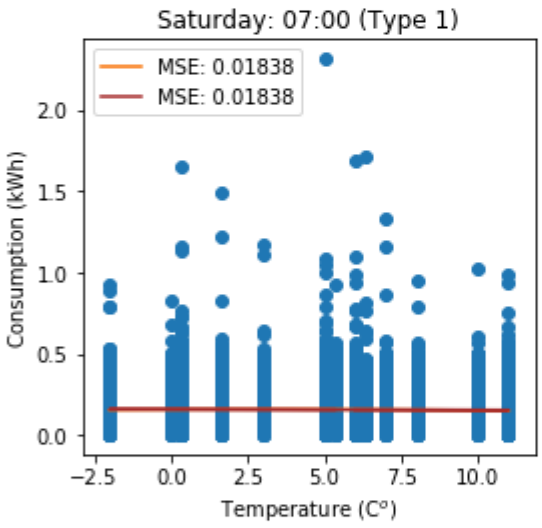

Temperature (C°)

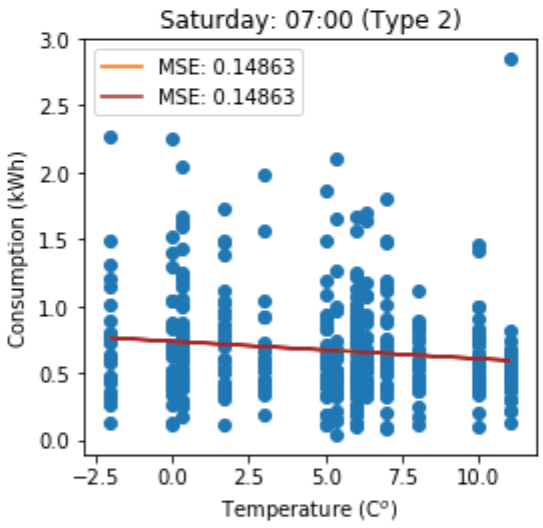

Temperature (C°)

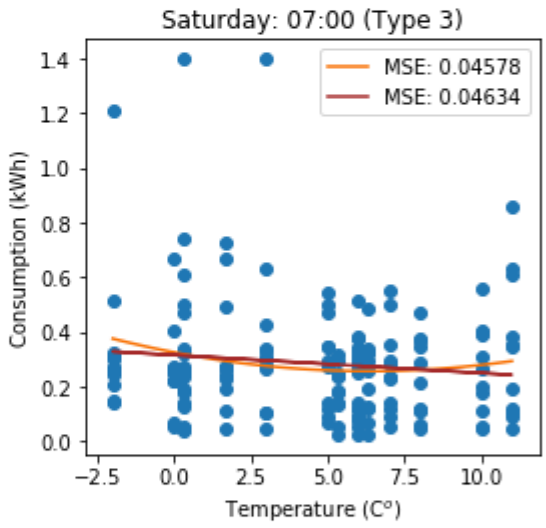

Saturday: 06:00 (Type 1)
MSE: 0.01066
MSE: 0.01066
Consumption (kWh)
Temperature (C°)

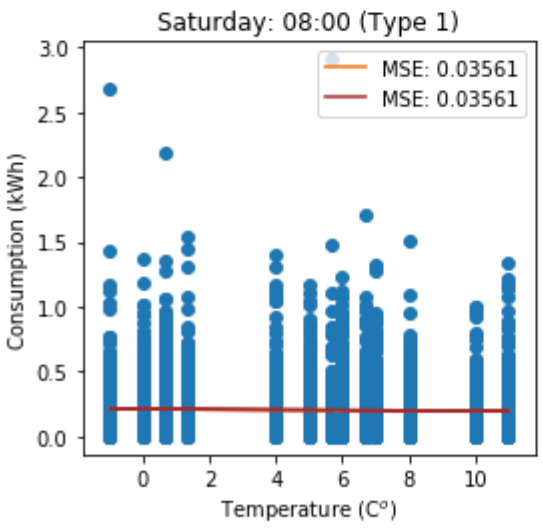

Saturday: 06:00 (Type 2)
MSE: 0.09159
MSE: 0.09166
Consumption (kWh)
Temperature (C°)

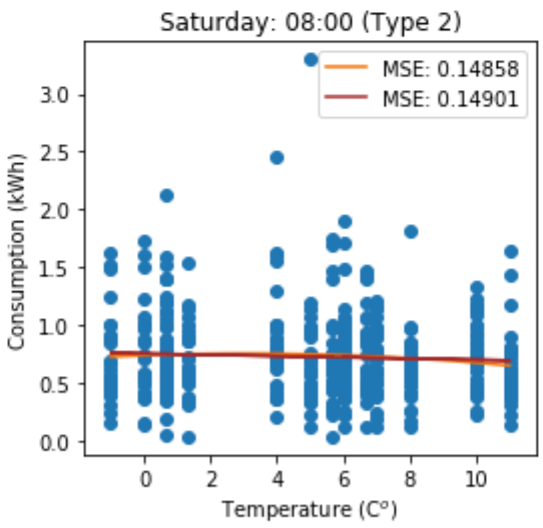

Saturday: 06:00 (Type 3)
MSE: 0.03587
MSE: 0.03612
Consumption (kWh)
Temperature (C°)

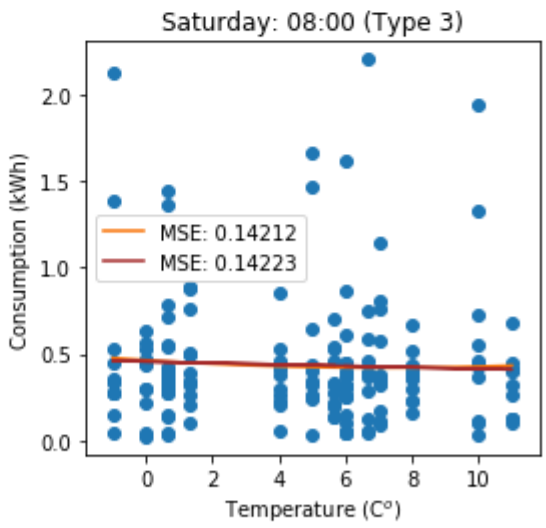

Saturday: 07:00 (Type 1)
MSE: 0.01838
MSE: 0.01838
Consumption (kWh)
Temperature (C°)

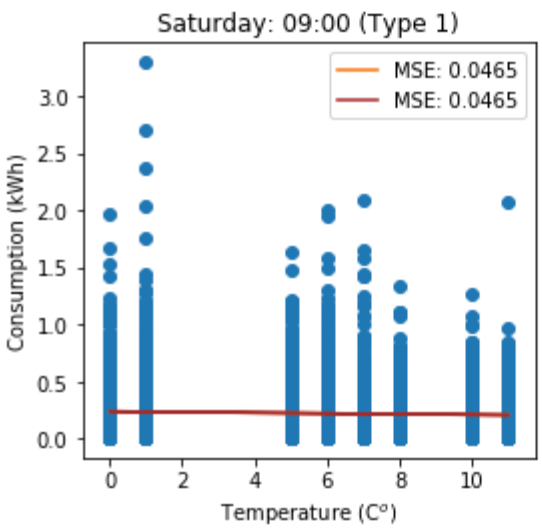

Saturday: 07:00 (Type 2)
MSE: 0.14863
MSE: 0.14863
Consumption (kWh)
Temperature (C°)

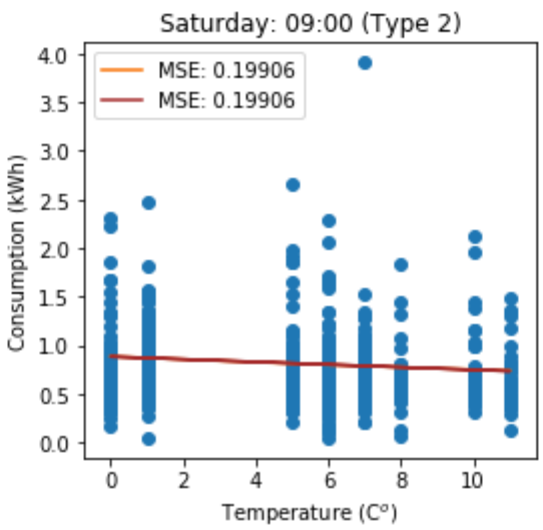

Saturday: 07:00 (Type 3)
MSE: 0.04578
MSE: 0.04634
Consumption (kWh)
Temperature (C°)

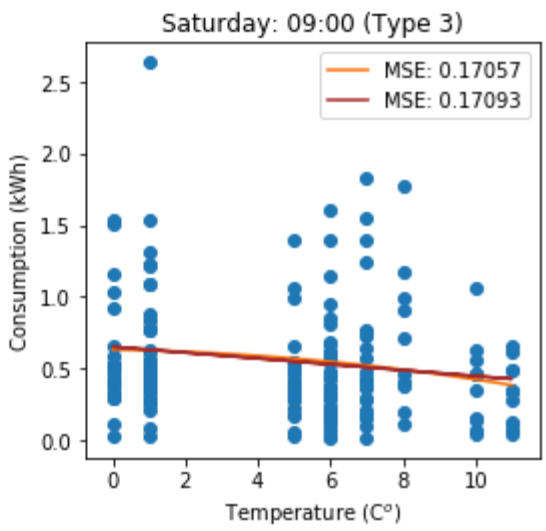

Saturday: 08:00 (Type 1)
MSE: 0.03561
MSE: 0.03561
Consumption (kWh)
Temperature (C°)

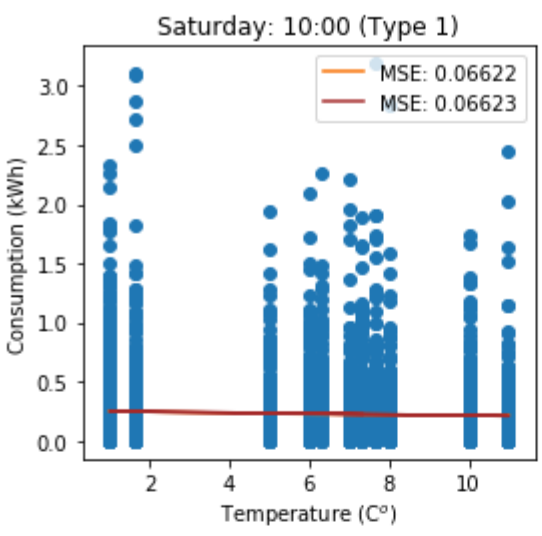

Saturday: 08:00 (Type 2)
MSE: 0.14858
MSE: 0.14901
Consumption (kWh)
Temperature (C°)

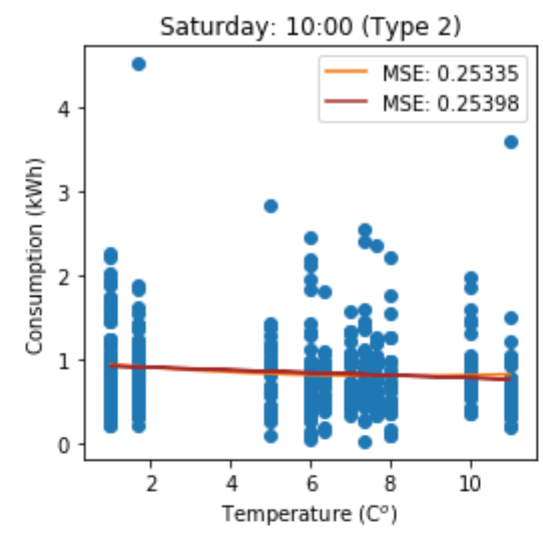

Saturday: 08:00 (Type 3)
MSE: 0.14212
MSE: 0.14223
Consumption (kWh)
Temperature (C°)

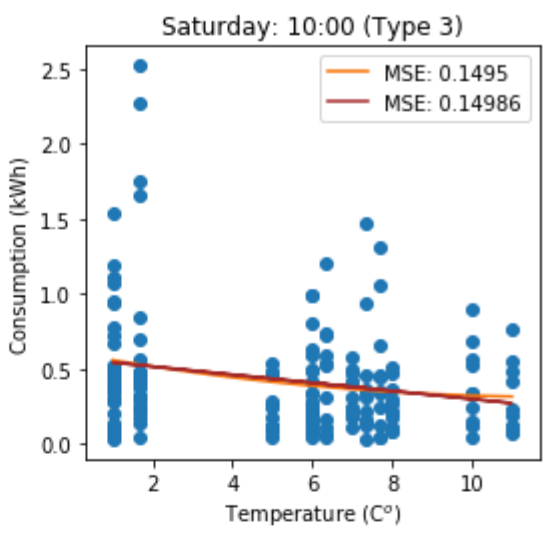

Saturday: 09:00 (Type 1)
MSE: 0.0465
MSE: 0.0465
Consumption (kWh)
Temperature (C°)

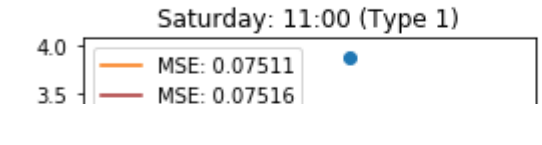

Saturday: 09:00 (Type 2)
MSE: 0.19906
MSE: 0.19906
Consumption (kWh)
Temperature (C°)

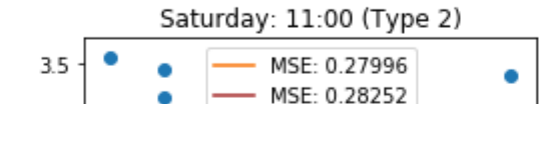

Saturday: 09:00 (Type 3)
MSE: 0.17057
MSE: 0.17093
Consumption (kWh)
Temperature (C°)


Saturday: 10:00 (Type 1)
MSE: 0.06622
MSE: 0.06623
Consumption (kWh)
Temperature (C°)

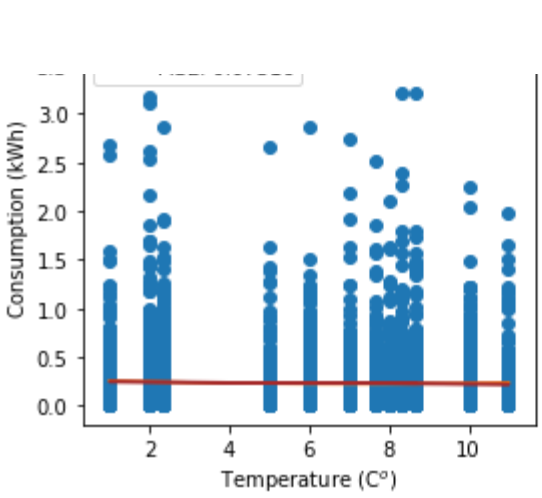

Saturday: 10:00 (Type 2)
MSE: 0.25335
MSE: 0.25398
Consumption (kWh)
Temperature (C°)

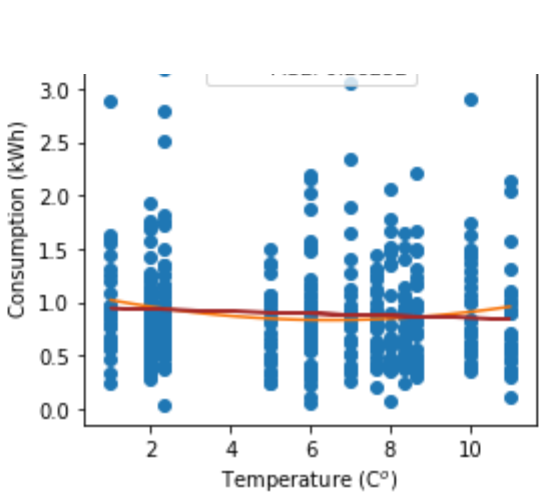

Saturday: 10:00 (Type 3)
MSE: 0.1495
MSE: 0.14986
Consumption (kWh)
Temperature (C°)

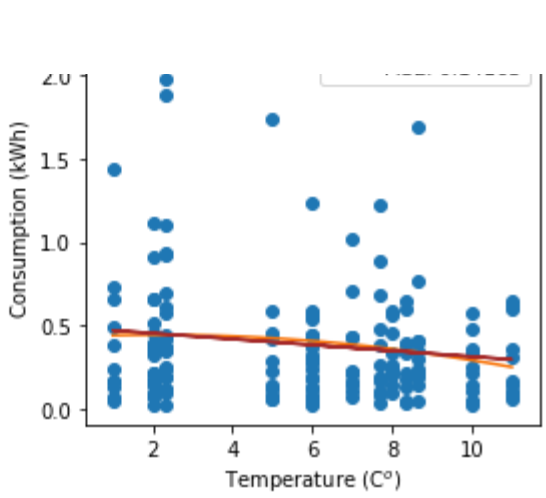

Saturday: 11:00 (Type 1)
MSE: 0.07511
MSE: 0.07516

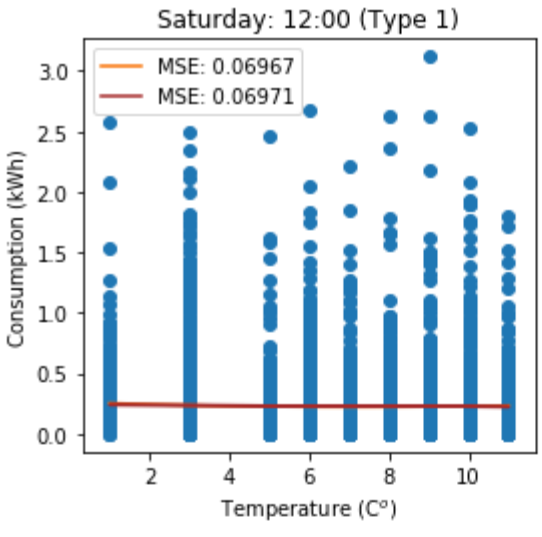

Saturday: 11:00 (Type 2)
MSE: 0.27996
MSE: 0.28252

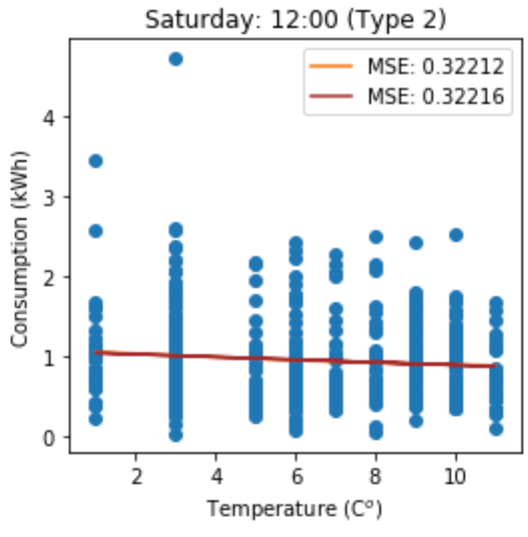

Saturday: 11:00 (Type 3)
MSE: 0.14122
MSE: 0.14163

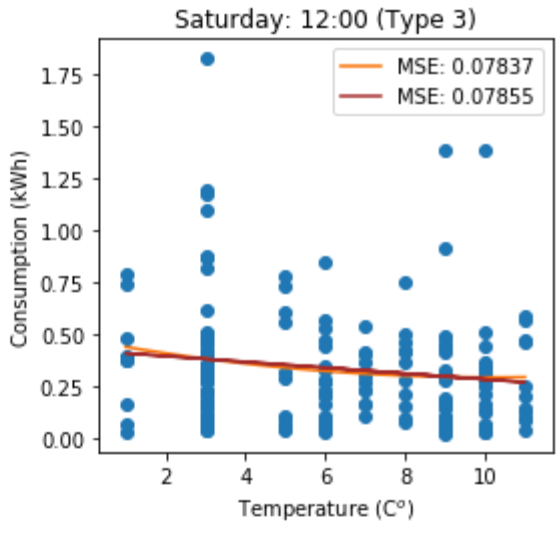

Consumption (kWh)
Temperature (C°)

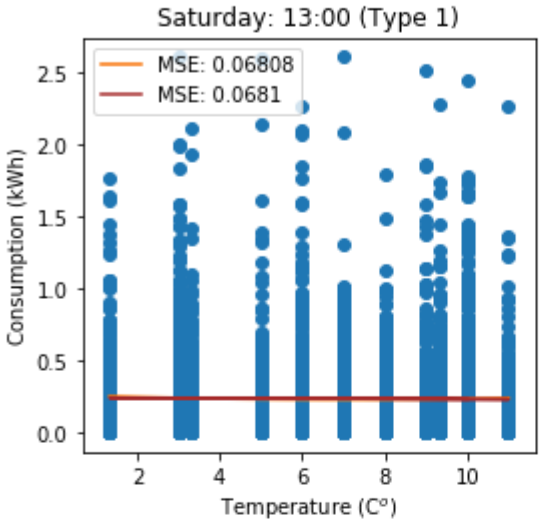

Consumption (kWh)
Temperature (C°)

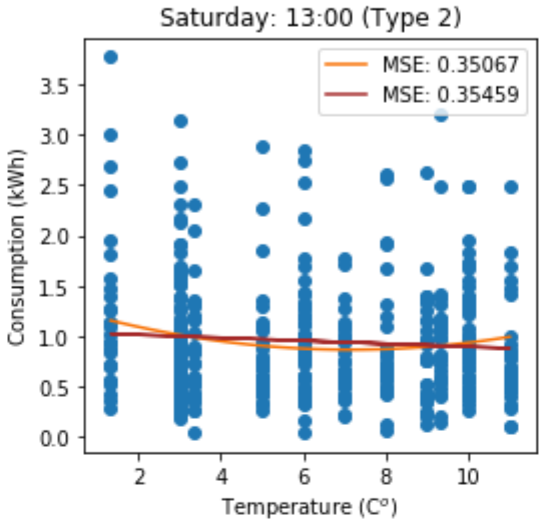

Consumption (kWh)
Temperature (C°)

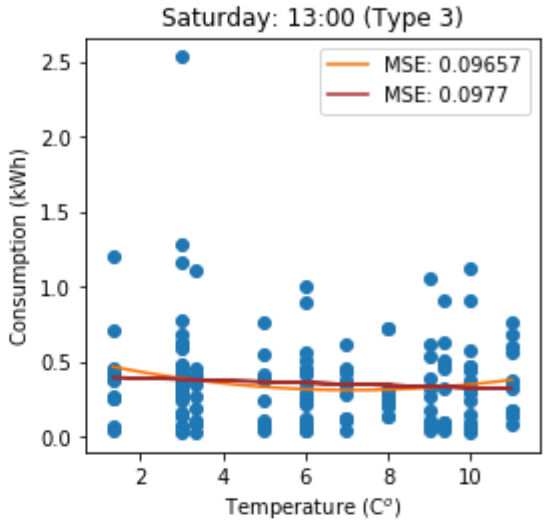

Saturday: 12:00 (Type 1)
MSE: 0.06967
MSE: 0.06971
Consumption (kWh)
Temperature (C°)

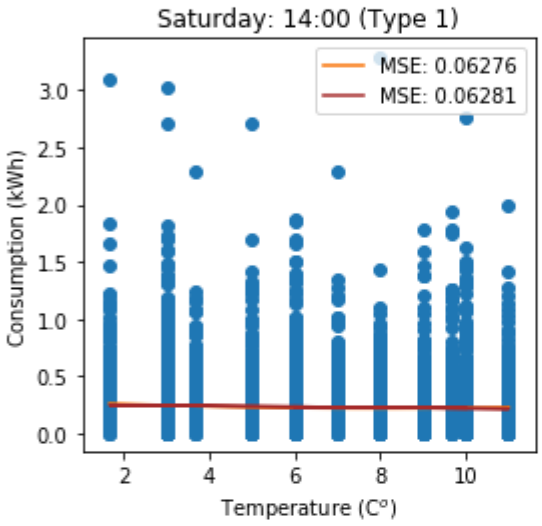

Saturday: 12:00 (Type 2)
MSE: 0.32212
MSE: 0.32216
Consumption (kWh)
Temperature (C°)

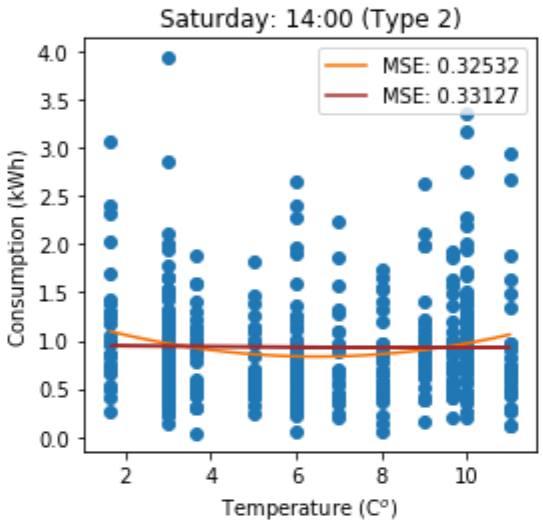

Saturday: 12:00 (Type 3)
MSE: 0.07837
MSE: 0.07855
Consumption (kWh)
Temperature (C°)

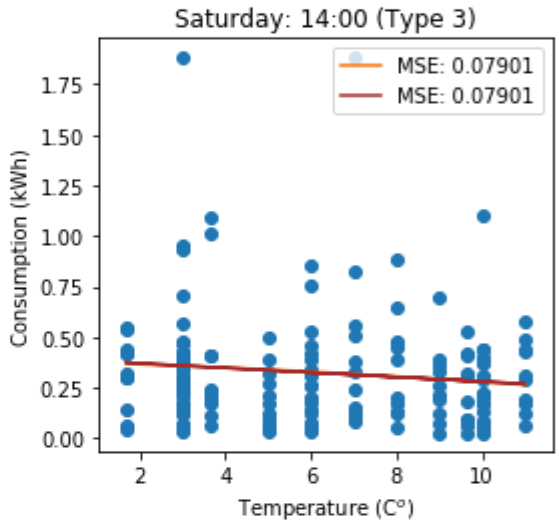

Saturday: 13:00 (Type 1)
MSE: 0.06808
MSE: 0.0681
Consumption (kWh)
Temperature (C°)

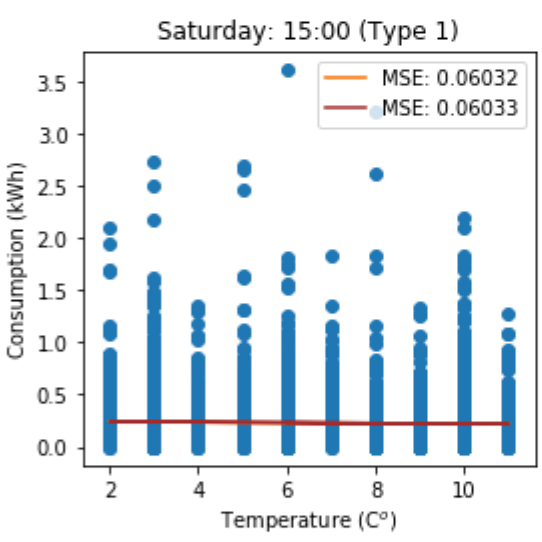

Saturday: 13:00 (Type 2)
MSE: 0.35067
MSE: 0.35459
Consumption (kWh)
Temperature (C°)

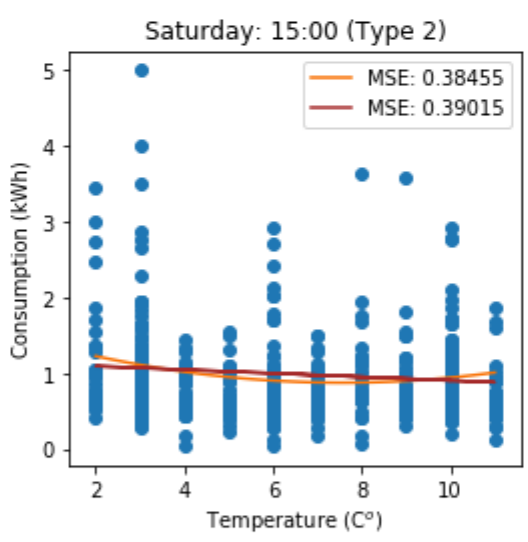

Saturday: 13:00 (Type 3)
MSE: 0.09657
MSE: 0.0977
Consumption (kWh)
Temperature (C°)

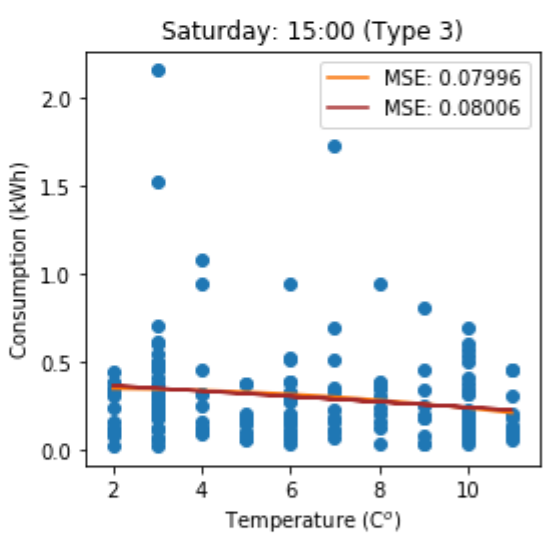

Saturday: 14:00 (Type 1)
MSE: 0.06276
MSE: 0.06281
Consumption (kWh)
Temperature (C°)

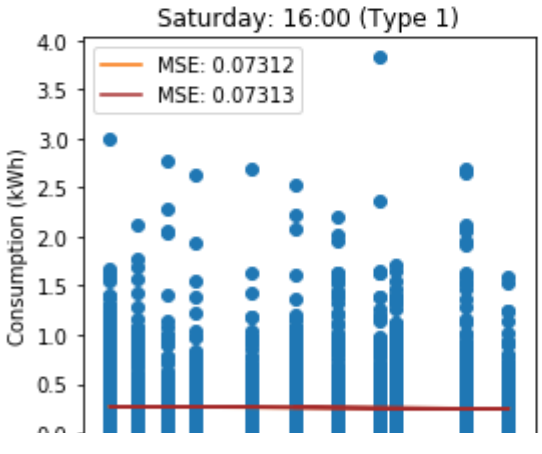

Saturday: 14:00 (Type 2)
MSE: 0.32532
MSE: 0.33127
Consumption (kWh)
Temperature (C°)

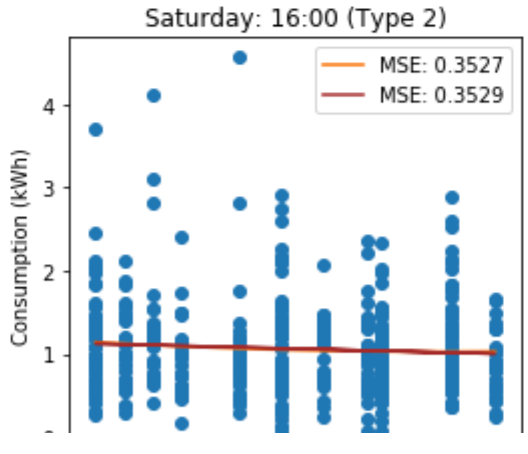

Saturday: 14:00 (Type 3)
MSE: 0.07901
MSE: 0.07901
Consumption (kWh)
Temperature (C°)

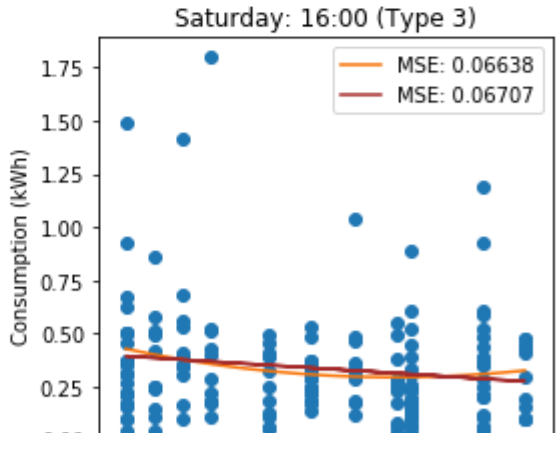

Saturday: 15:00 (Type 1)
MSE: 0.06032
MSE: 0.06033
Consumption (kWh)
Temperature (C°)

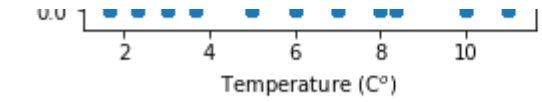

Saturday: 15:00 (Type 2)
MSE: 0.38455
MSE: 0.39015
Consumption (kWh)
Temperature (C°)

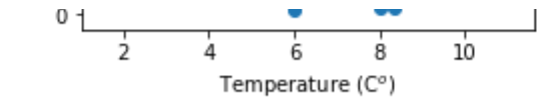

Saturday: 15:00 (Type 3)
MSE: 0.07996
MSE: 0.08006
Consumption (kWh)
Temperature (C°)

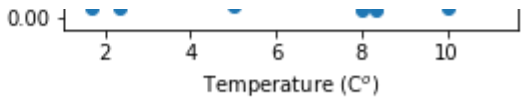

Saturday: 16:00 (Type 1)
MSE: 0.07312
MSE: 0.07313
Consumption (kWh)

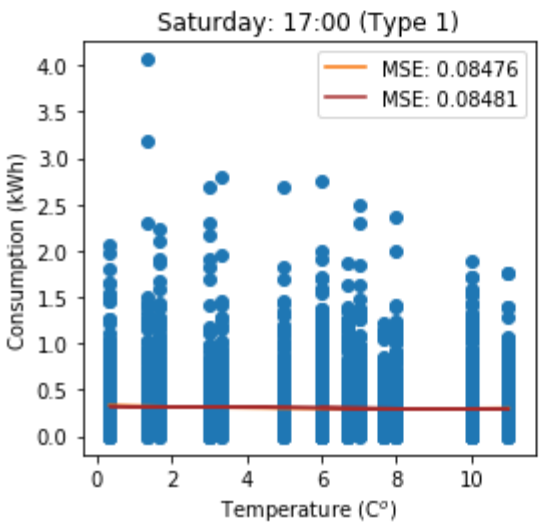

Saturday: 16:00 (Type 2)
MSE: 0.3527
MSE: 0.3529
Consumption (kWh)

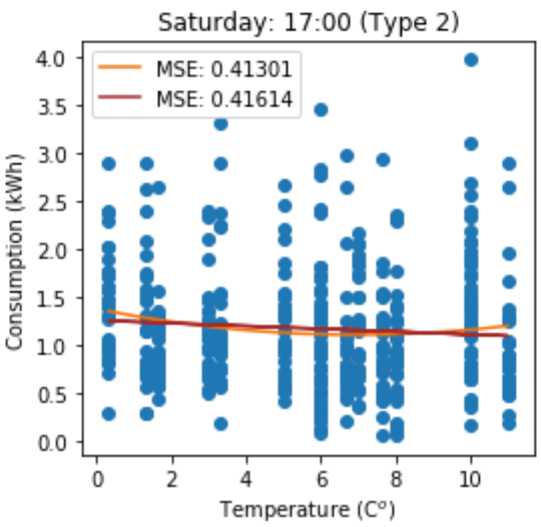

Saturday: 16:00 (Type 3)
MSE: 0.06638
MSE: 0.06707
Consumption (kWh)

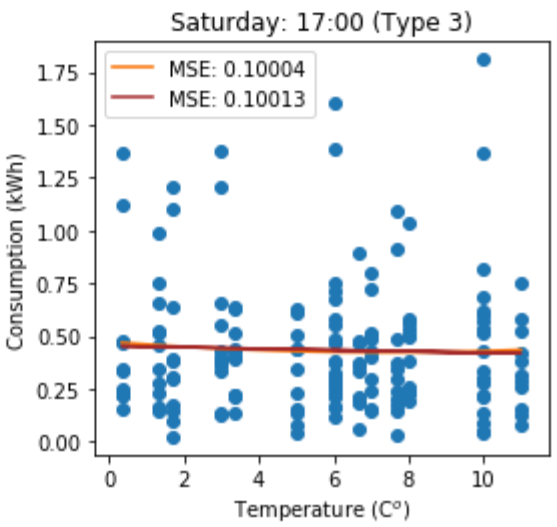

Temperature (C°)

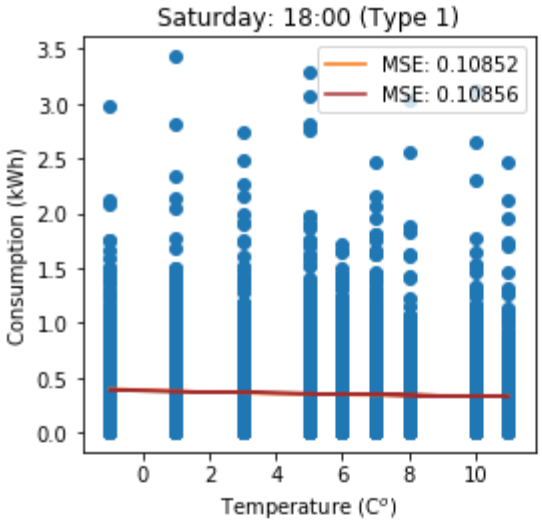

Temperature (C°)

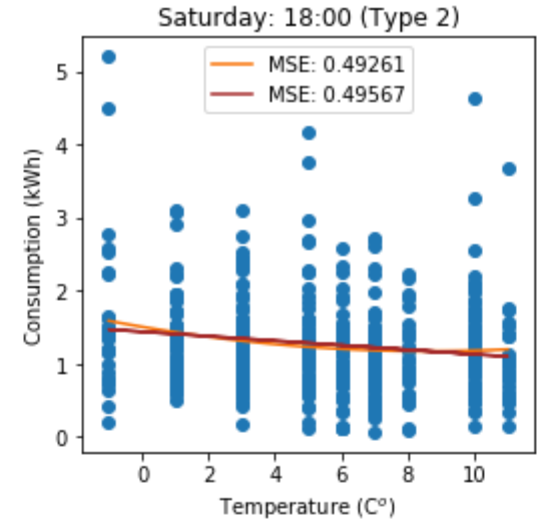

Temperature (C°)

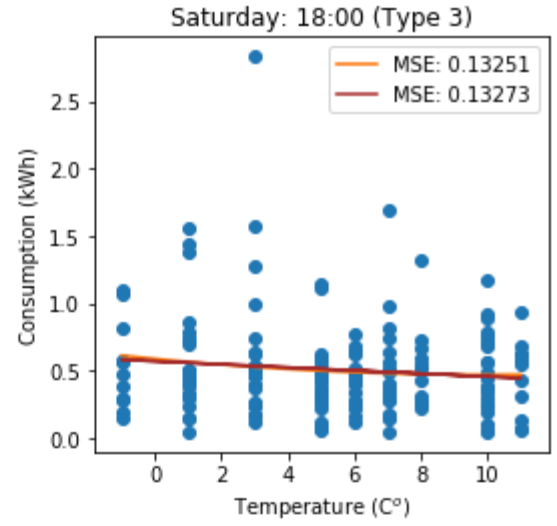

Saturday: 17:00 (Type 1)
MSE: 0.08476
MSE: 0.08481
Consumption (kWh)
Temperature (C°)

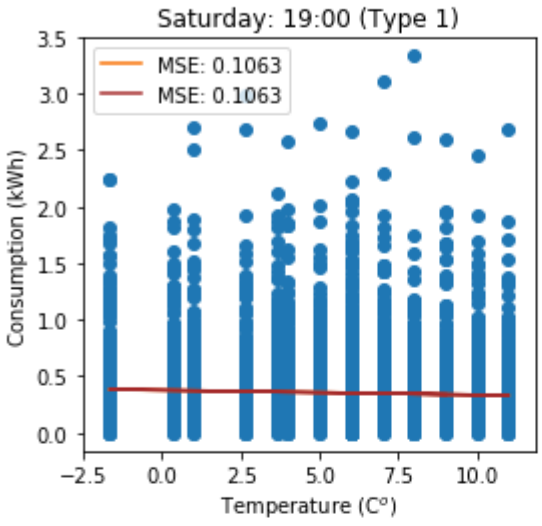

Saturday: 17:00 (Type 2)
MSE: 0.41301
MSE: 0.41614
Consumption (kWh)
Temperature (C°)

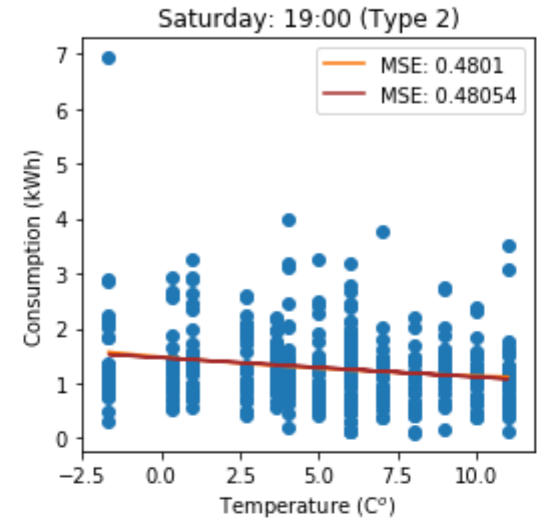

Saturday: 17:00 (Type 3)
MSE: 0.10004
MSE: 0.10013
Consumption (kWh)
Temperature (C°)

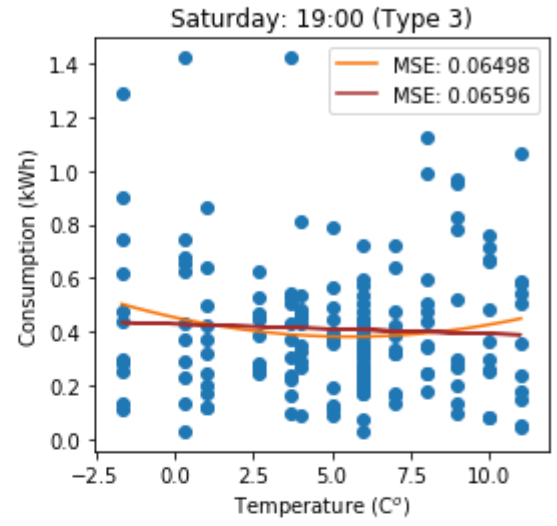

Saturday: 18:00 (Type 1)
MSE: 0.10852
MSE: 0.10856
Consumption (kWh)
Temperature (C°)

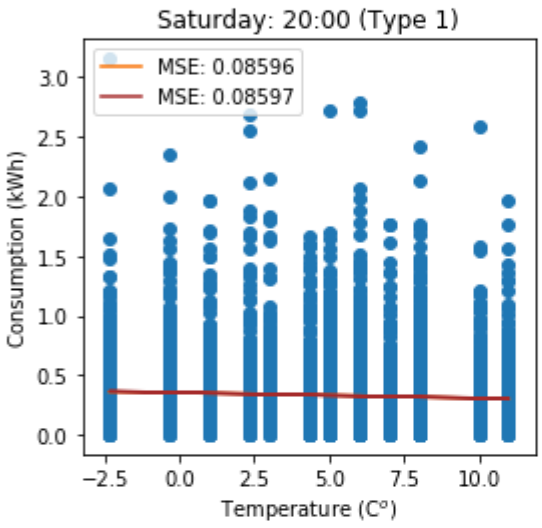

Saturday: 18:00 (Type 2)
MSE: 0.49261
MSE: 0.49567
Consumption (kWh)
Temperature (C°)

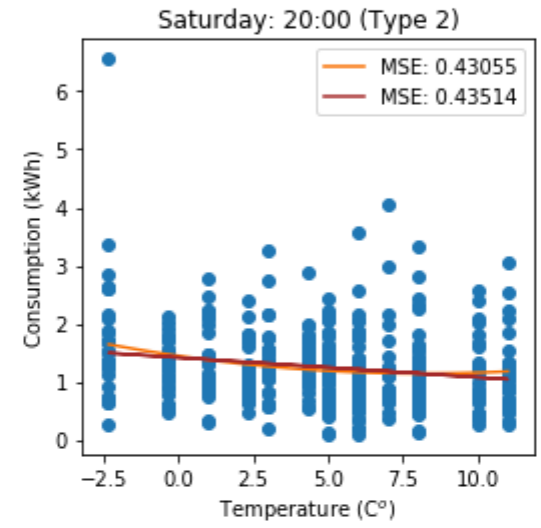

Saturday: 18:00 (Type 3)
MSE: 0.13251
MSE: 0.13273
Consumption (kWh)
Temperature (C°)

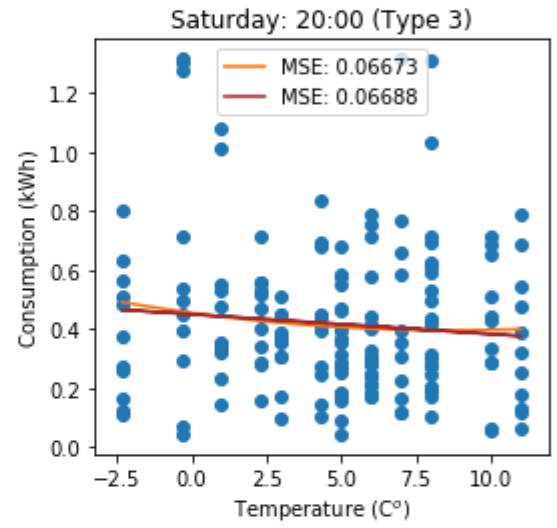

Saturday: 19:00 (Type 1)
MSE: 0.1063
MSE: 0.1063
Consumption (kWh)
Temperature (C°)

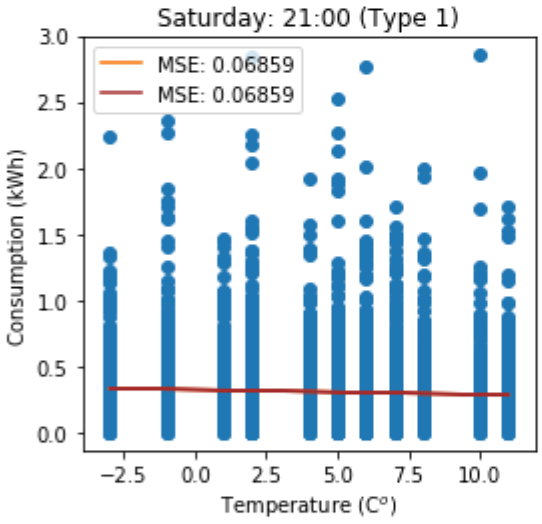

Saturday: 19:00 (Type 2)
MSE: 0.4801
MSE: 0.48054
Consumption (kWh)
Temperature (C°)

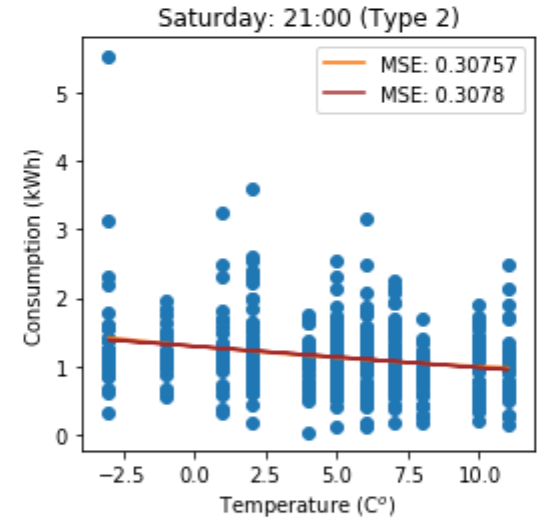

Saturday: 19:00 (Type 3)
MSE: 0.06498
MSE: 0.06596
Consumption (kWh)
Temperature (C°)

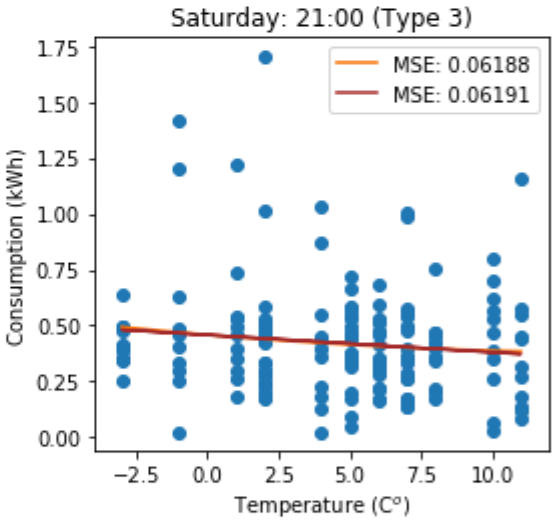

Saturday: 20:00 (Type 1)
MSE: 0.08596
MSE: 0.08597
Consumption (kWh)
Temperature (C°)


Saturday: 20:00 (Type 2)
MSE: 0.43055
MSE: 0.43514
Consumption (kWh)
Temperature (C°)


Saturday: 20:00 (Type 3)
MSE: 0.06673
MSE: 0.06688
Consumption (kWh)
Temperature (C°)


Saturday: 21:00 (Type 1)
MSE: 0.06859
MSE: 0.06859
Consumption (kWh)
Temperature (C°)

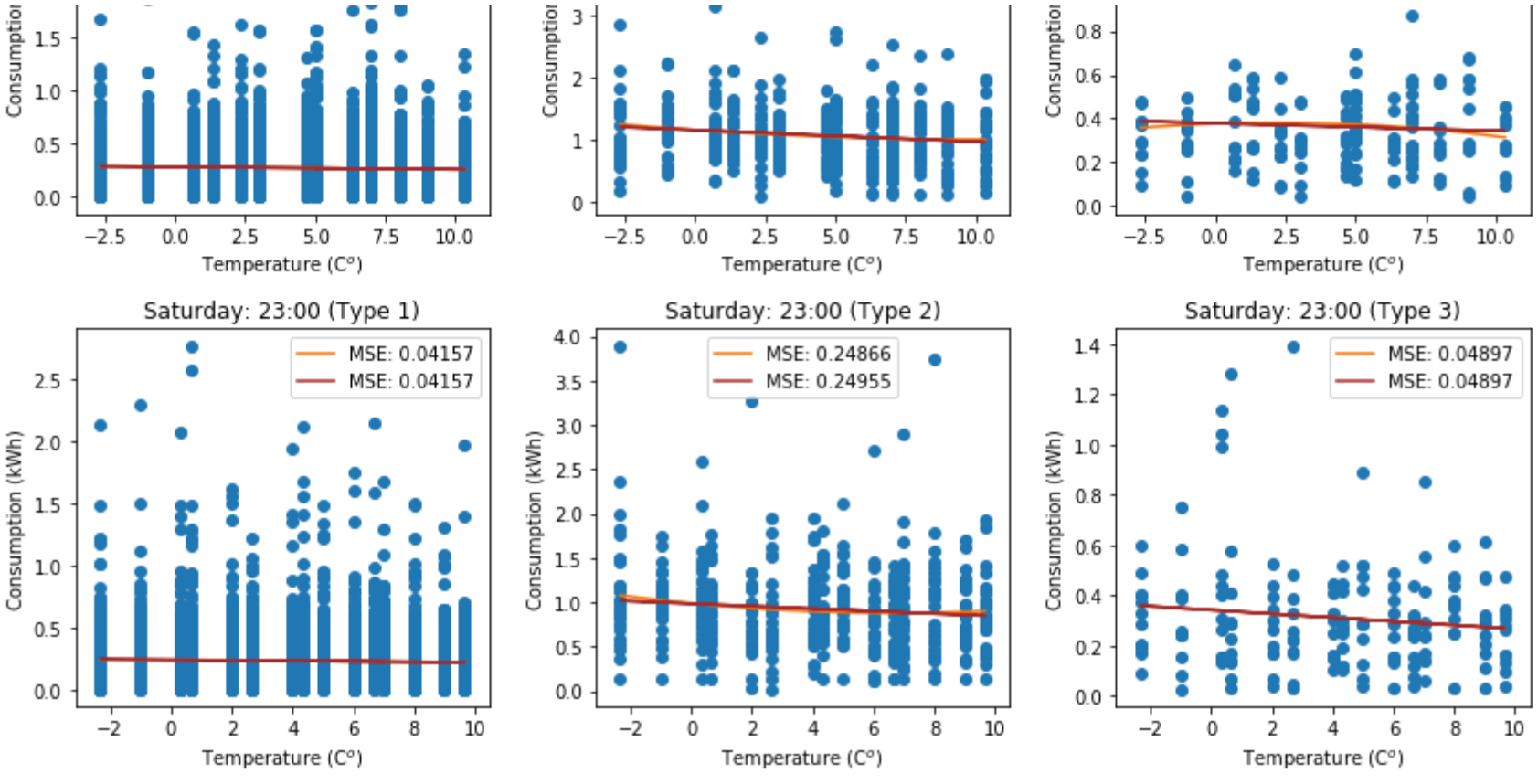

Saturday: 21:00 (Type 2)
MSE: 0.30757
MSE: 0.3078
Consumption (kWh)
Temperature (C°)


Saturday: 21:00 (Type 3)
MSE: 0.06188
MSE: 0.06191
Consumption (kWh)
Temperature (C°)

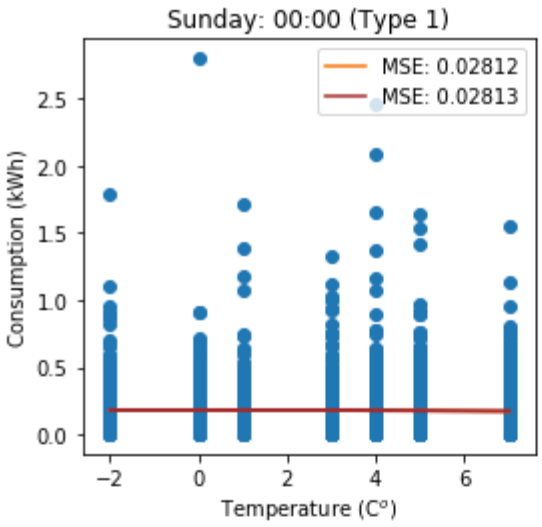
Saturday: 22:00 (Type 1)
MSE: 0.05085
MSE: 0.05085

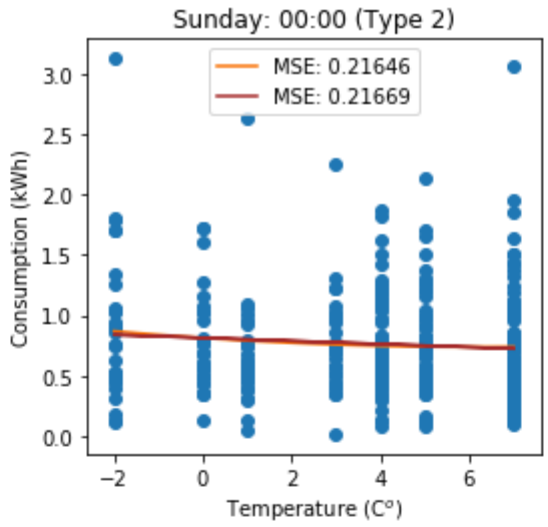
Saturday: 22:00 (Type 2)
MSE: 0.29035
MSE: 0.29076

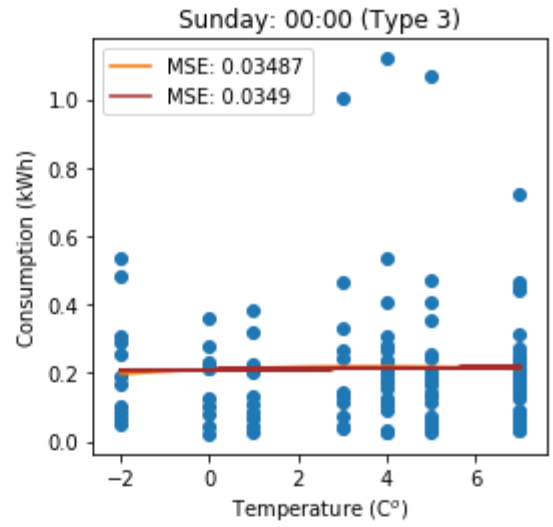
Saturday: 22:00 (Type 3)
MSE: 0.05194
MSE: 0.05214

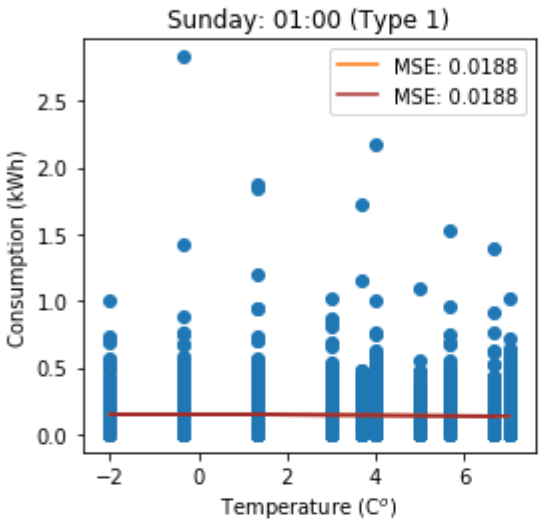

Saturday: 23:00 (Type 1)
MSE: 0.04157
MSE: 0.04157
Saturday: 23:00 (Type 2)
MSE: 0.24866
MSE: 0.24955
Saturday: 23:00 (Type 3)
MSE: 0.04897
MSE: 0.04897
Consumption (kWh)
Temperature (C°)

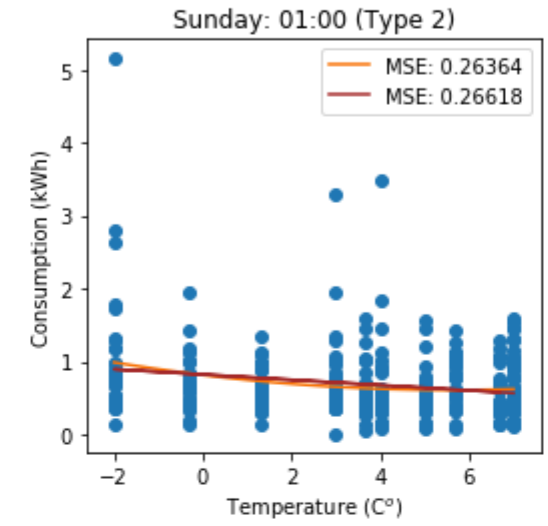
In [14]:
```
# Sunday hourly regressions in cold seasons
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
fig_all = plt.figure(figsize = (12,90))
ax_Ntou = [] # store subplot objects
weeks = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
day_of_week = 6 # 0 is Monday, 6 is Sunday
lm = LinearRegression()
for g in range(3): # doing this for three groups
    for i in range(24):
        ax_Ntou.append(fig_all.add_subplot(24, 3, 3 * i + (g + 1)))
        if i <= 9:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains('0' + str(i) + ':00:00')) & (pd.to_datetime
(z.GMT).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': 0' + str(i) + ':00 (Type ' + str(g+1) + ')' )
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
        else:
            z = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
            y_temp = z[['GMT'] + list(houseLableDf[(houseLableDf.Label == houseGroupDf.Label.iloc[g]) ].House
)]
            y = y_temp[pd.to_datetime(z.GMT).dt.dayofweek == day_of_week] # find the users' data for the corr
esponding day of a week
            y.set_index('GMT', inplace = True)
            y = y.fillna(y.mean()).values
            y = np.reshape(y, y.shape[0] * y.shape[1], order = 'F') # reconstruct the y variable

            x = df_wea1h_nf_cold.TempC[(df_wea1h.GMT.str.contains(str(i) + ':00:00')) & (pd.to_datetime(z.GMT
).dt.dayofweek == day_of_week)].values
            x = np.tile(x, reps = sum(houseLableDf.Label == houseGroupDf.Label.iloc[g])) # reconstruct the x
 variable
            xp = np.linspace(min(x), max(x), 50) # polynomial regression
            z = np.polyfit(x, y, 2)
            p = np.poly1d(z)
            ax_Ntou[-1].plot(xp, p(xp), color = 'xkcd:orange', label = 'MSE: ' + str(round(mean_squared_error
(y, p(x)), 5)))
            x = x.reshape(-1, 1)
            y = y.reshape(-1, 1)
            lm.fit(x, y)
            ax_Ntou[-1].plot(x, lm.predict(x), color = 'brown', label = 'MSE: ' + str(round(mean_squared_erro
r(y, lm.predict(x)), 5)))
            ax_Ntou[-1].scatter(x, y)
            ax_Ntou[-1].set_title(weeks[day_of_week] + ': ' + str(i) + ':00 (Type ' + str(g+1) + ')')
            ax_Ntou[-1].set_xlabel(r'Temperature (C$^o$)')
            ax_Ntou[-1].set_ylabel('Consumption (kWh)')
            ax_Ntou[-1].legend()
plt.tight_layout()
```

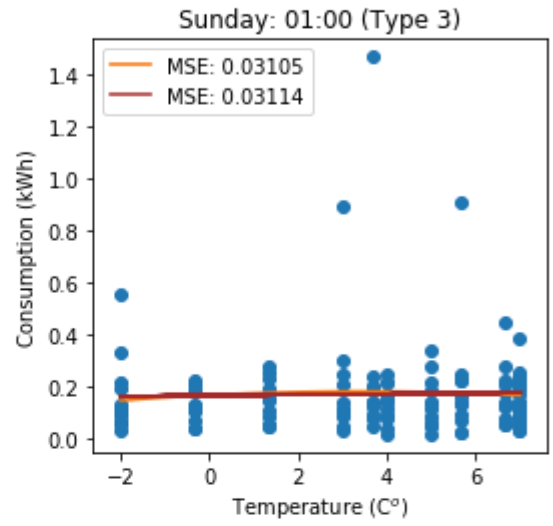

Sunday: 00:00 (Type 1)
MSE: 0.02812
MSE: 0.02813
Consumption (kWh)
Temperature (C°)

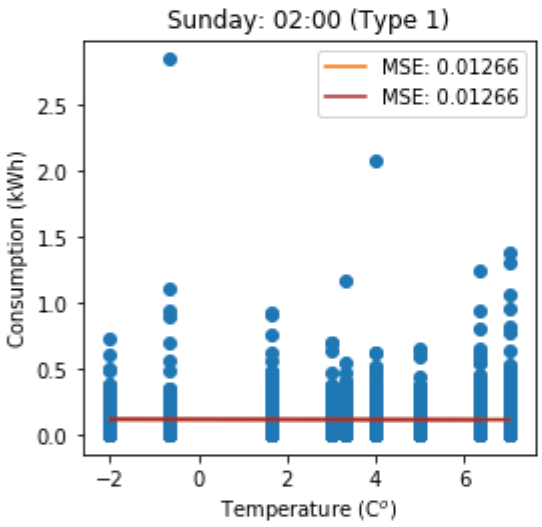

Sunday: 00:00 (Type 2)
MSE: 0.21646
MSE: 0.21669
Consumption (kWh)
Temperature (C°)

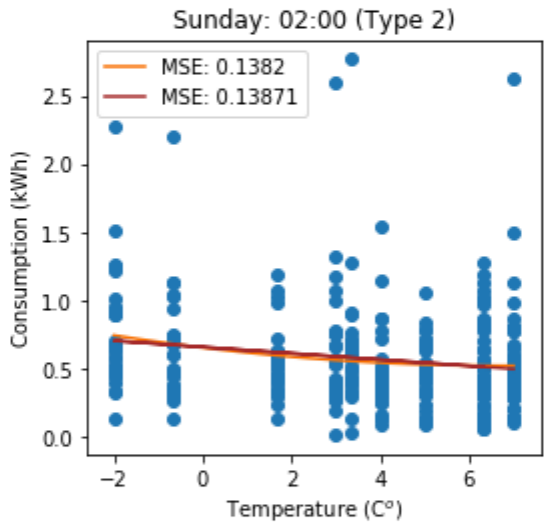

Sunday: 00:00 (Type 3)
MSE: 0.03487
MSE: 0.0349
Consumption (kWh)
Temperature (C°)

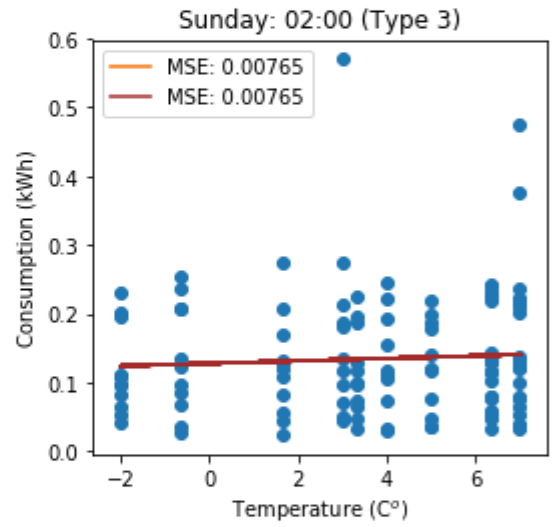

Sunday: 01:00 (Type 1)
MSE: 0.0188
MSE: 0.0188
Consumption (kWh)
Temperature (C°)

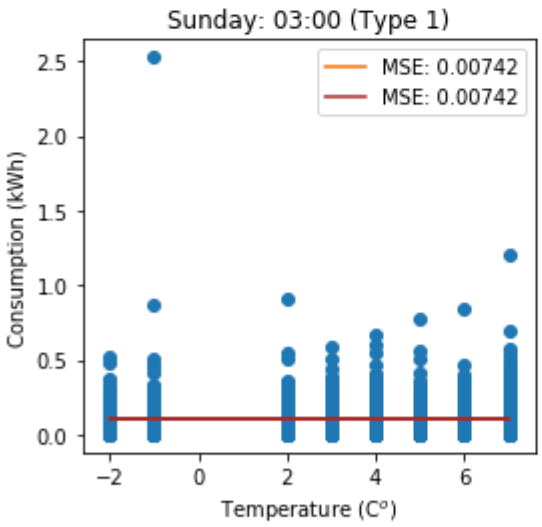

Sunday: 01:00 (Type 2)
MSE: 0.26364
MSE: 0.26618
Consumption (kWh)
Temperature (C°)

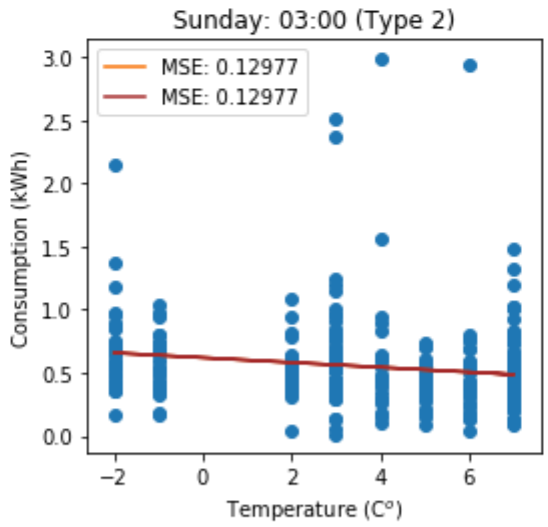

Sunday: 01:00 (Type 3)
MSE: 0.03105
MSE: 0.03114
Consumption (kWh)
Temperature (C°)

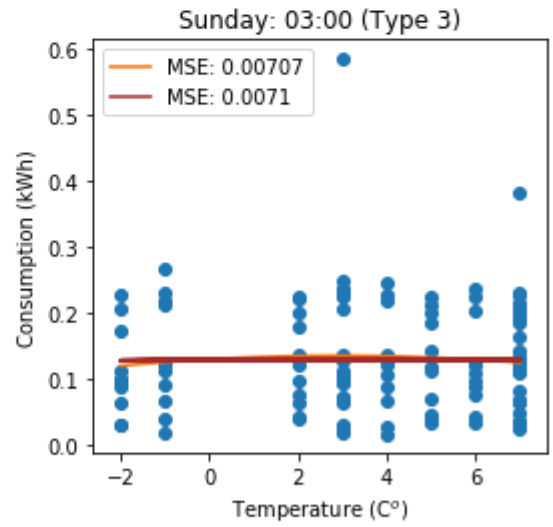

Sunday: 02:00 (Type 1)
MSE: 0.01266
MSE: 0.01266
Consumption (kWh)
Temperature (C°)

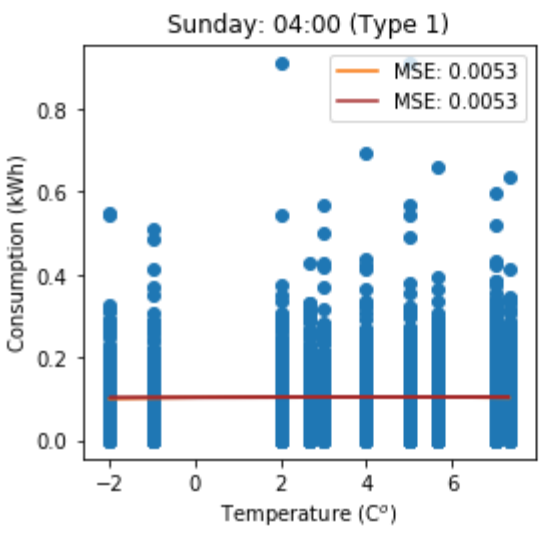

Sunday: 02:00 (Type 2)
MSE: 0.1382
MSE: 0.13871
Consumption (kWh)
Temperature (C°)

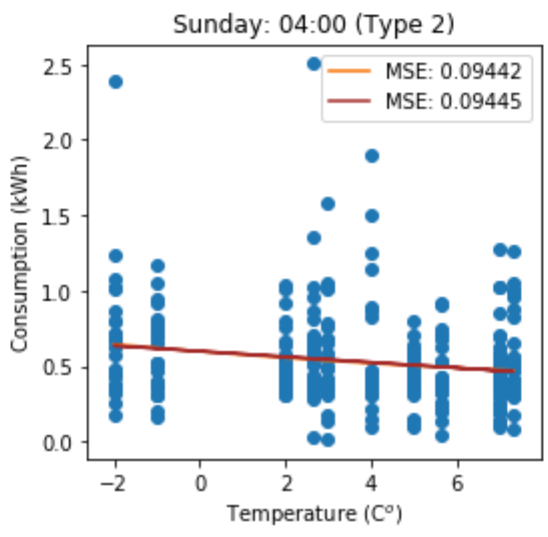

Sunday: 02:00 (Type 3)
MSE: 0.00765
MSE: 0.00765
Consumption (kWh)
Temperature (C°)

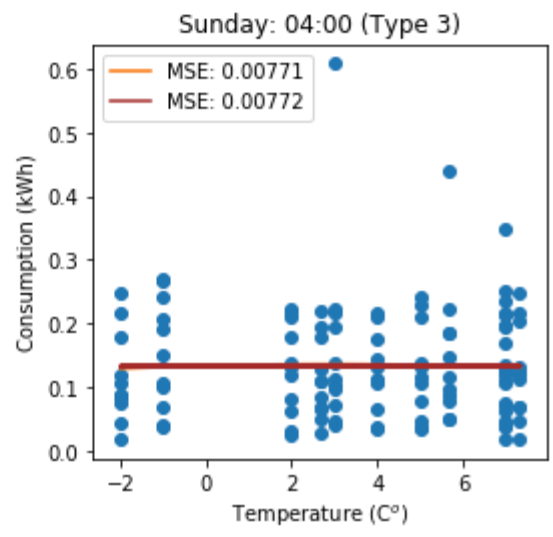

Sunday: 03:00 (Type 1)
MSE: 0.00742
MSE: 0.00742
Consumption (kWh)
Temperature (C°)

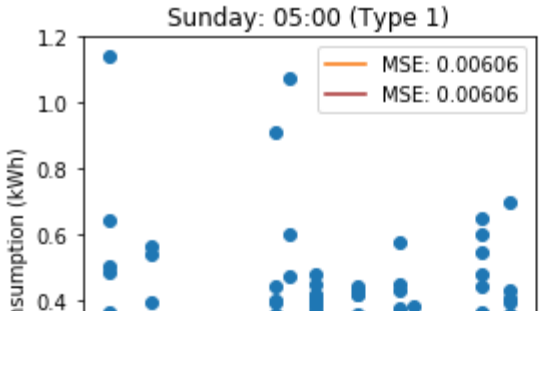

Sunday: 03:00 (Type 2)
MSE: 0.12977
MSE: 0.12977
Consumption (kWh)
Temperature (C°)

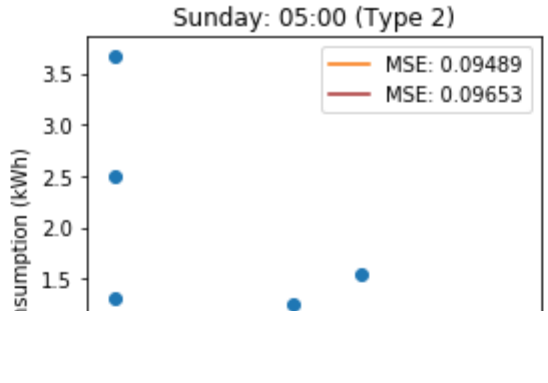

Sunday: 03:00 (Type 3)
MSE: 0.00707
MSE: 0.0071
Consumption (kWh)
Temperature (C°)

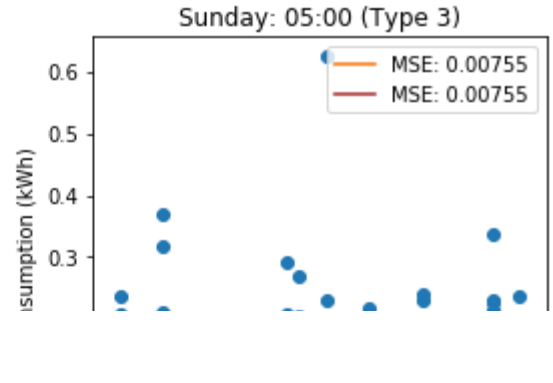

Sunday: 04:00 (Type 1)
MSE: 0.0053
MSE: 0.0053
Consumption (kWh)
Temperature (C°)

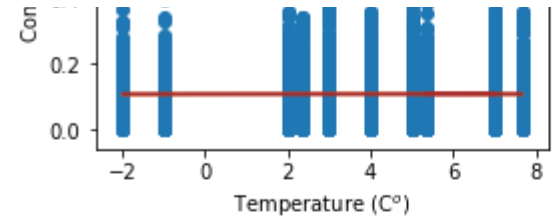

Sunday: 04:00 (Type 2)
MSE: 0.09442
MSE: 0.09445
Consumption (kWh)
Temperature (C°)

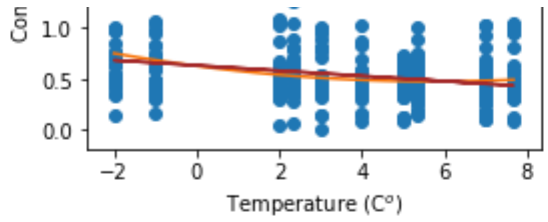

Sunday: 04:00 (Type 3)
MSE: 0.00771
MSE: 0.00772
Consumption (kWh)
Temperature (C°)

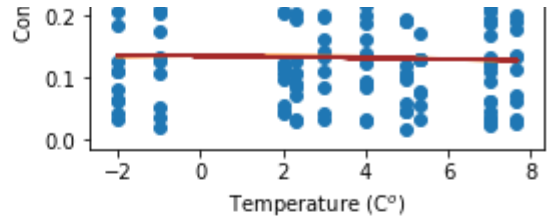

Sunday: 05:00 (Type 1)
MSE: 0.00606
MSE: 0.00606

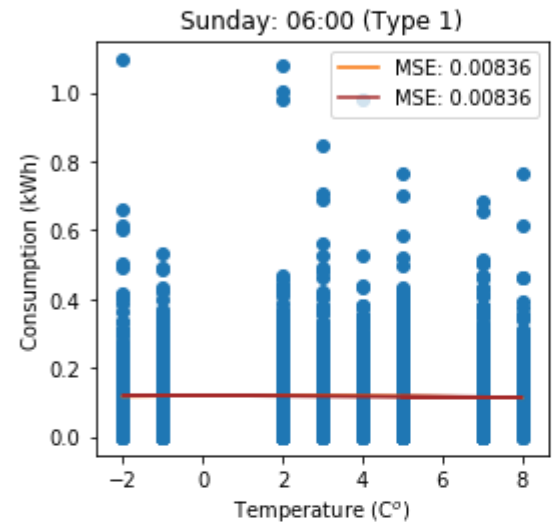

Sunday: 05:00 (Type 2)
MSE: 0.09489
MSE: 0.09653

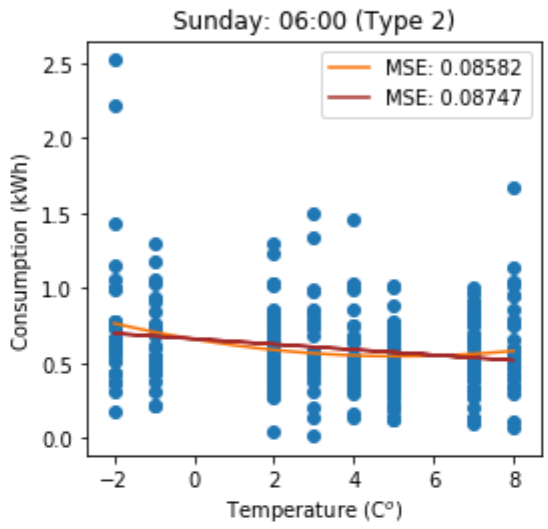

Sunday: 05:00 (Type 3)
MSE: 0.00755
MSE: 0.00755

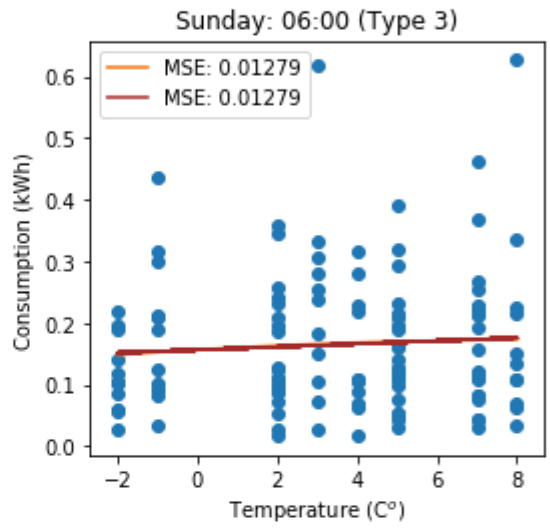

Temperature (C°)

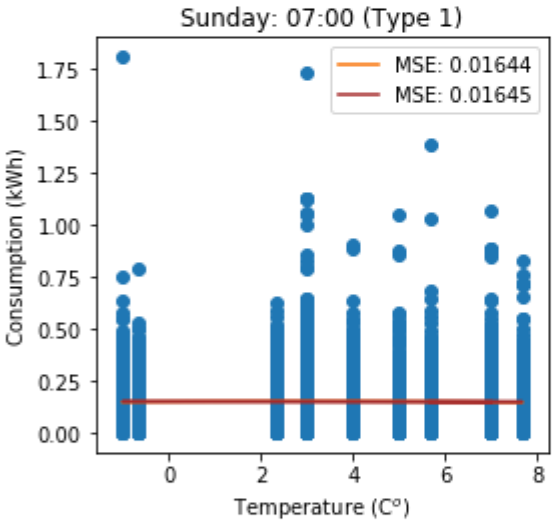

Temperature (C°)

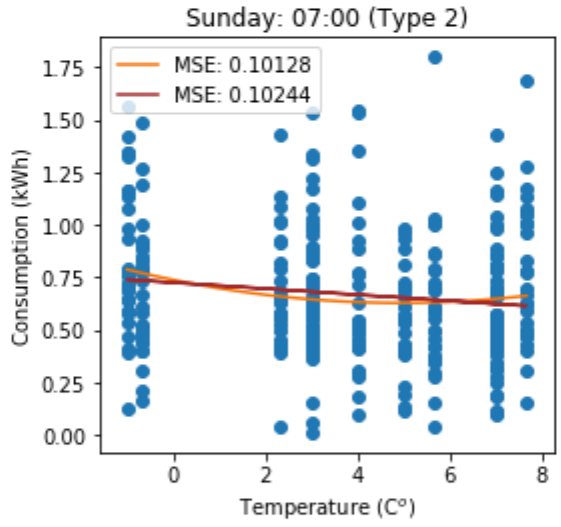

Temperature (C°)

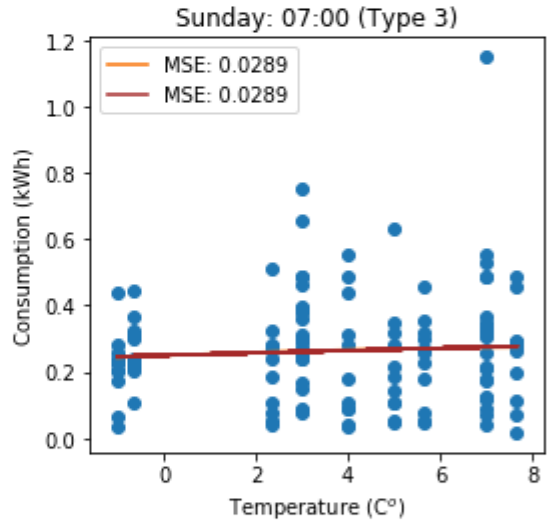

Sunday: 06:00 (Type 1)
MSE: 0.00836
MSE: 0.00836
Consumption (kWh)
Temperature (C°)

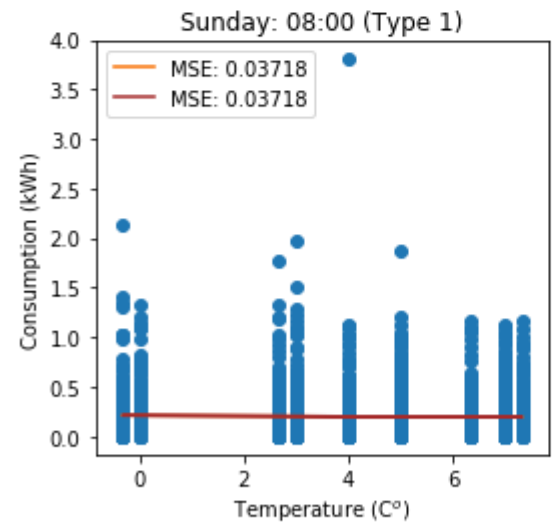

Sunday: 06:00 (Type 2)
MSE: 0.08582
MSE: 0.08747
Consumption (kWh)
Temperature (C°)

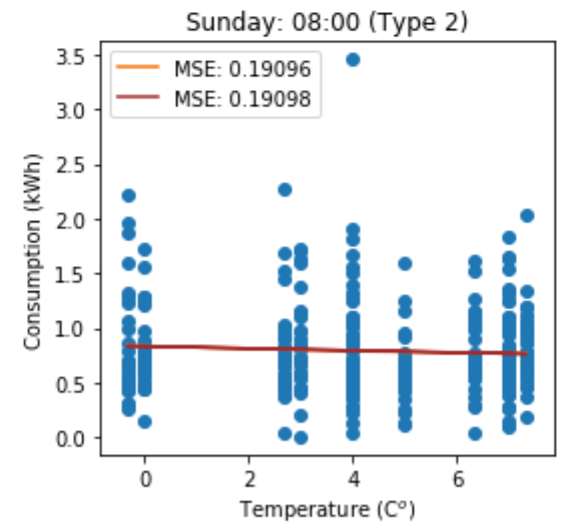

Sunday: 06:00 (Type 3)
MSE: 0.01279
MSE: 0.01279
Consumption (kWh)
Temperature (C°)

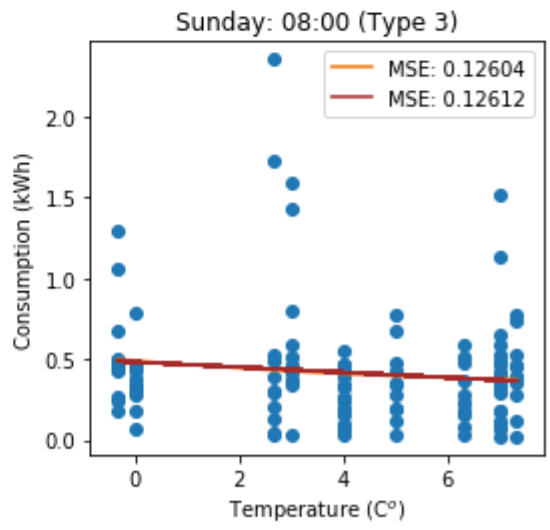

Sunday: 07:00 (Type 1)
MSE: 0.01644
MSE: 0.01645
Consumption (kWh)
Temperature (C°)

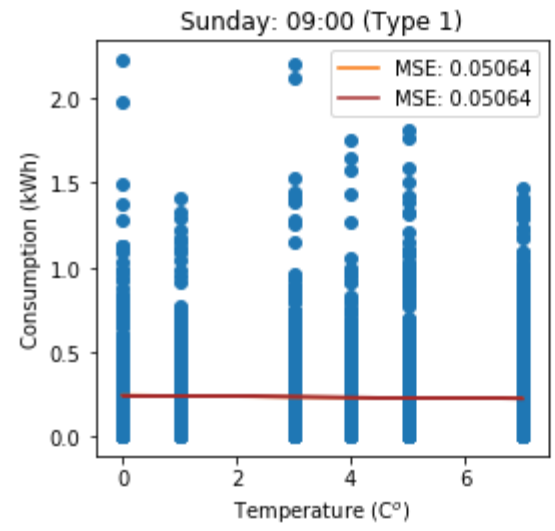

Sunday: 07:00 (Type 2)
MSE: 0.10128
MSE: 0.10244
Consumption (kWh)
Temperature (C°)

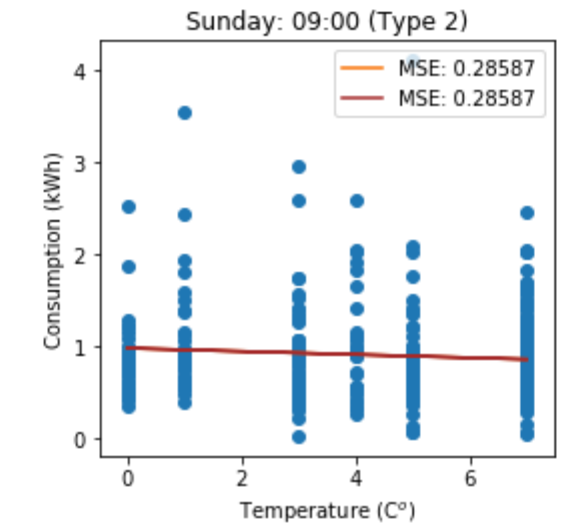

Sunday: 07:00 (Type 3)
MSE: 0.0289
MSE: 0.0289
Consumption (kWh)
Temperature (C°)

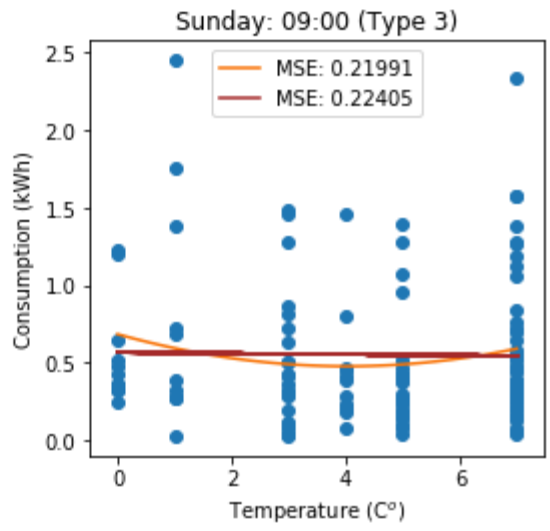

Sunday: 08:00 (Type 1)
MSE: 0.03718
MSE: 0.03718
Consumption (kWh)
Temperature (C°)

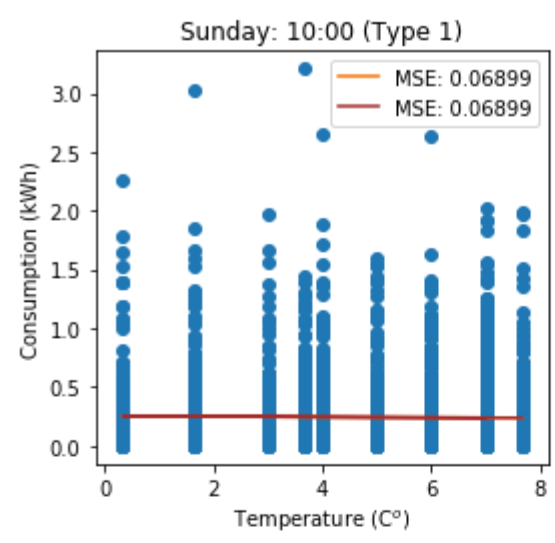

Sunday: 08:00 (Type 2)
MSE: 0.19096
MSE: 0.19098
Consumption (kWh)
Temperature (C°)

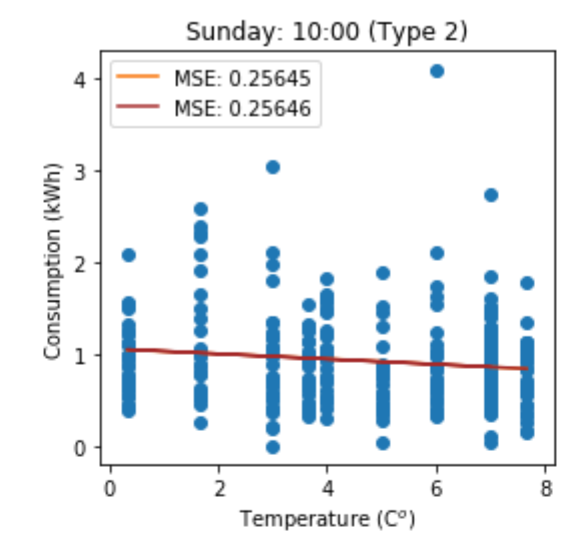

Sunday: 08:00 (Type 3)
MSE: 0.12604
MSE: 0.12612
Consumption (kWh)
Temperature (C°)

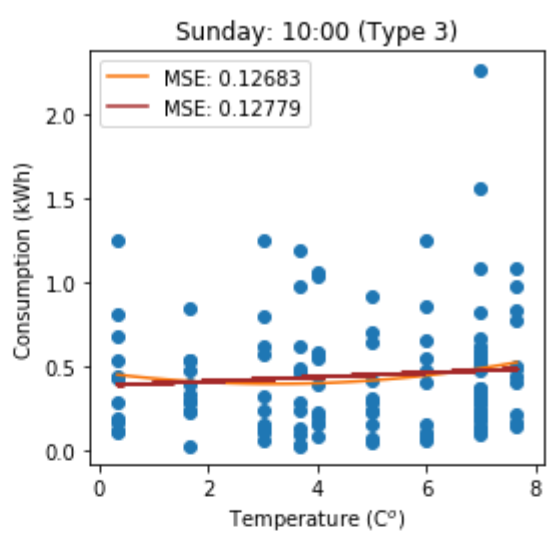

Sunday: 09:00 (Type 1)
MSE: 0.05064
MSE: 0.05064
Consumption (kWh)
Temperature (C°)

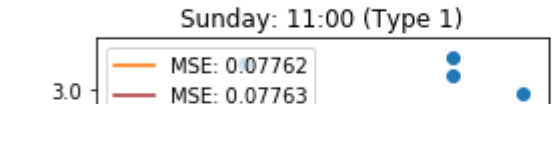

Sunday: 09:00 (Type 2)
MSE: 0.28587
MSE: 0.28587
Consumption (kWh)
Temperature (C°)

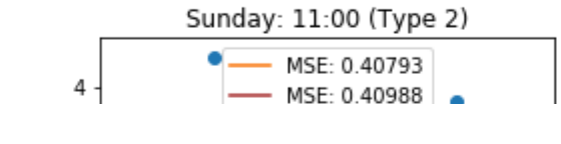

Sunday: 09:00 (Type 3)
MSE: 0.21991
MSE: 0.22405
Consumption (kWh)
Temperature (C°)

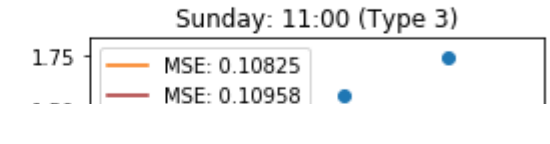

Sunday: 10:00 (Type 1)
MSE: 0.06899
MSE: 0.06899
Consumption (kWh)
Temperature (C°)

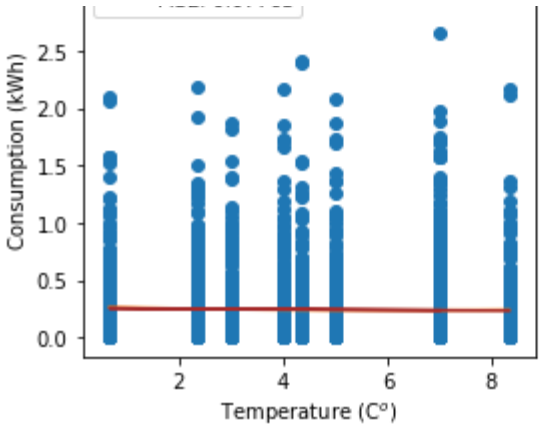

Sunday: 10:00 (Type 2)
MSE: 0.25645
MSE: 0.25646
Consumption (kWh)
Temperature (C°)

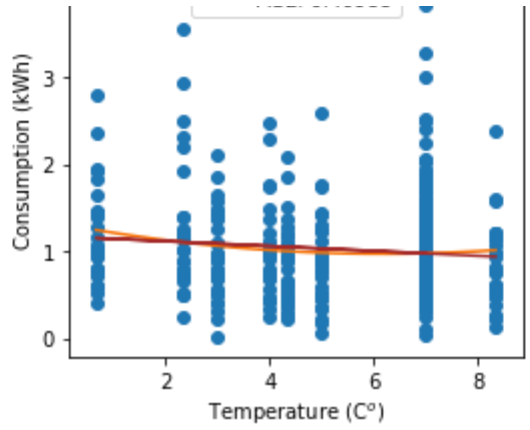

Sunday: 10:00 (Type 3)
MSE: 0.12683
MSE: 0.12779
Consumption (kWh)
Temperature (C°)

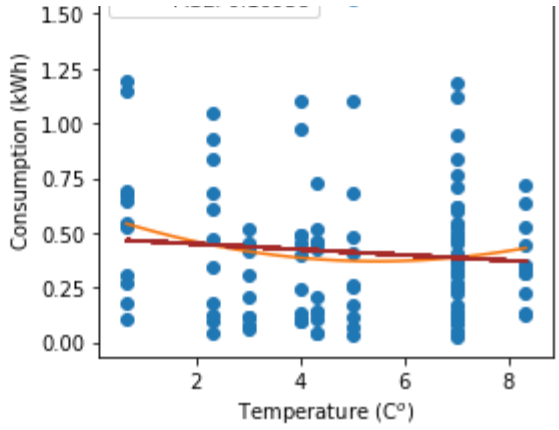

Sunday: 11:00 (Type 1)
MSE: 0.07762
MSE: 0.07763

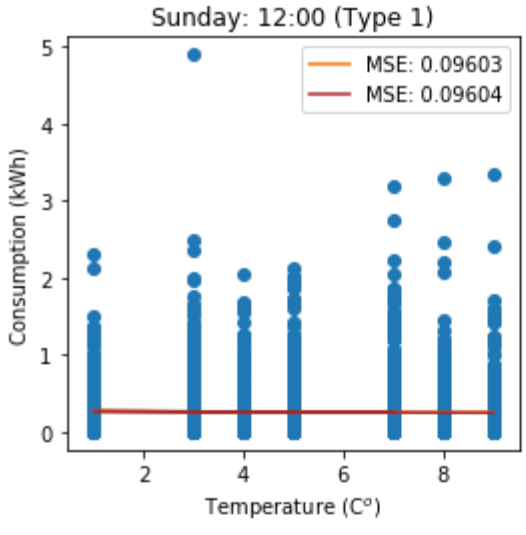

Sunday: 11:00 (Type 2)
MSE: 0.40793
MSE: 0.40988

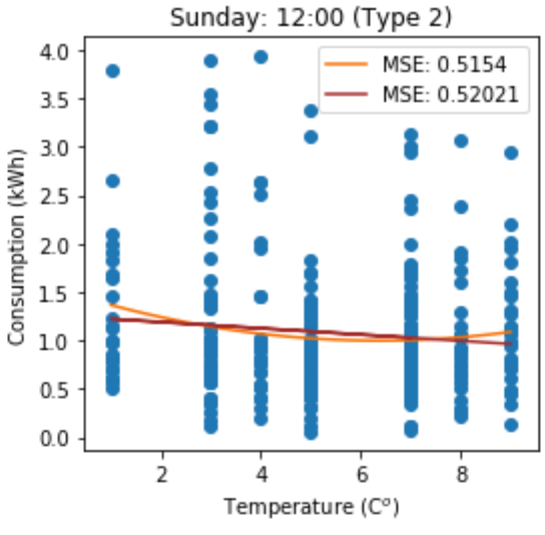

Sunday: 11:00 (Type 3)
MSE: 0.10825
MSE: 0.10958

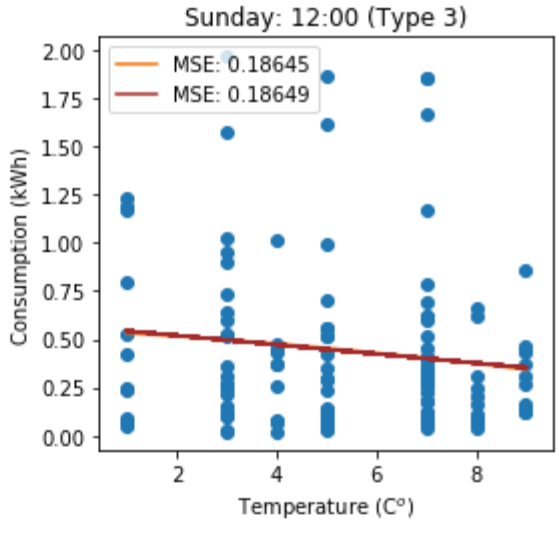

Consumption (kWh)
Temperature (C°)

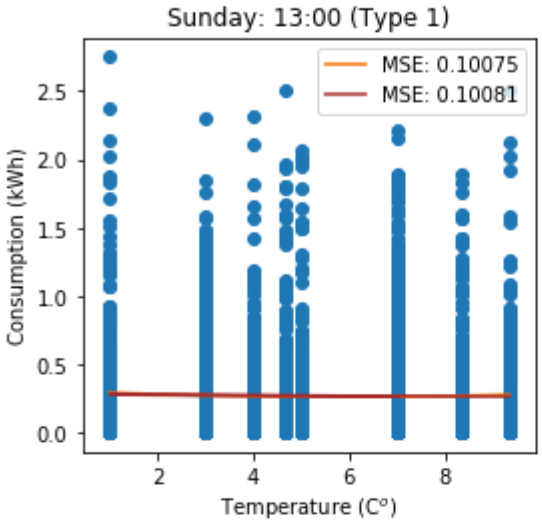

Consumption (kWh)
Temperature (C°)

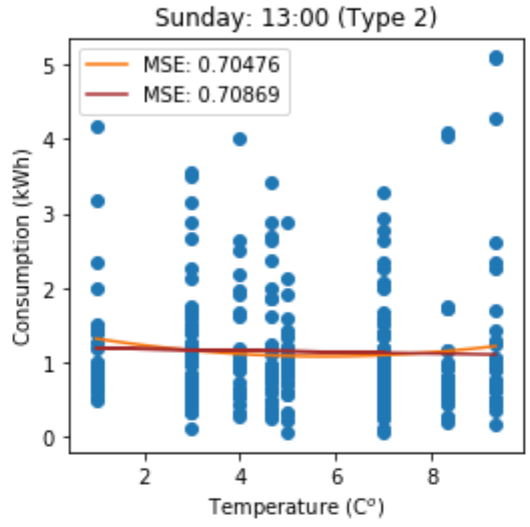

Consumption (kWh)
Temperature (C°)

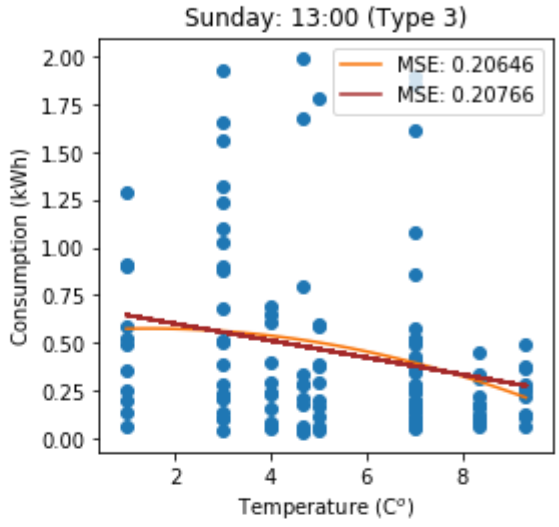

Sunday: 12:00 (Type 1)
MSE: 0.09603
MSE: 0.09604
Consumption (kWh)
Temperature (C°)

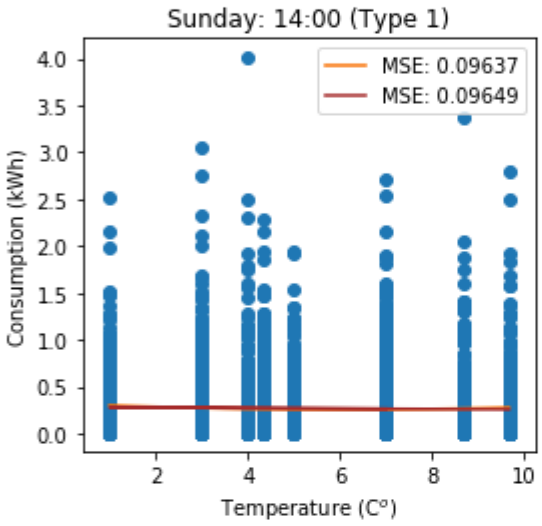

Sunday: 12:00 (Type 2)
MSE: 0.5154
MSE: 0.52021
Consumption (kWh)
Temperature (C°)

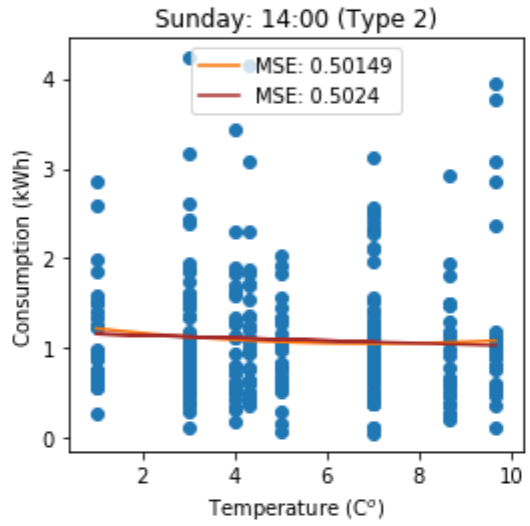

Sunday: 12:00 (Type 3)
MSE: 0.18645
MSE: 0.18649
Consumption (kWh)
Temperature (C°)

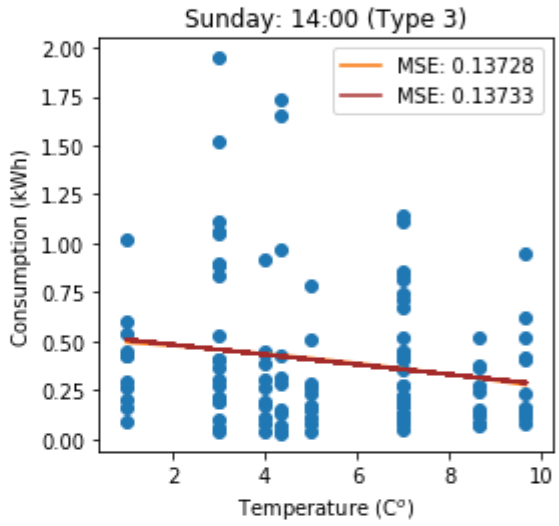

Sunday: 13:00 (Type 1)
MSE: 0.10075
MSE: 0.10081
Consumption (kWh)
Temperature (C°)

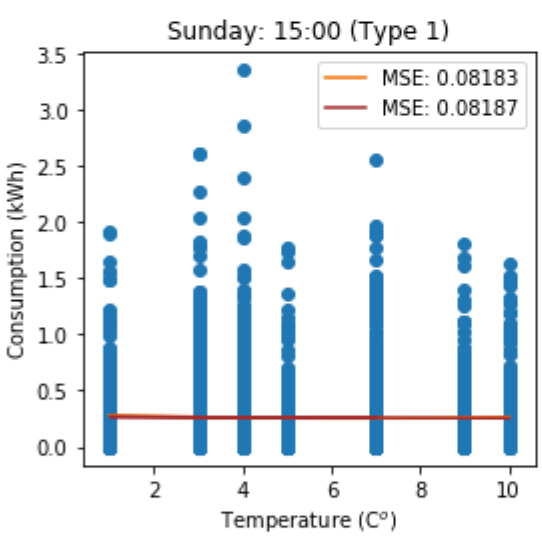

Sunday: 13:00 (Type 2)
MSE: 0.70476
MSE: 0.70869
Consumption (kWh)
Temperature (C°)

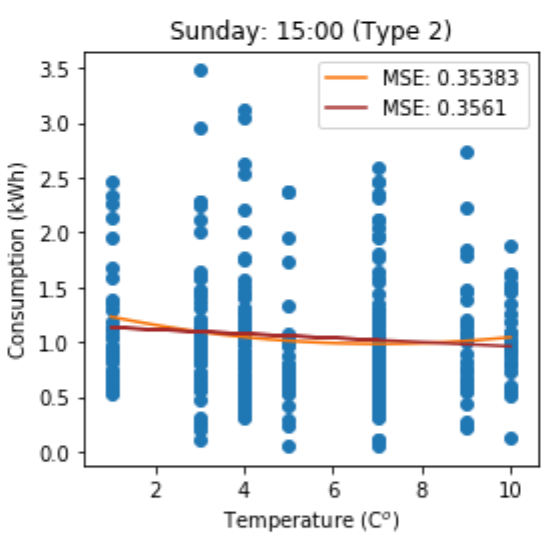

Sunday: 13:00 (Type 3)
MSE: 0.20646
MSE: 0.20766
Consumption (kWh)
Temperature (C°)

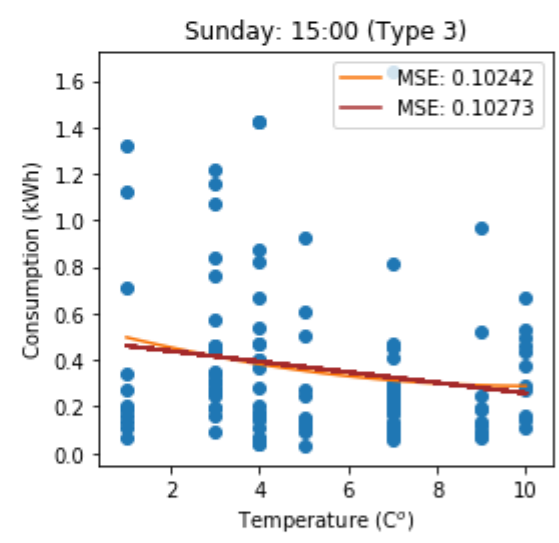

Sunday: 14:00 (Type 1)
MSE: 0.09637
MSE: 0.09649
Consumption (kWh)
Temperature (C°)

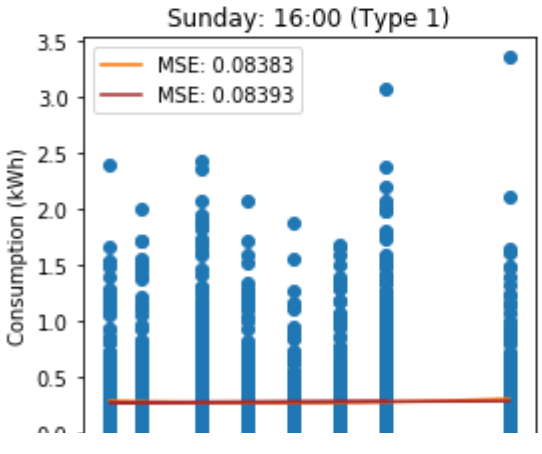

Sunday: 14:00 (Type 2)
MSE: 0.50149
MSE: 0.5024
Consumption (kWh)
Temperature (C°)

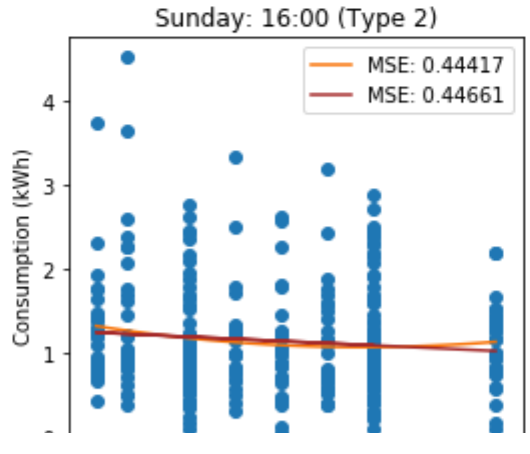

Sunday: 14:00 (Type 3)
MSE: 0.13728
MSE: 0.13733
Consumption (kWh)
Temperature (C°)

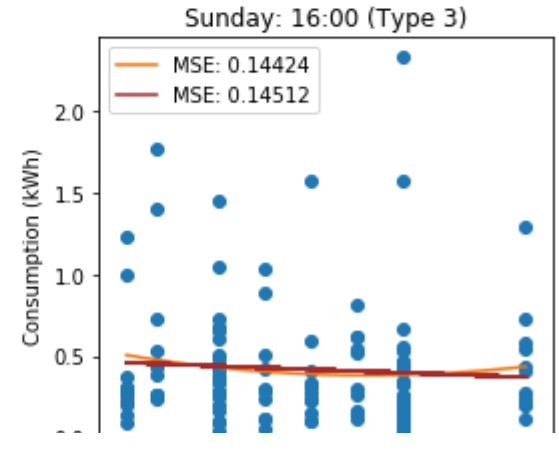

Sunday: 15:00 (Type 1)
MSE: 0.08183
MSE: 0.08187
Consumption (kWh)
Temperature (C°)

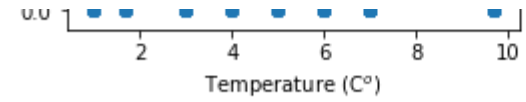

Sunday: 15:00 (Type 2)
MSE: 0.35383
MSE: 0.3561
Consumption (kWh)
Temperature (C°)

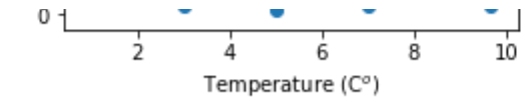

Sunday: 15:00 (Type 3)
MSE: 0.10242
MSE: 0.10273
Consumption (kWh)
Temperature (C°)

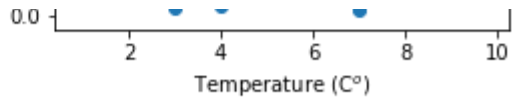

Sunday: 16:00 (Type 1)
MSE: 0.08383
MSE: 0.08393
Consumption (kWh)

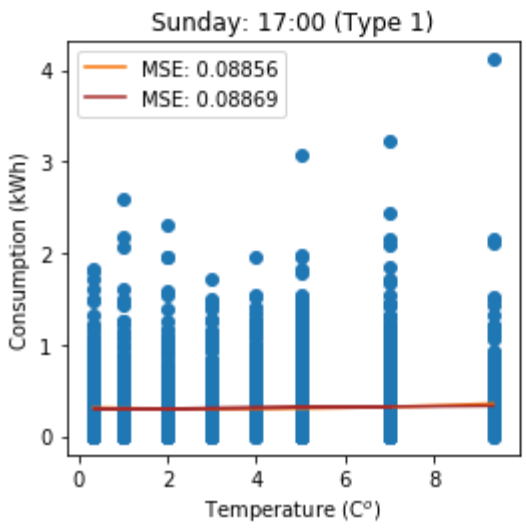

Sunday: 16:00 (Type 2)
MSE: 0.44417
MSE: 0.44661
Consumption (kWh)

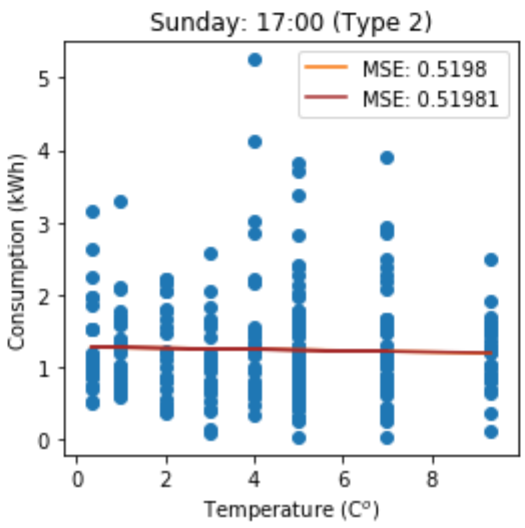

Sunday: 16:00 (Type 3)
MSE: 0.14424
MSE: 0.14512
Consumption (kWh)

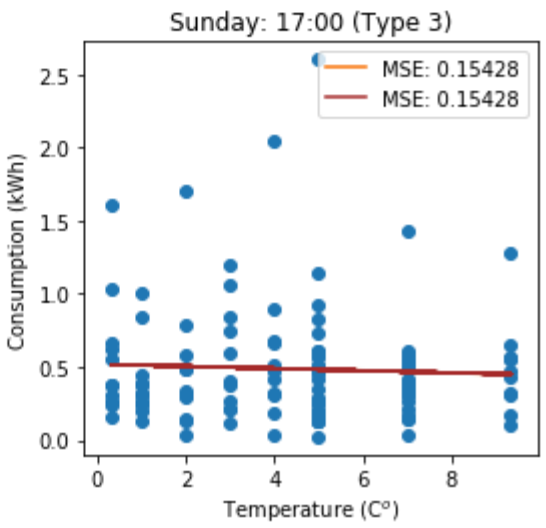

Temperature (C°)

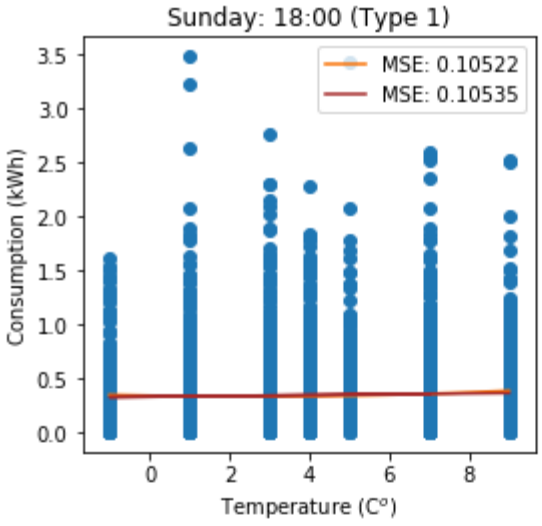

Temperature (C°)

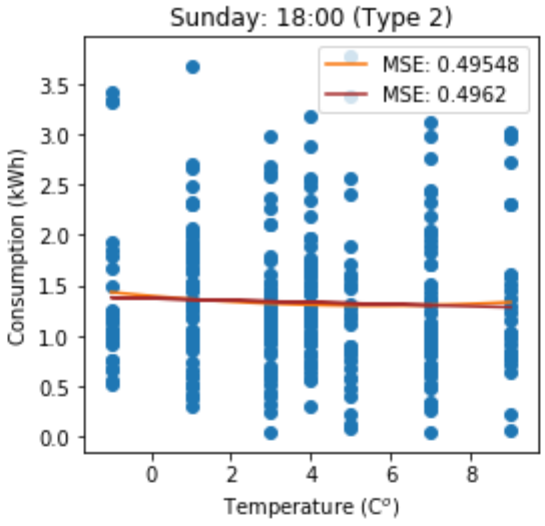

Temperature (C°)

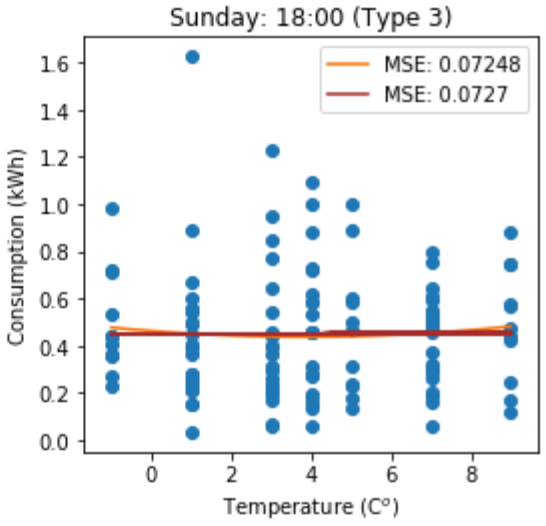

Sunday: 17:00 (Type 1)
MSE: 0.08856
MSE: 0.08869
Consumption (kWh)
Temperature (C°)

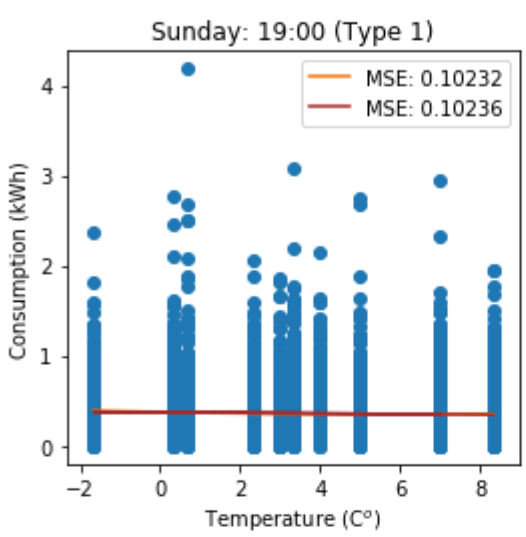

Sunday: 17:00 (Type 2)
MSE: 0.5198
MSE: 0.51981
Consumption (kWh)
Temperature (C°)

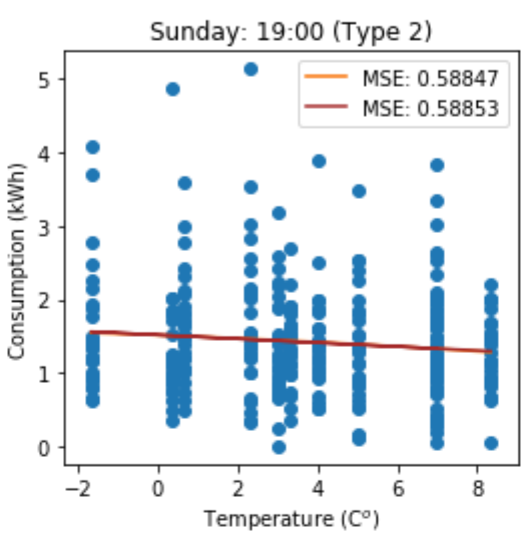

Sunday: 17:00 (Type 3)
MSE: 0.15428
MSE: 0.15428
Consumption (kWh)
Temperature (C°)

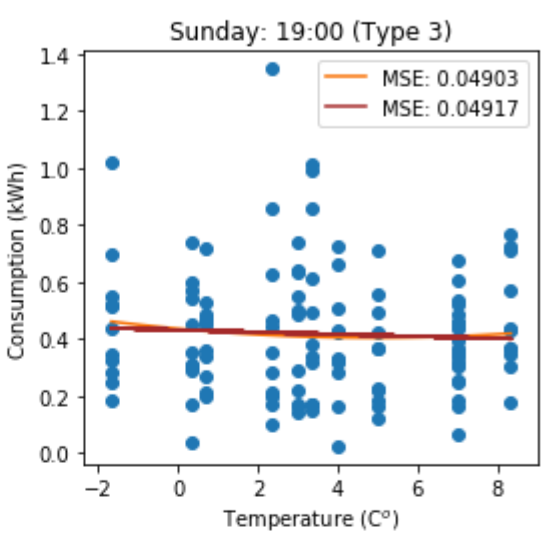

Sunday: 18:00 (Type 1)
MSE: 0.10522
MSE: 0.10535
Consumption (kWh)
Temperature (C°)

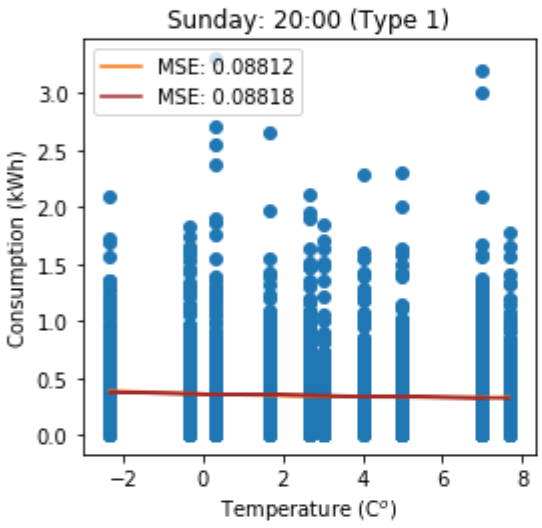

Sunday: 18:00 (Type 2)
MSE: 0.49548
MSE: 0.4962
Consumption (kWh)
Temperature (C°)

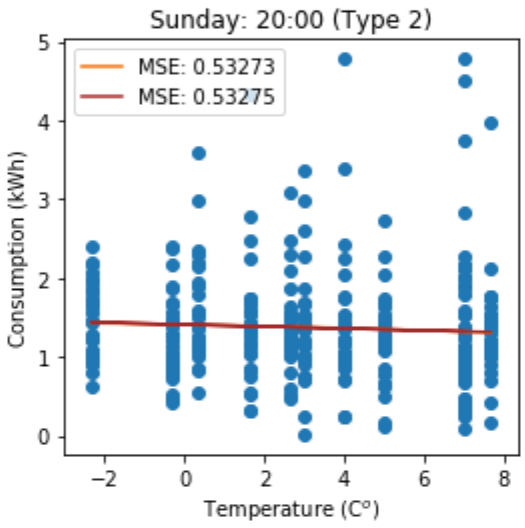

Sunday: 18:00 (Type 3)
MSE: 0.07248
MSE: 0.0727
Consumption (kWh)
Temperature (C°)

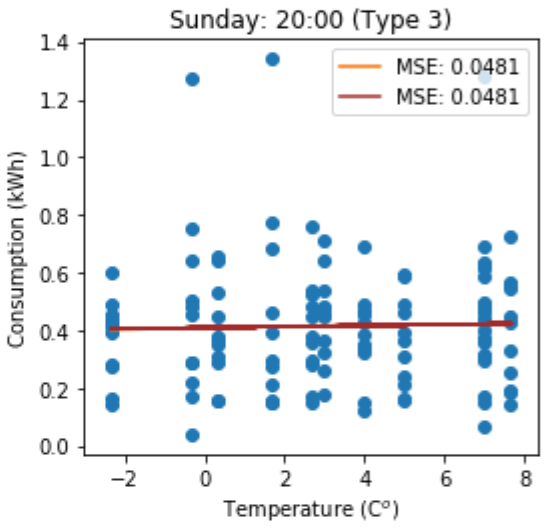

Sunday: 19:00 (Type 1)
MSE: 0.10232
MSE: 0.10236
Consumption (kWh)
Temperature (C°)

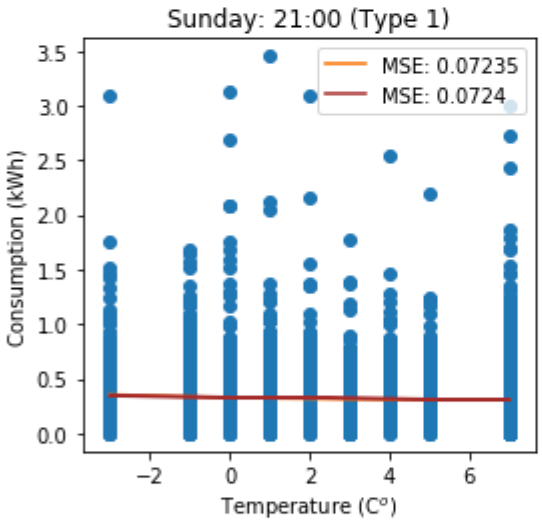

Sunday: 19:00 (Type 2)
MSE: 0.58847
MSE: 0.58853
Consumption (kWh)
Temperature (C°)

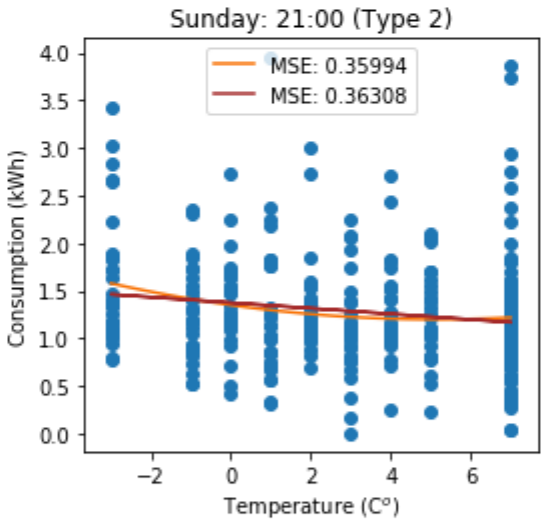

Sunday: 19:00 (Type 3)
MSE: 0.04903
MSE: 0.04917
Consumption (kWh)
Temperature (C°)

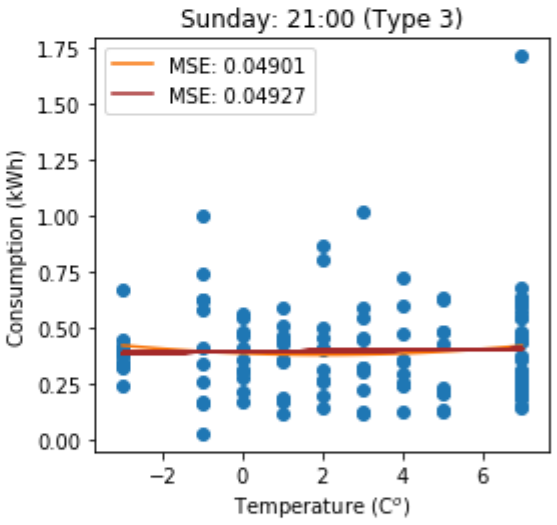

Sunday: 20:00 (Type 1)
MSE: 0.08812
MSE: 0.08818
Consumption (kWh)
Temperature (C°)

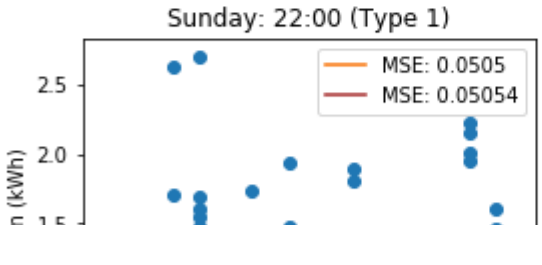

Sunday: 20:00 (Type 2)
MSE: 0.53273
MSE: 0.53275
Consumption (kWh)
Temperature (C°)

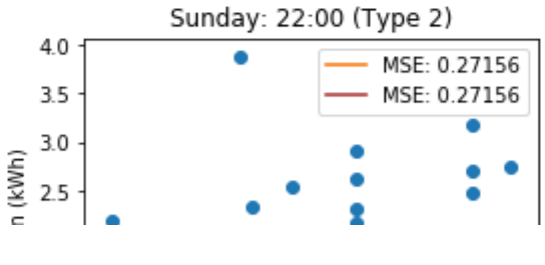

Sunday: 20:00 (Type 3)
MSE: 0.0481
MSE: 0.0481
Consumption (kWh)
Temperature (C°)

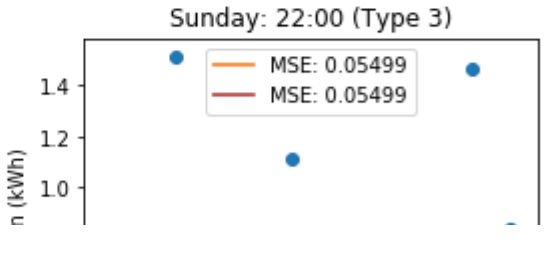

Sunday: 21:00 (Type 1)
MSE: 0.07235
MSE: 0.0724
Consumption (kWh)
Temperature (C°)

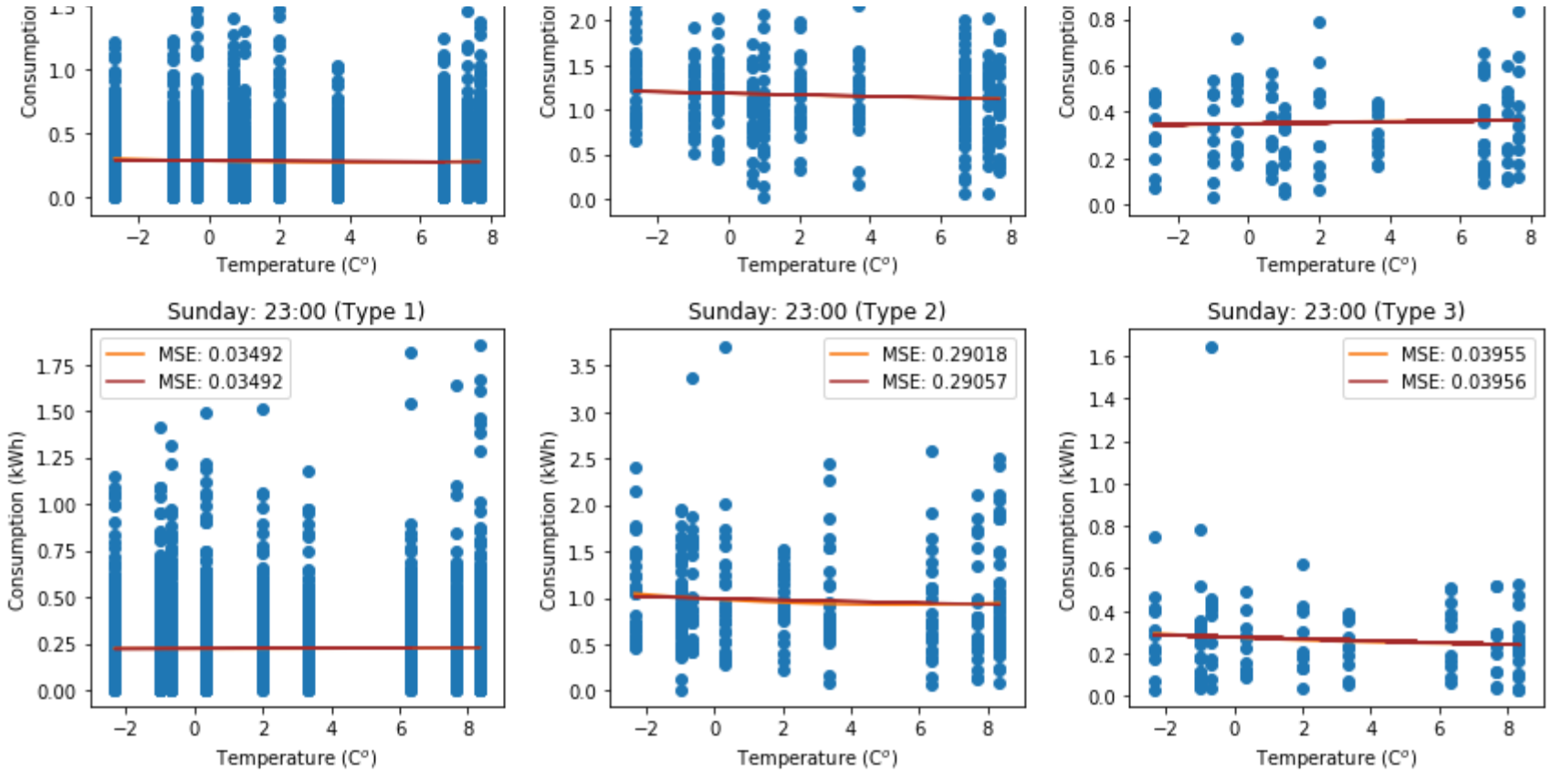

Sunday: 21:00 (Type 2)
MSE: 0.35994
MSE: 0.36308
Consumption (kWh)
Temperature (C°)


Sunday: 21:00 (Type 3)
MSE: 0.04901
MSE: 0.04927
Consumption (kWh)
Temperature (C°)


Sunday: 22:00 (Type 1)
MSE: 0.0505
MSE: 0.05054

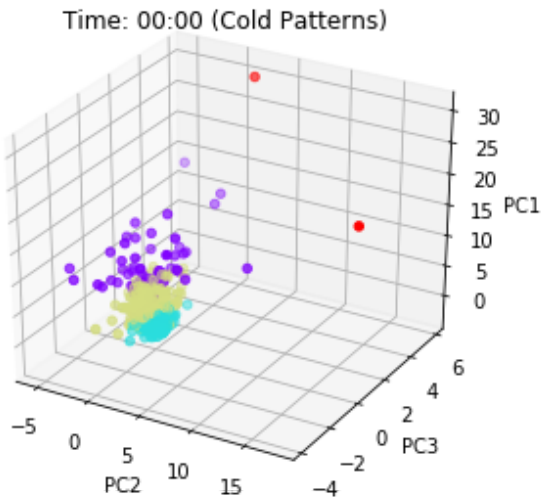

Sunday: 22:00 (Type 2)
MSE: 0.27156
MSE: 0.27156

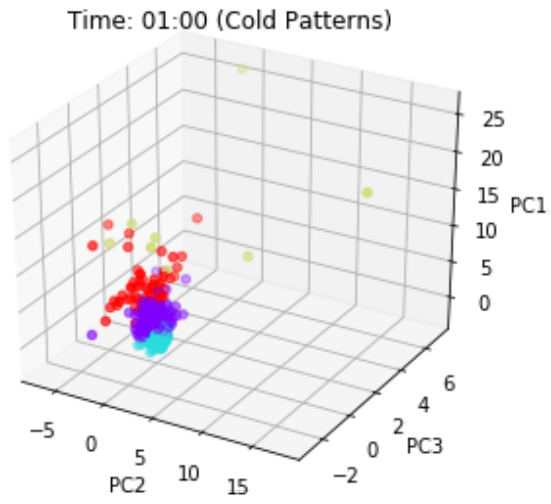

Sunday: 22:00 (Type 3)
MSE: 0.05499
MSE: 0.05499

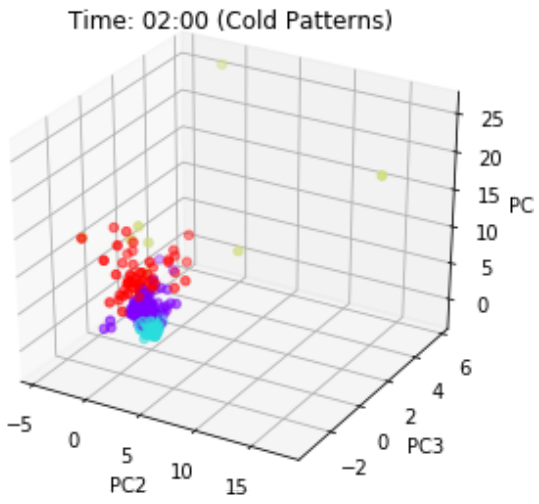


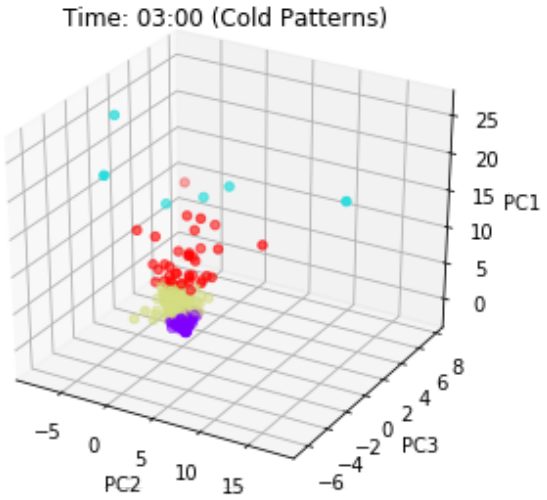
What if we use 4 cluseters:

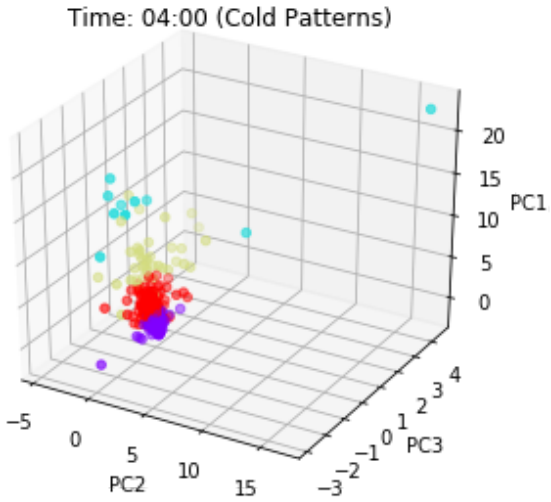
In [373]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = 4).fit(principleDf_nstd.iloc[:,:3])
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10), cm
ap = 'rainbow')
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold Patterns)')
        ax.set_xlabel('PC2')
        ax.set_ylabel('PC3')
        ax.set_zlabel('PC1')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        dayofWeek= pd.DataFrame(pd.to_datetime(x.GMT).dt.dayofweek)
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = 4).fit(principleDf_nstd.iloc[:,:3])
        ax.scatter(principleDf_nstd[1], principleDf_nstd[2], principleDf_nstd[0], c = (kmeans.labels_*10), cm
ap = 'rainbow')
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold Patterns)')
        ax.set_xlabel('PC2')
        ax.set_ylabel('PC3')
        ax.set_zlabel('PC1')
plt.tight_layout()
```

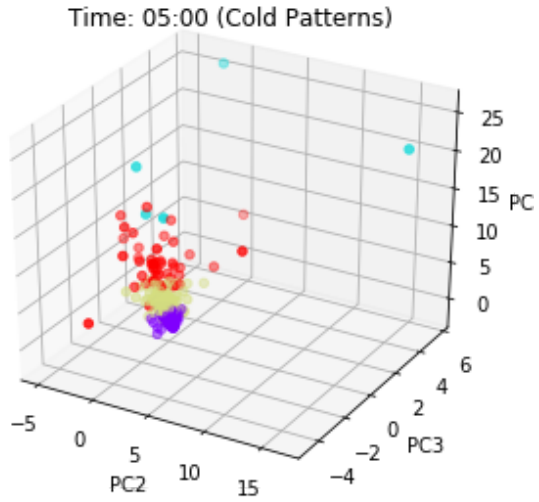

Time: 00:00 (Cold Patterns)
PC1
PC2
PC3

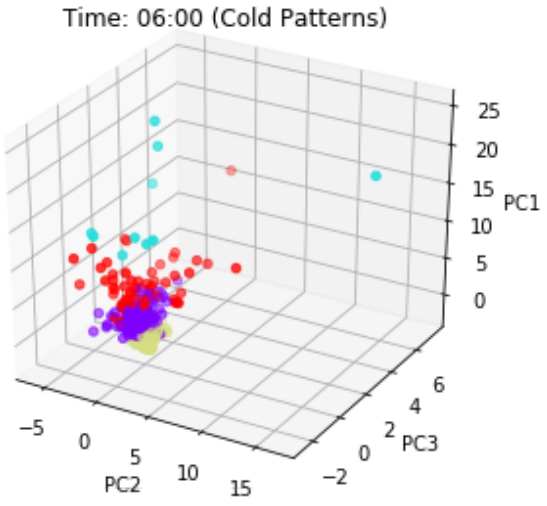

Time: 01:00 (Cold Patterns)
PC1
PC2
PC3

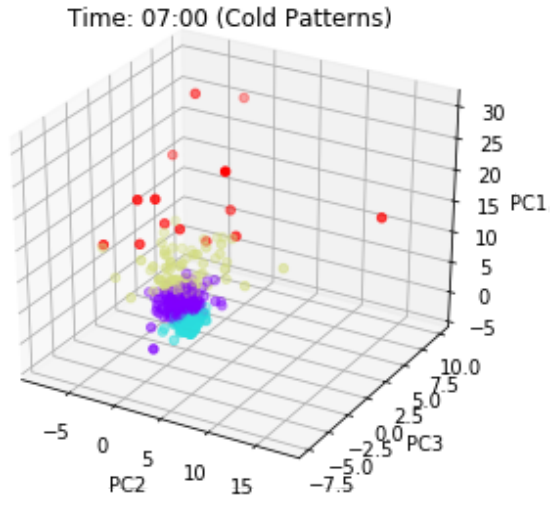

Time: 02:00 (Cold Patterns)
PC
PC2
PC3

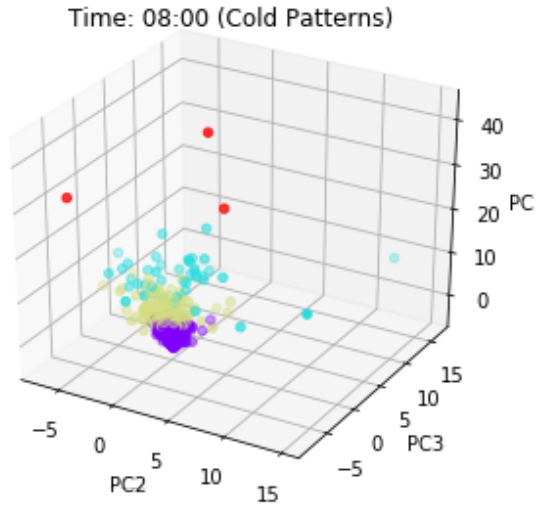

Time: 03:00 (Cold Patterns)
PC1
PC2
PC3

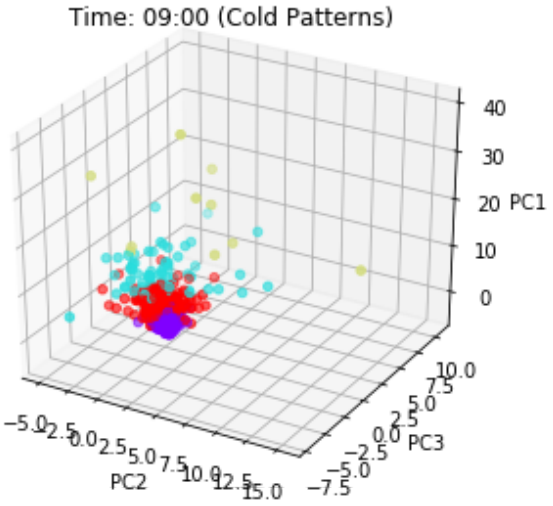

Time: 04:00 (Cold Patterns)
PC1
PC2
PC3

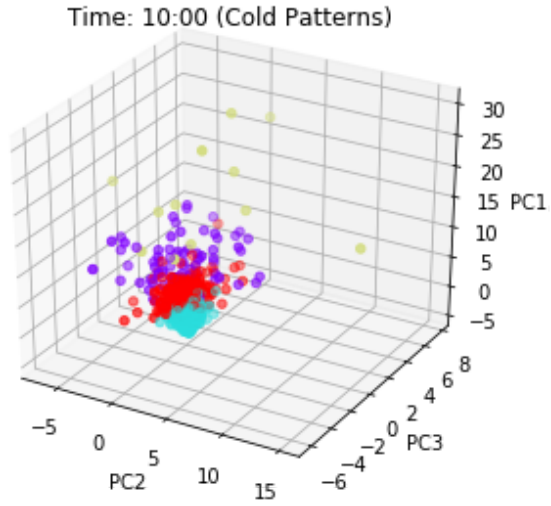

Time: 05:00 (Cold Patterns)
PC
PC2
PC3

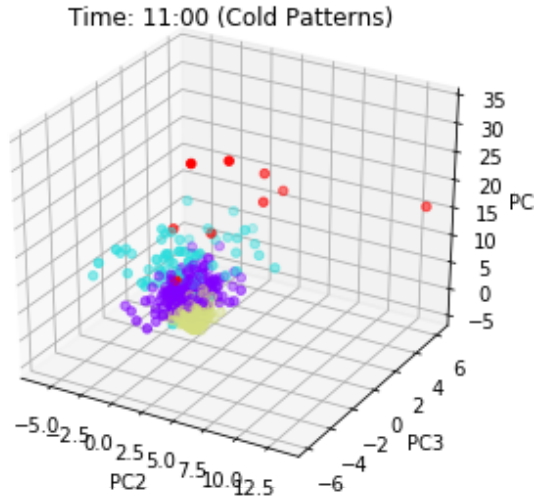

Time: 06:00 (Cold Patterns)
PC1
PC2
PC3

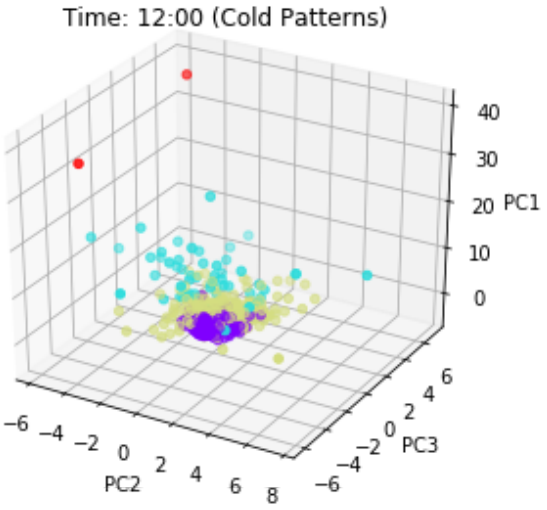

Time: 07:00 (Cold Patterns)
PC1
PC2
PC3

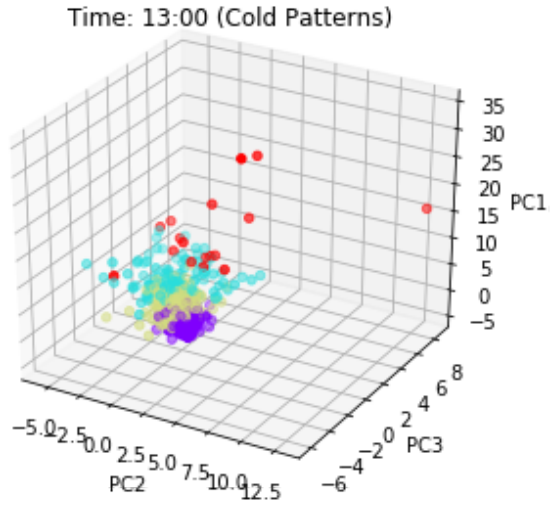

Time: 08:00 (Cold Patterns)
PC
PC2
PC3

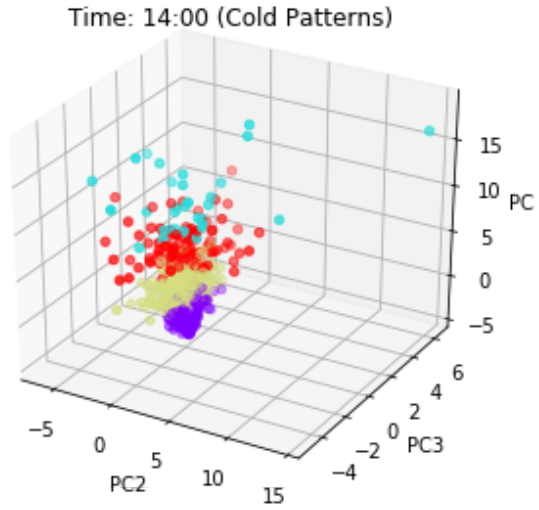

Time: 09:00 (Cold Patterns)
PC1
PC2
PC3

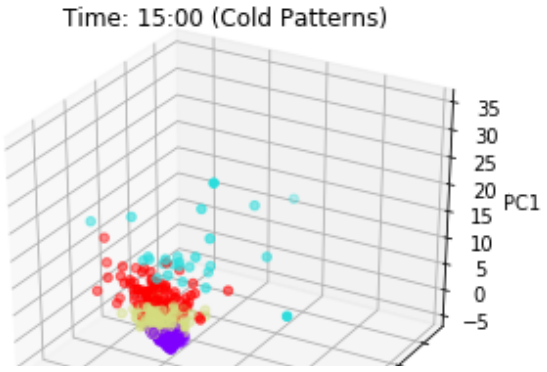

Time: 10:00 (Cold Patterns)
PC1
PC2
PC3

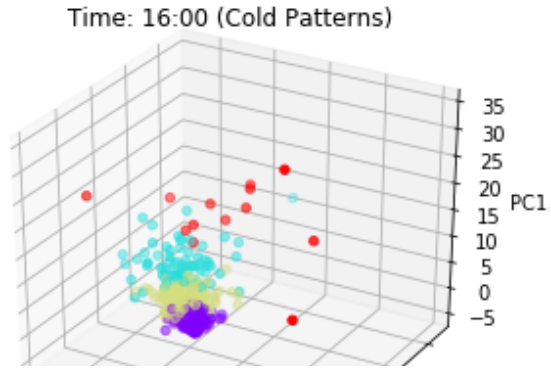

Time: 11:00 (Cold Patterns)
PC
PC2
PC3

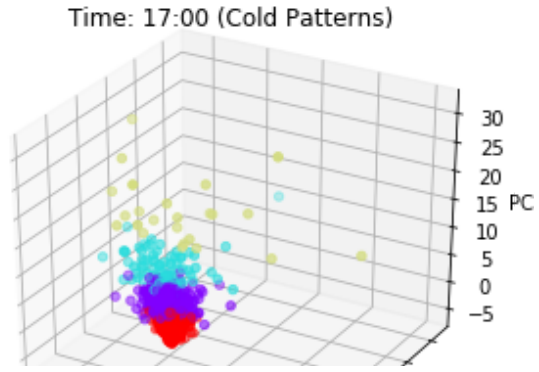

Time: 12:00 (Cold Patterns)
PC1
PC2
PC3

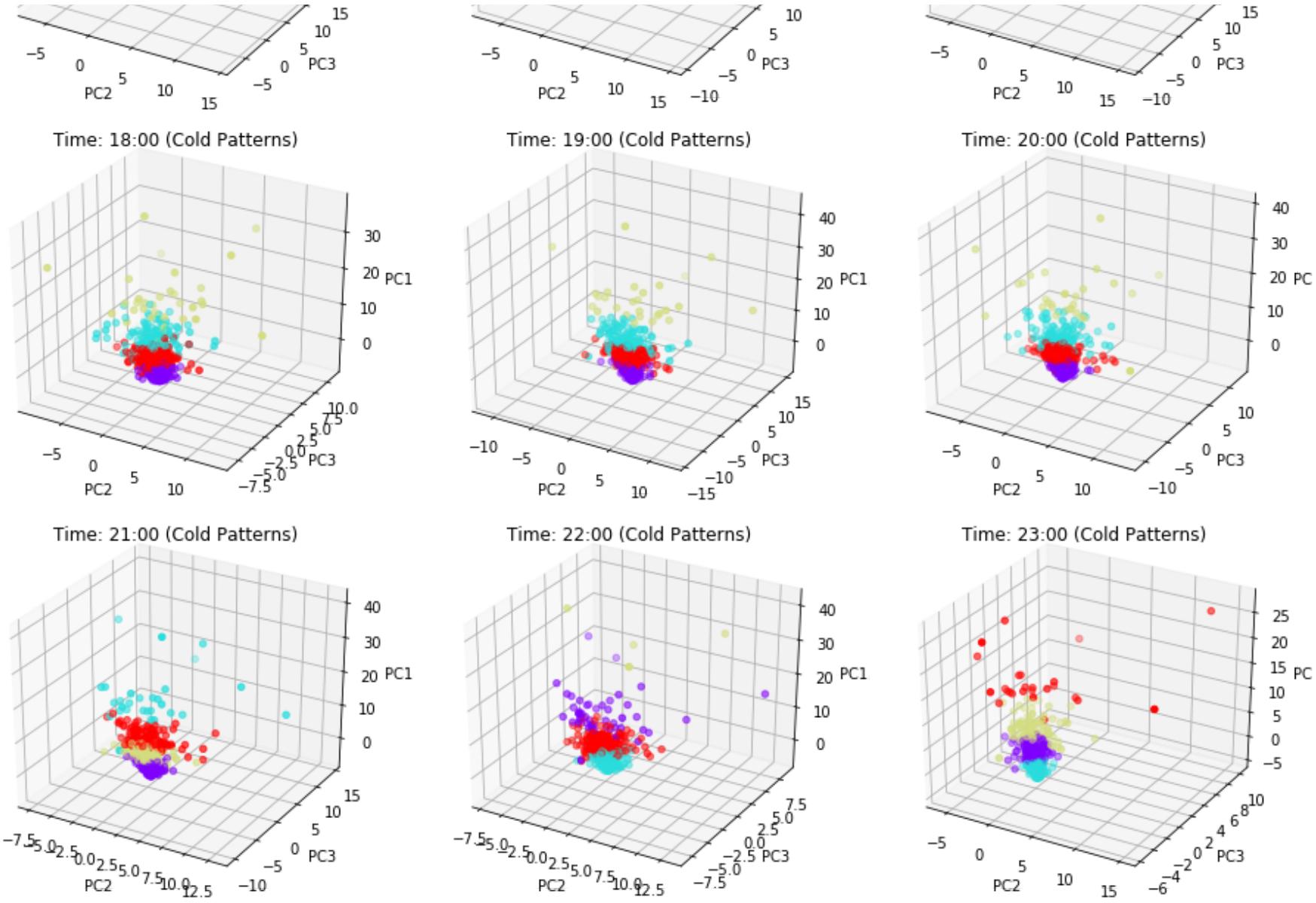

Time: 13:00 (Cold Patterns)
PC1
PC2
PC3


Time: 14:00 (Cold Patterns)
PC
PC2
PC3


Time: 15:00 (Cold Patterns)
PC1

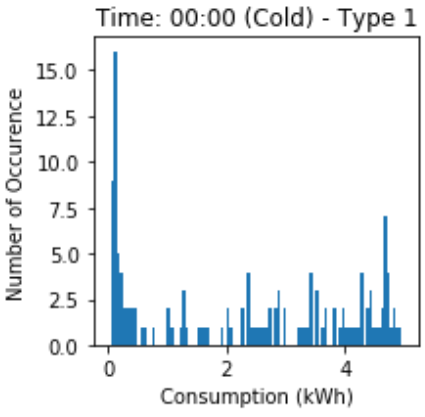

Time: 16:00 (Cold Patterns)
PC1

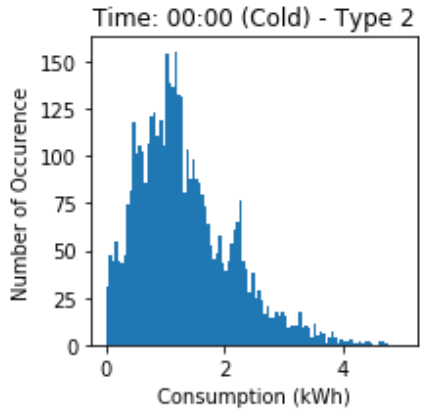

Time: 17:00 (Cold Patterns)
PC

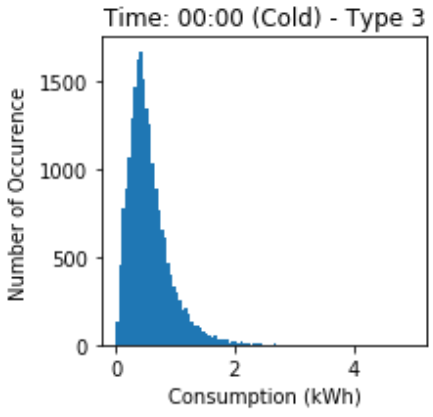


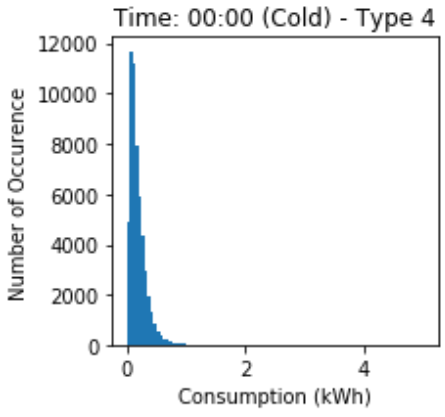
What we find from the k-mean clustering is that if we use the criteria paper 1 used, to find out the minimum number of clustering that violation of that theata threshold conditions is no more than 5%, then we will end up with about 200 clusters, which is too many to be a managable amount for system operators to classify the user behaviors. Here for simplisity, we think 3 group is easier to handle and to understand, we will take a look at the temperature, consumption distribution, day of a week to name these 3 groups accordingly.

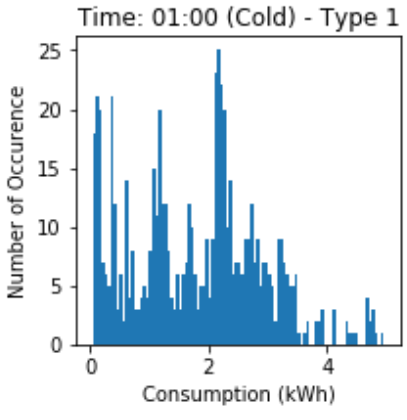
In [377]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 4 # cluster numbers
cons_type = ['Type 1', 'Type 2', 'Type 3', 'Type 4']
fig_all = plt.figure(figsize = (13,72))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1)
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            ax.hist(t2[t2.columns[1:-1]].values.flatten(), bins=100, range=(0,5))
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1)
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            ax.hist(t2[t2.columns[1:-1]].values.flatten(), bins=100, range=(0,5))
            ax.set_title('Time: ' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Number of Occurence')
plt.tight_layout()
```

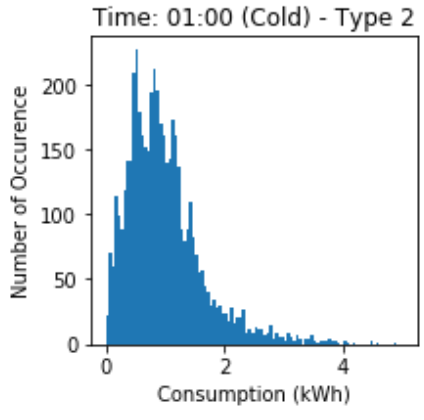

Time: 00:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

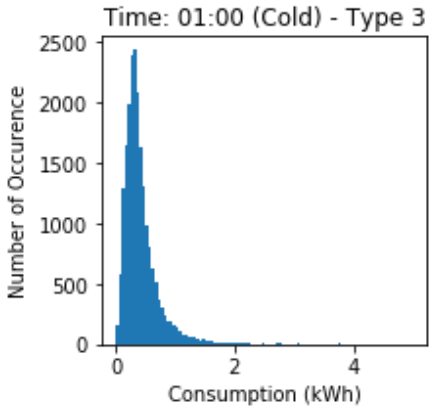

Time: 00:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

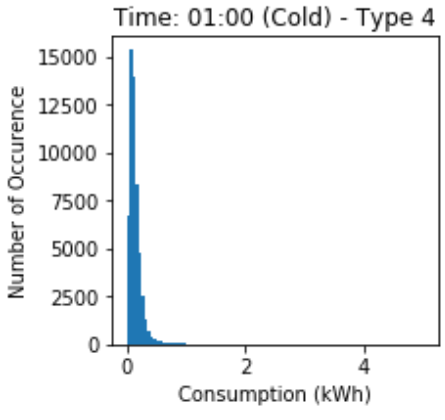

Time: 00:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

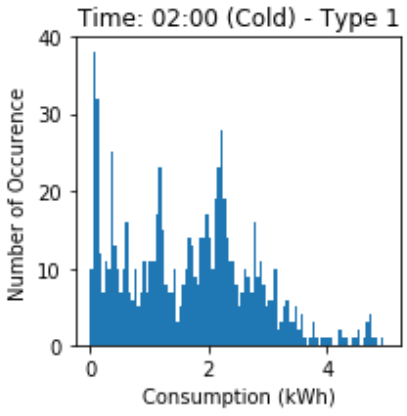

Time: 00:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

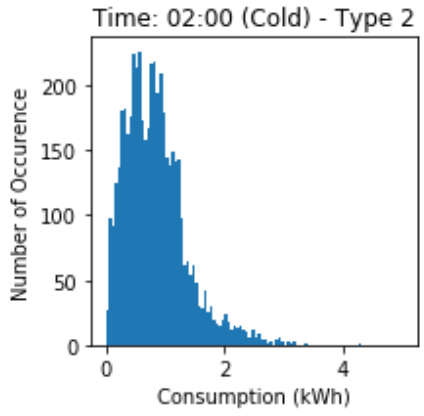

Time: 01:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

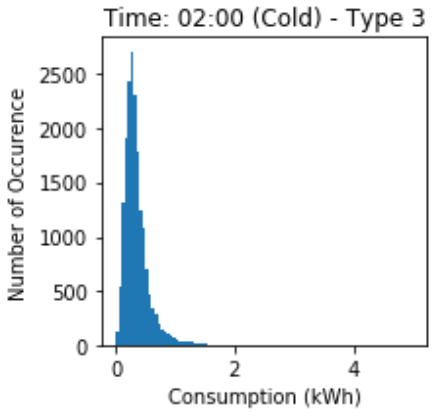

Time: 01:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

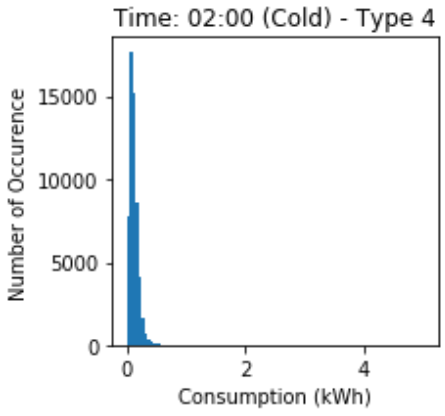

Time: 01:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

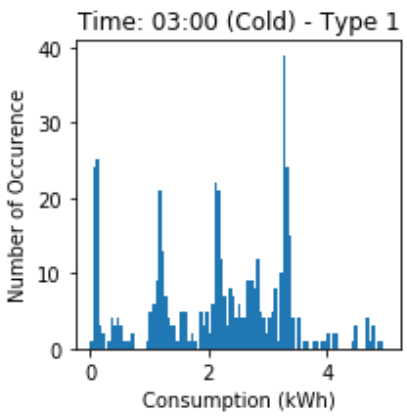

Time: 01:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

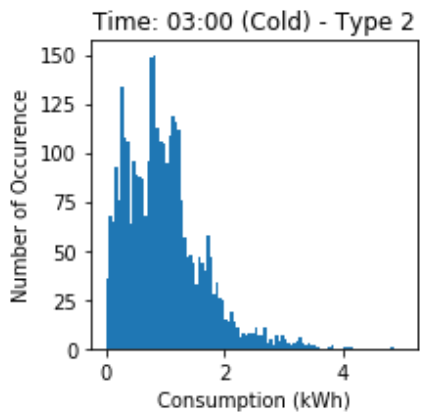

Time: 02:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

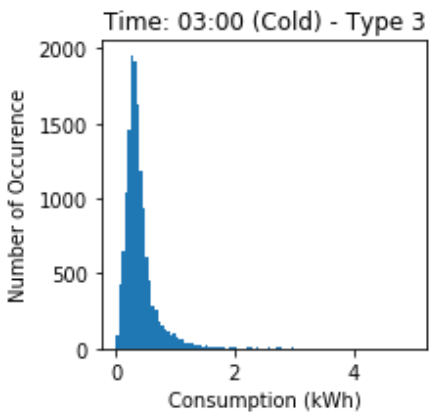

Time: 02:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

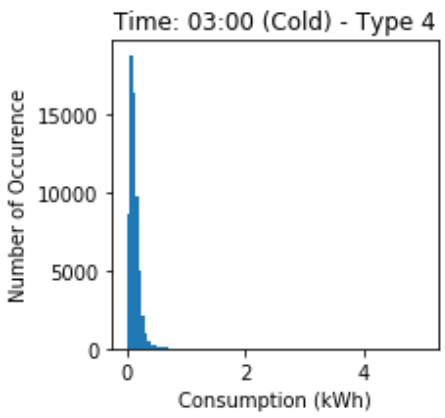

Time: 02:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

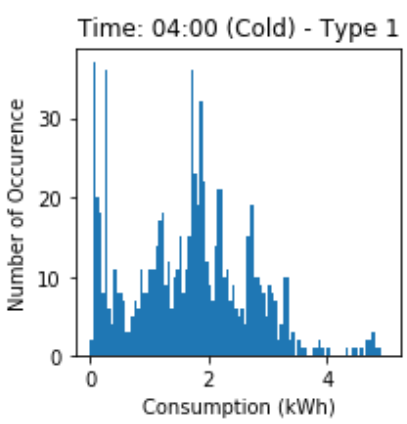

Time: 02:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

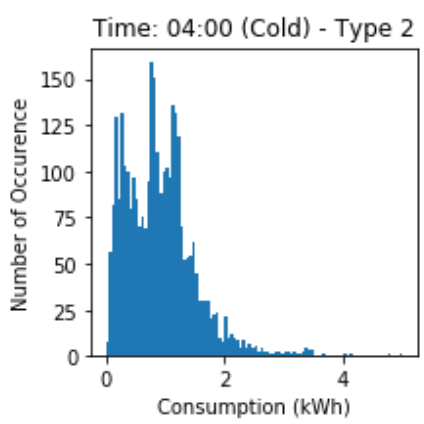

Time: 03:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

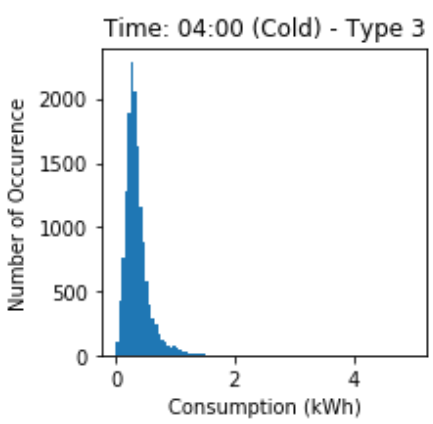

Time: 03:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

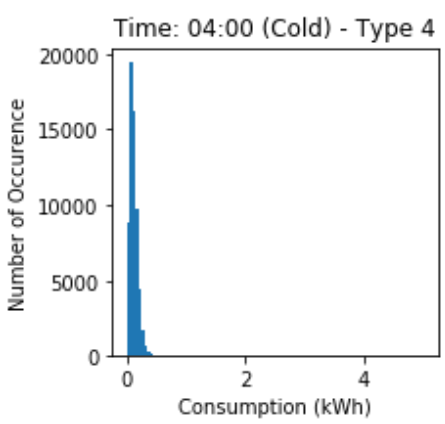

Time: 03:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

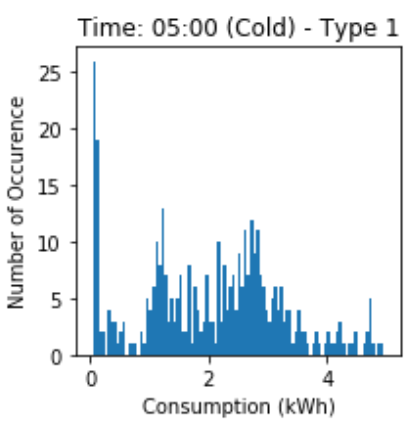

Time: 03:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

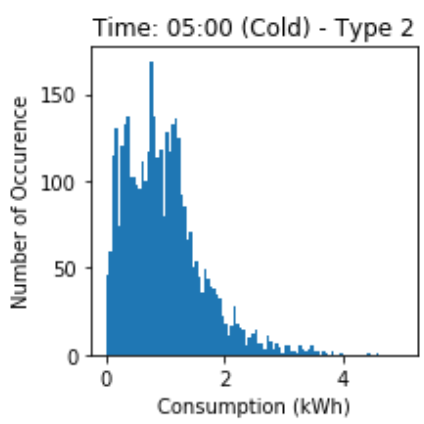

Time: 04:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

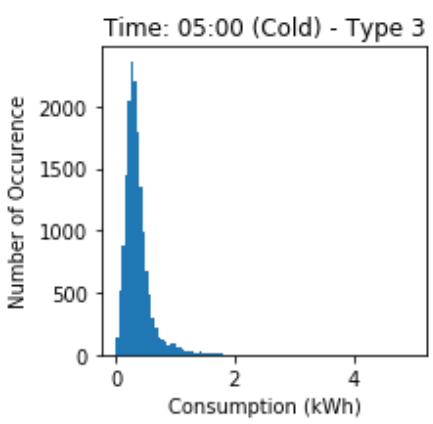

Time: 04:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

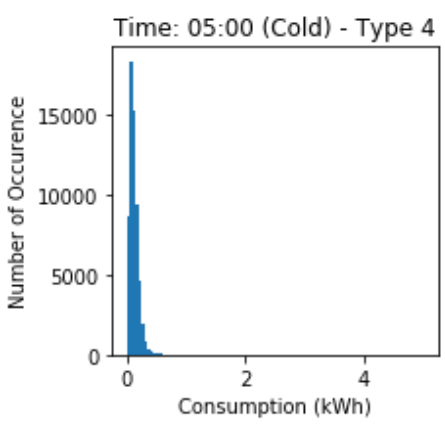

Time: 04:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

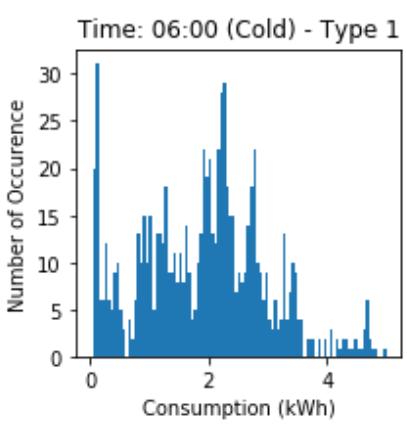

Time: 04:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

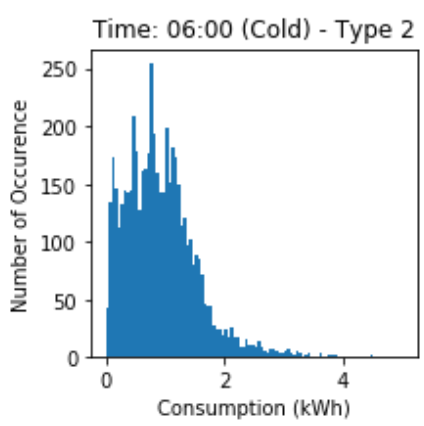

Time: 05:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

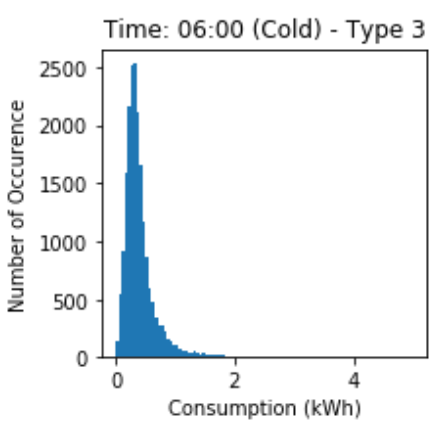

Time: 05:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

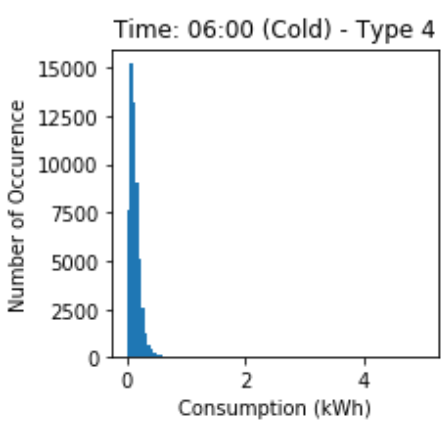

Time: 05:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)


Time: 05:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)


Time: 06:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)


Time: 06:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)


Time: 06:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

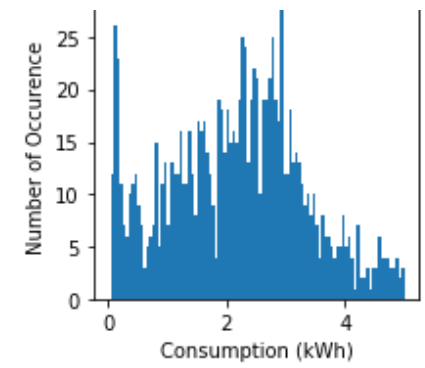

Time: 06:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

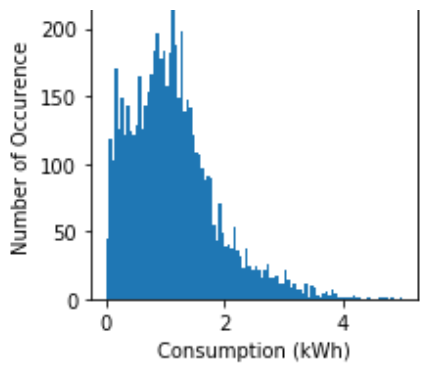
Time: 07:00 (Cold) - Type 1

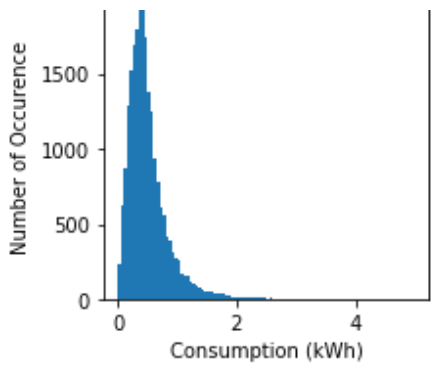
Time: 07:00 (Cold) - Type 2

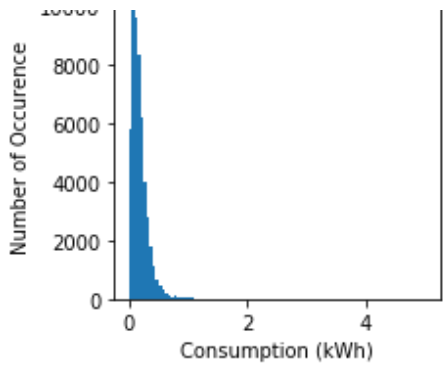
Time: 07:00 (Cold) - Type 3

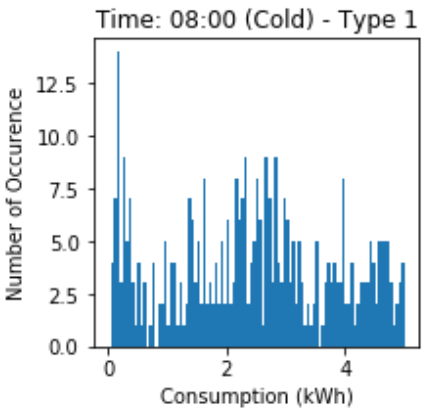
Time: 07:00 (Cold) - Type 4

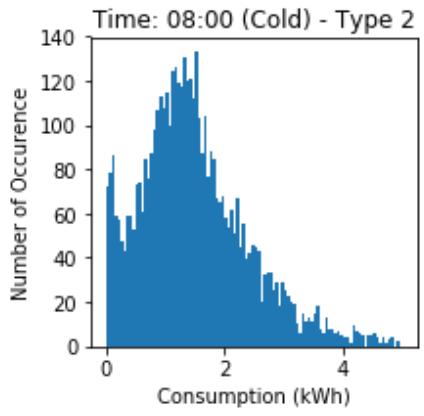

Number of Occurence
Consumption (kWh)

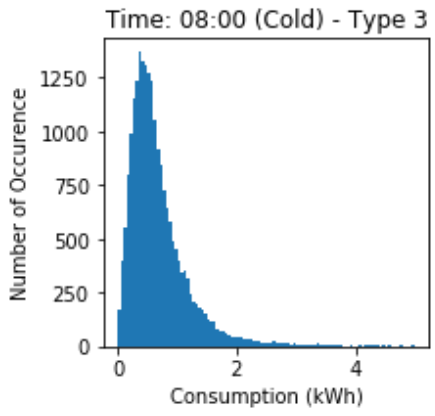

Number of Occurence
Consumption (kWh)

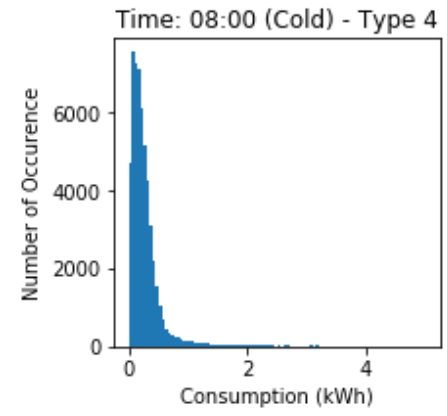

Number of Occurence
Consumption (kWh)

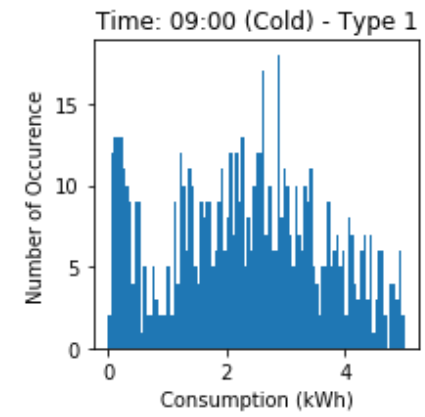

Number of Occurence
Consumption (kWh)

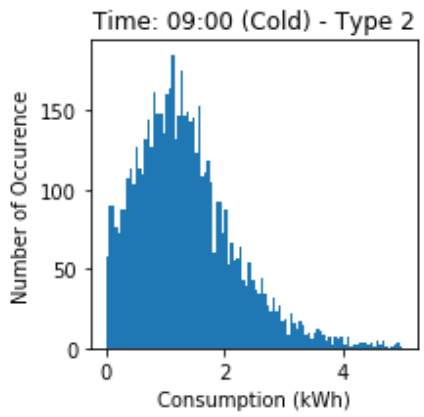

Time: 08:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

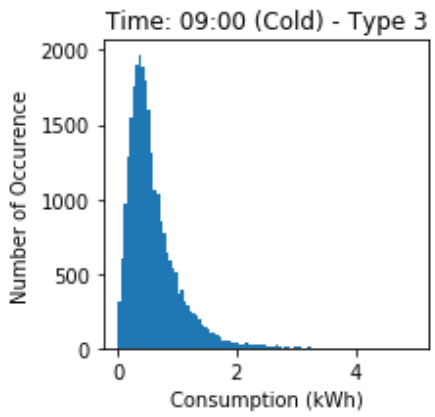

Time: 08:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

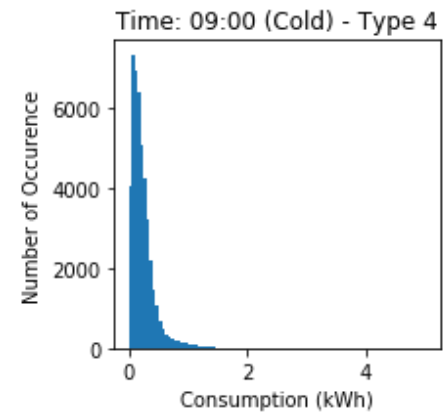

Time: 08:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

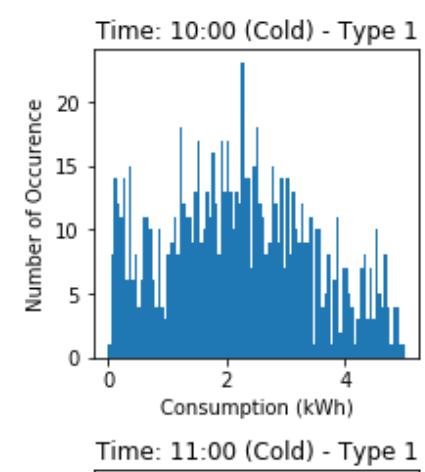

Time: 08:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

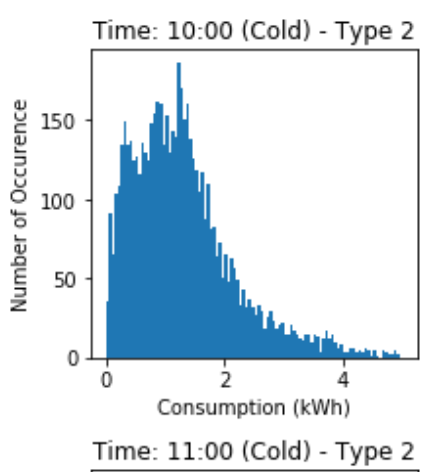

Time: 09:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

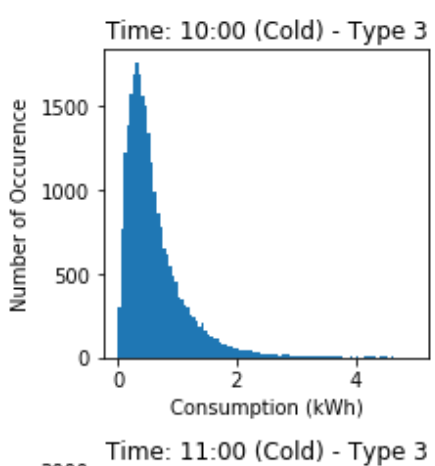

Time: 09:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

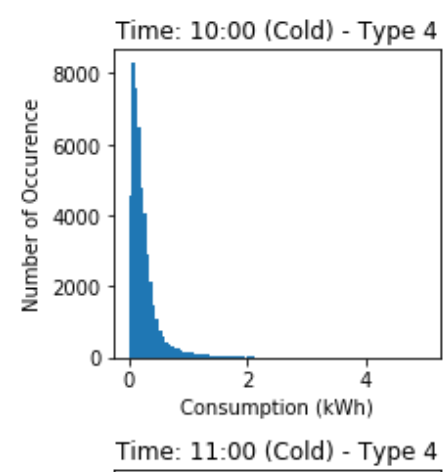

Time: 09:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

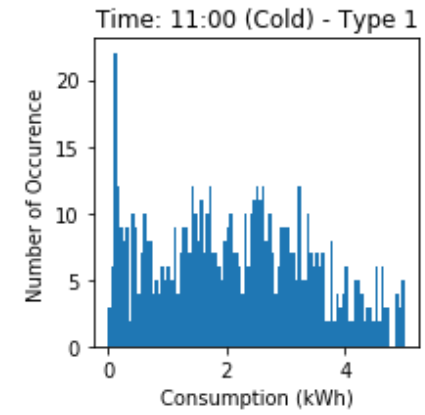

Time: 09:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

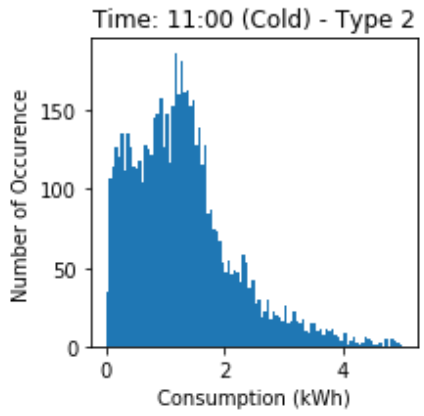

Time: 10:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

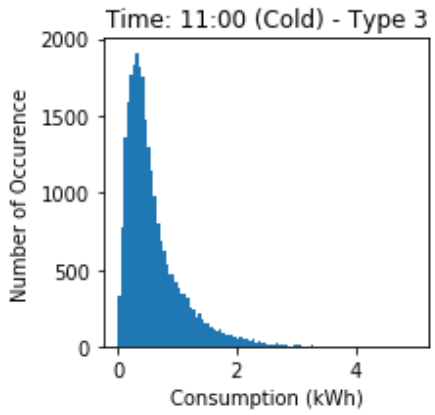

Time: 10:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

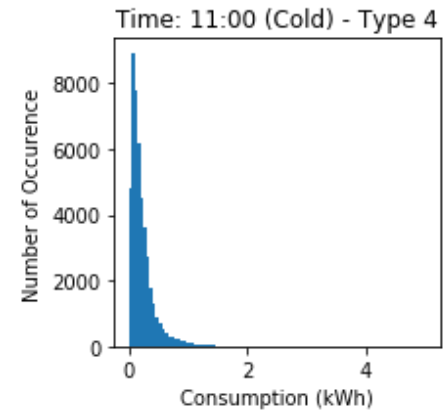

Time: 10:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

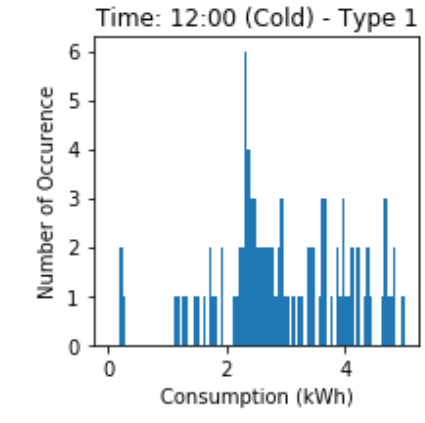

Time: 10:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

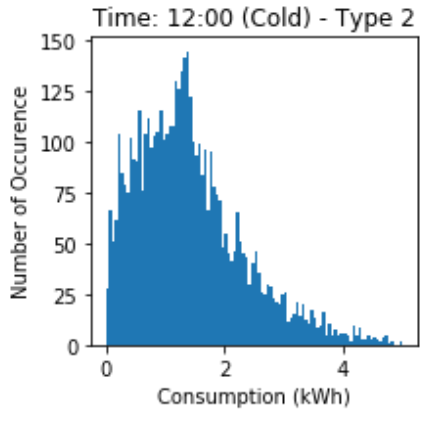

Time: 11:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

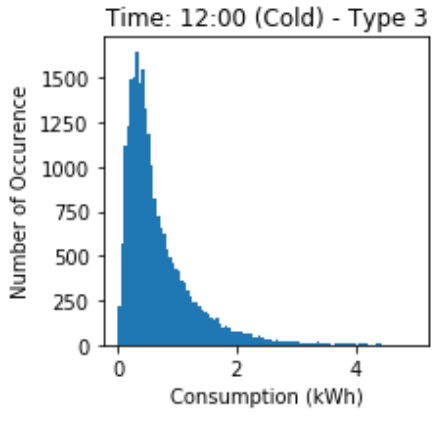

Time: 11:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

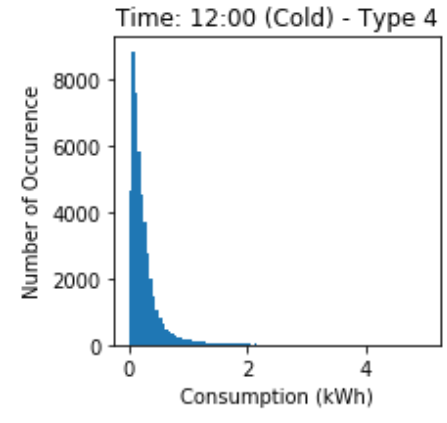

Time: 11:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

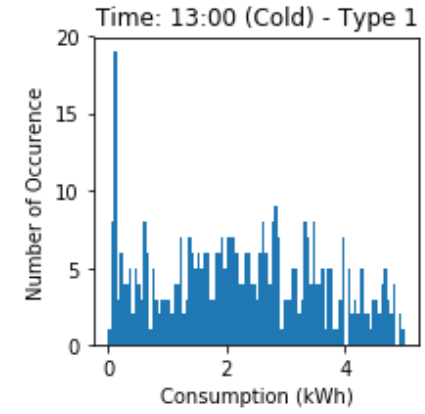

Time: 11:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

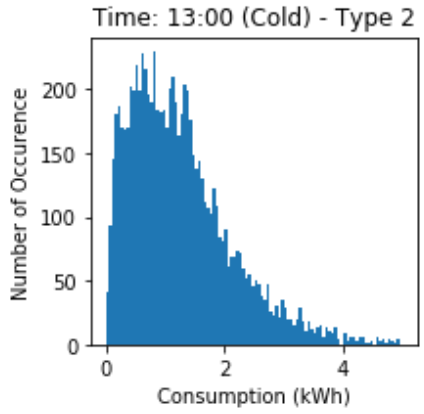

Time: 12:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

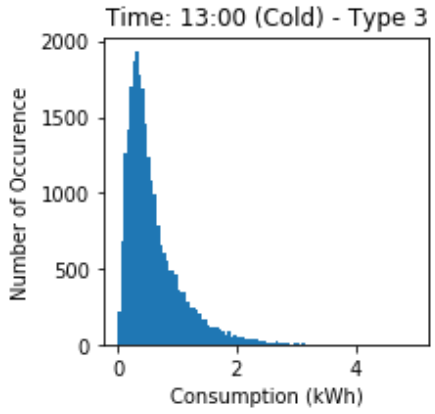

Time: 12:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

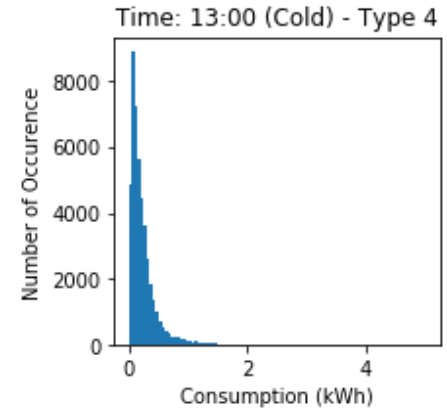

Time: 12:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

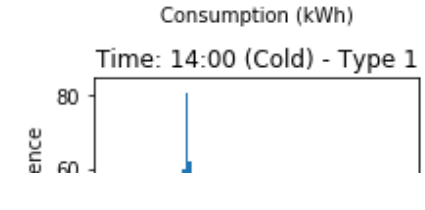

Time: 12:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

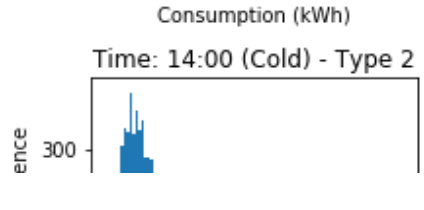

Time: 13:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

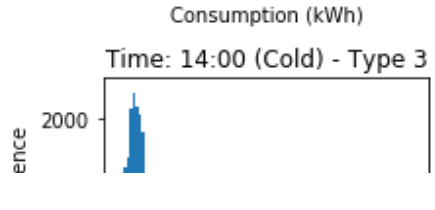

Time: 13:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)


Time: 13:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

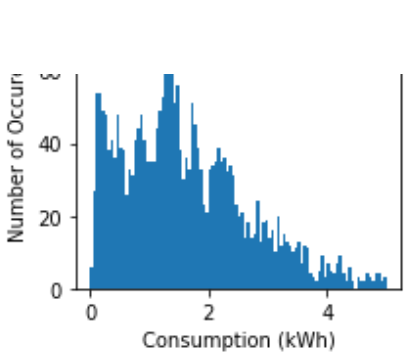

Time: 13:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

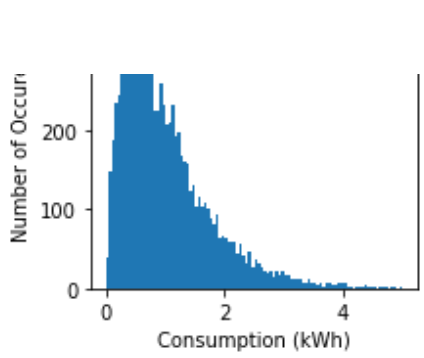

Time: 14:00 (Cold) - Type 1

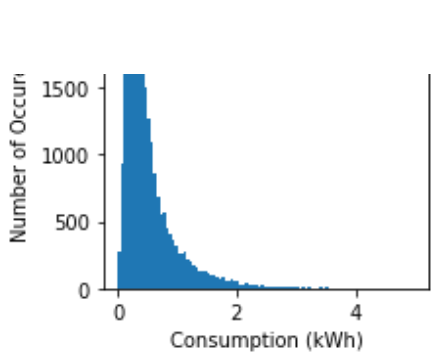

Time: 14:00 (Cold) - Type 2

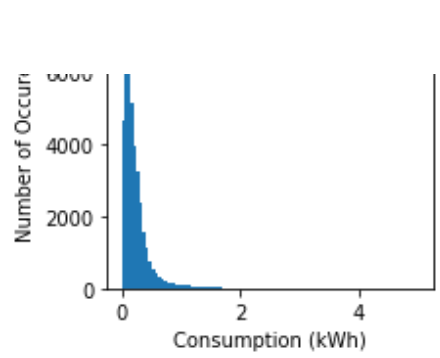

Time: 14:00 (Cold) - Type 3

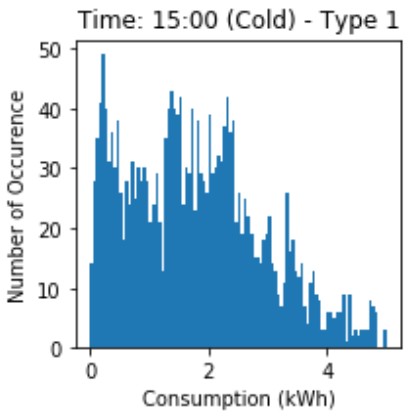

Time: 14:00 (Cold) - Type 4

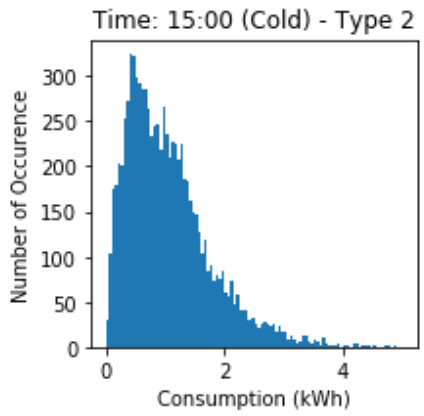

Number of Occurence
Consumption (kWh)

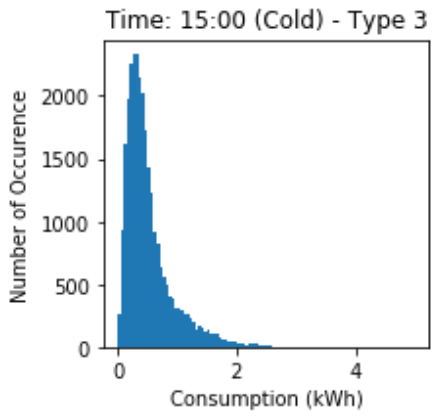

Number of Occurence
Consumption (kWh)

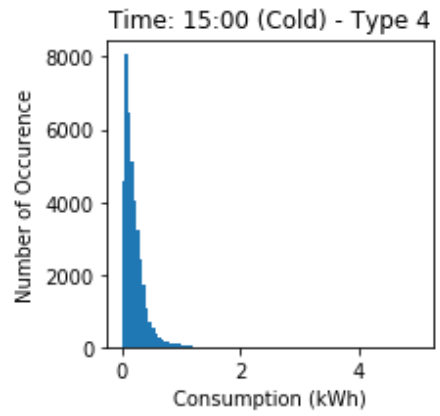

Number of Occurence
Consumption (kWh)

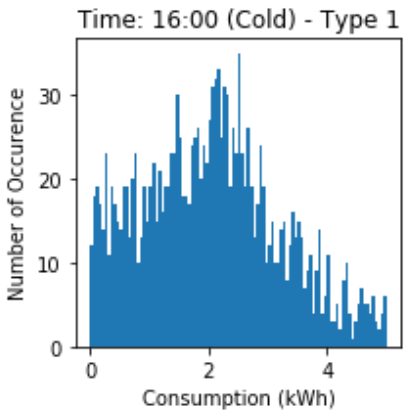

Number of Occurence
Consumption (kWh)

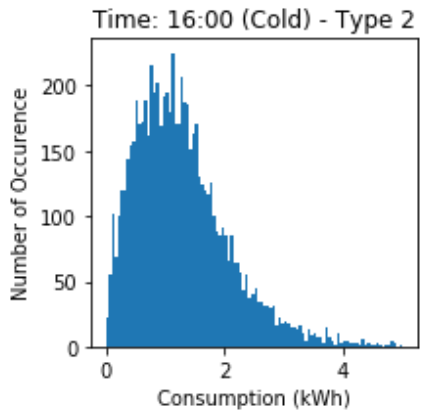

Time: 15:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

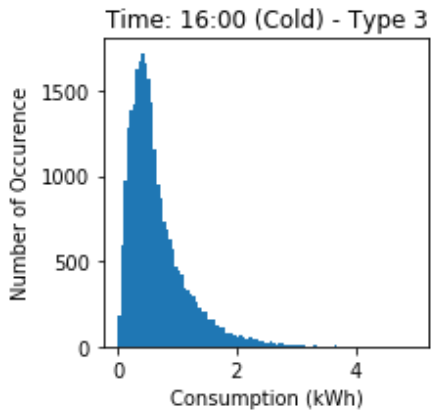

Time: 15:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

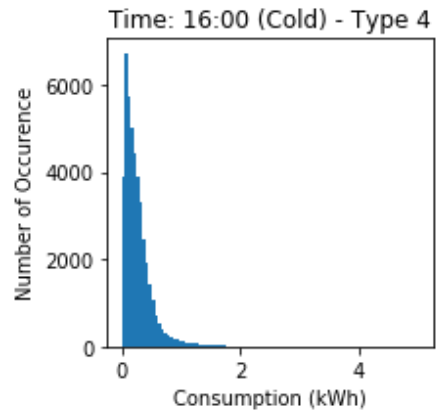

Time: 15:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

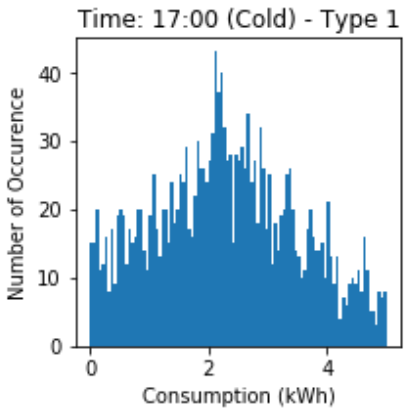

Time: 15:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

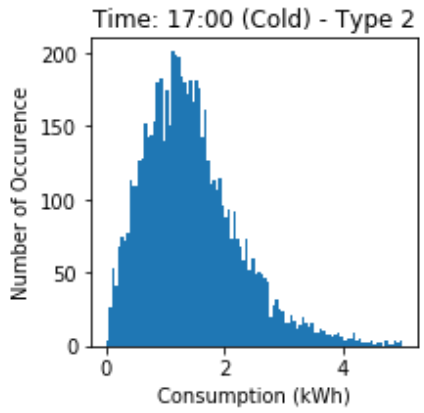

Time: 16:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

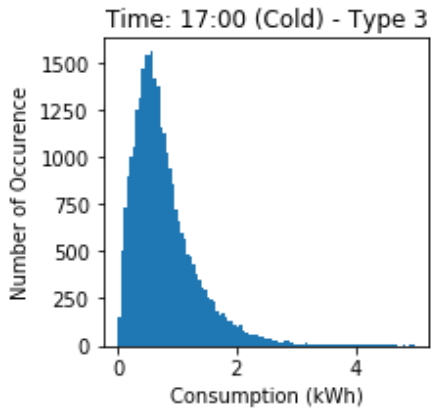

Time: 16:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

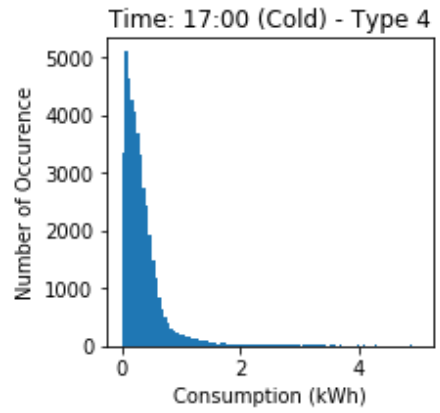

Time: 16:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

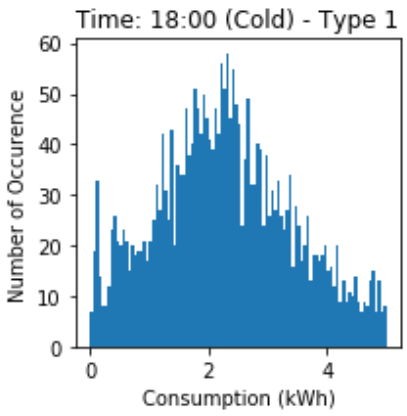

Time: 16:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

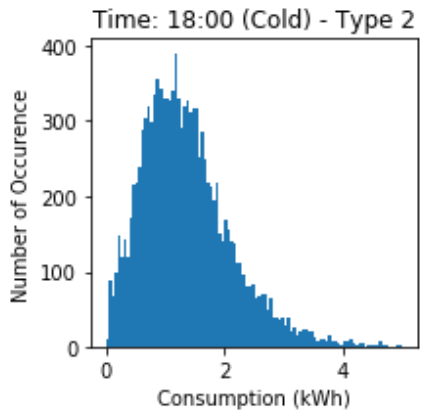

Time: 17:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

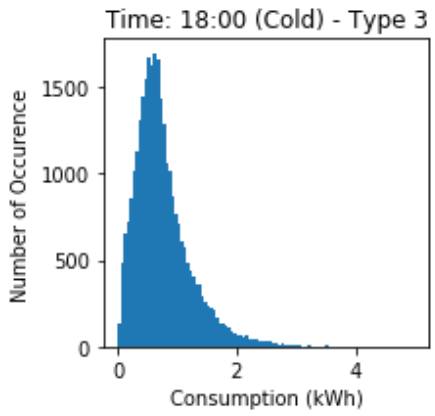

Time: 17:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

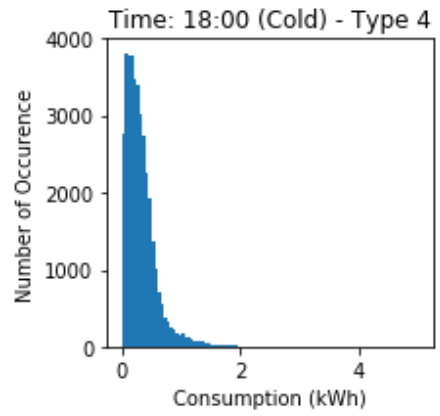

Time: 17:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

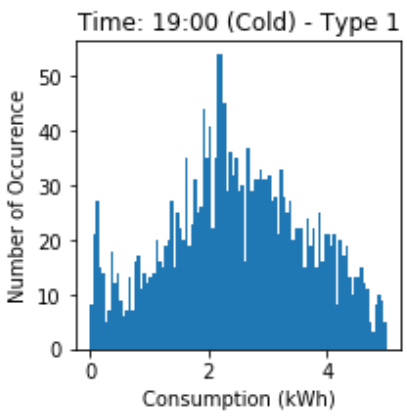

Time: 17:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

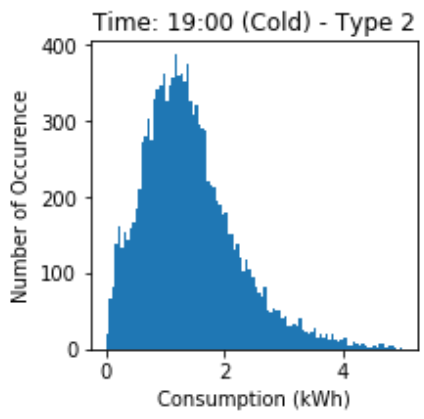

Time: 18:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

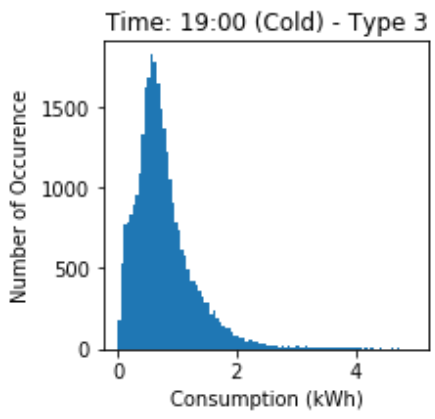

Time: 18:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

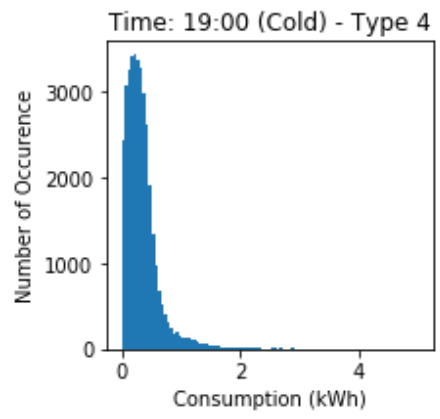

Time: 18:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

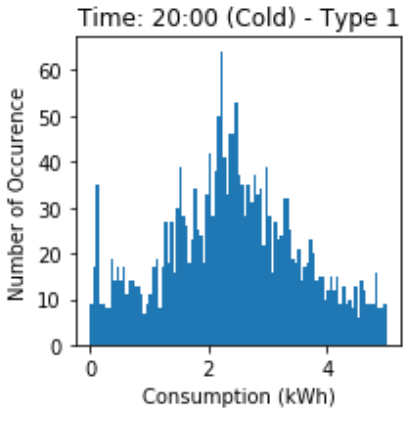

Time: 18:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

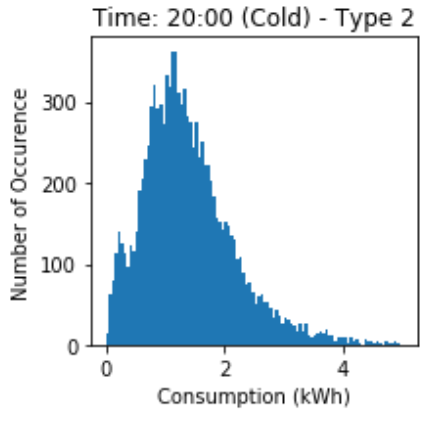

Number of Occurence
Consumption (kWh)

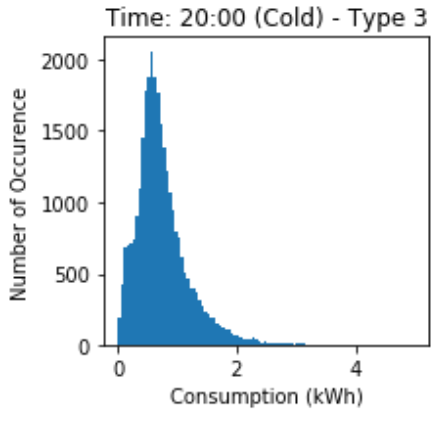

Number of Occurence
Consumption (kWh)

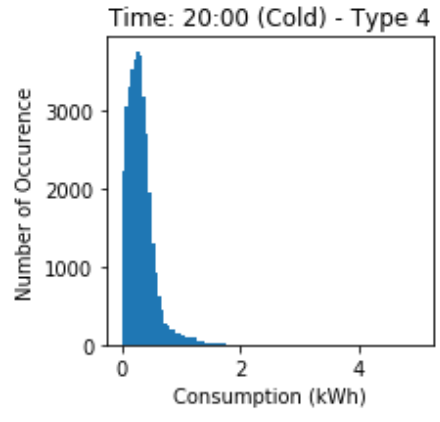

Number of Occurence
Consumption (kWh)

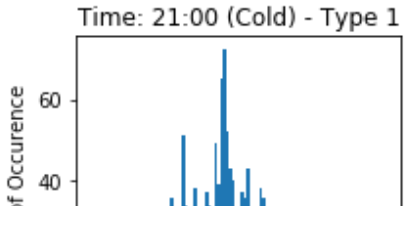

Number of Occurence
Consumption (kWh)

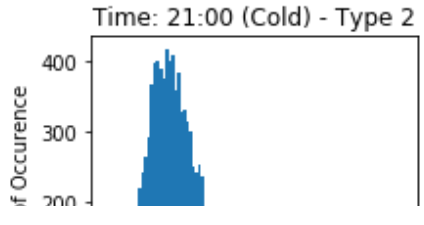

Time: 20:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

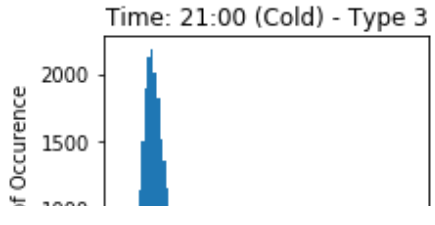

Time: 20:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

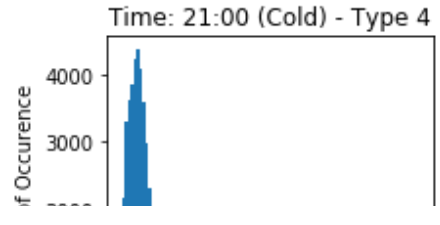

Time: 20:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

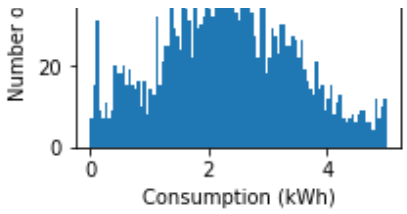

Time: 20:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

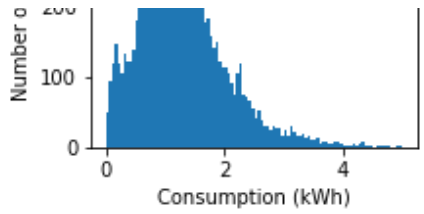

Time: 21:00 (Cold) - Type 1

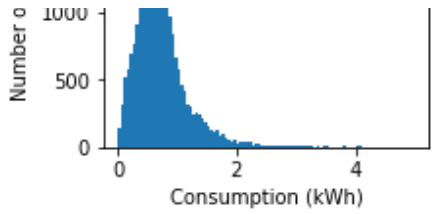

Time: 21:00 (Cold) - Type 2

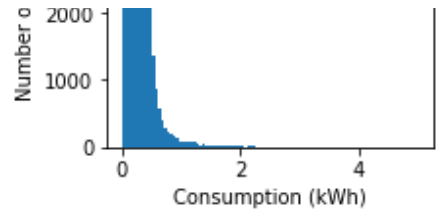

Time: 21:00 (Cold) - Type 3

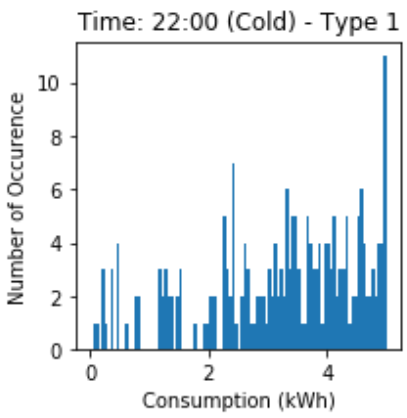

Time: 21:00 (Cold) - Type 4

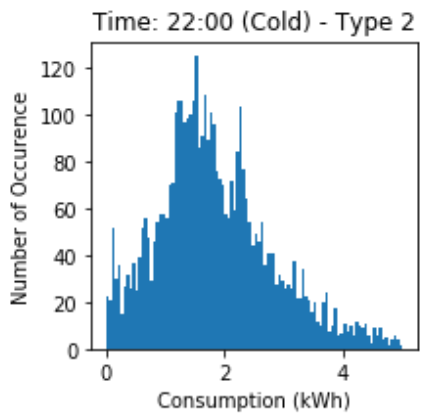

Consumption (kWh)

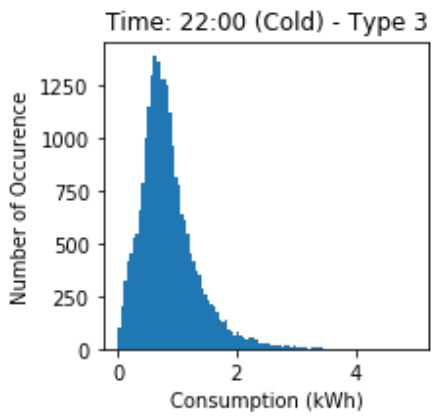

Consumption (kWh)

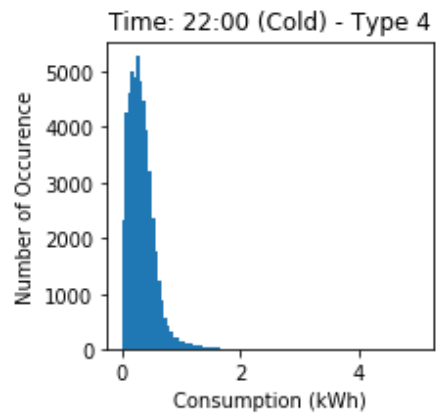

Consumption (kWh)

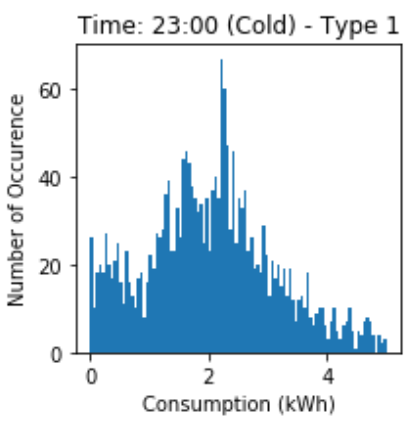

Consumption (kWh)

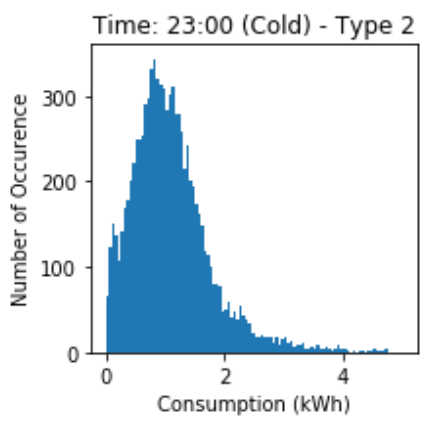

Time: 22:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

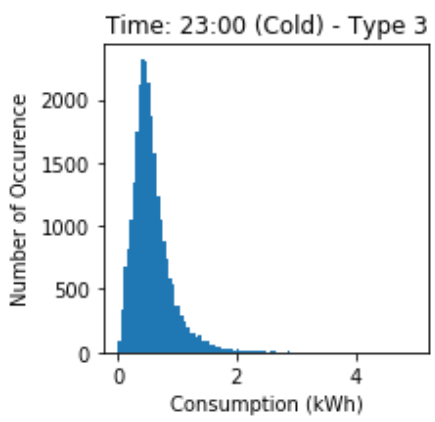

Time: 22:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

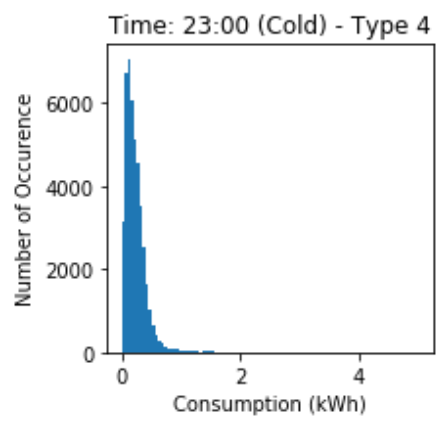

Time: 22:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)


Time: 22:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

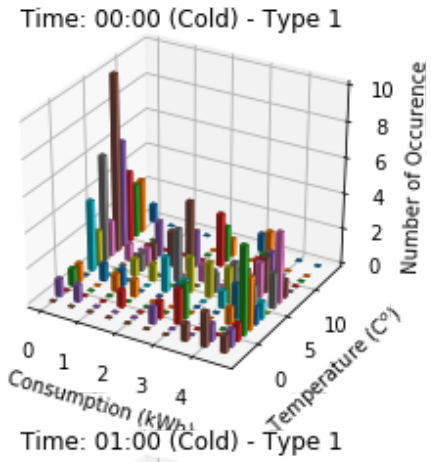

Time: 23:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)

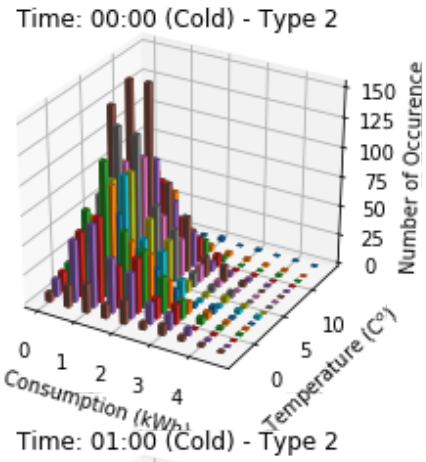

Time: 23:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)

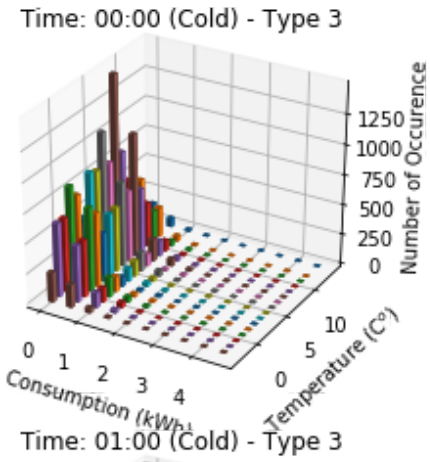

Time: 23:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)

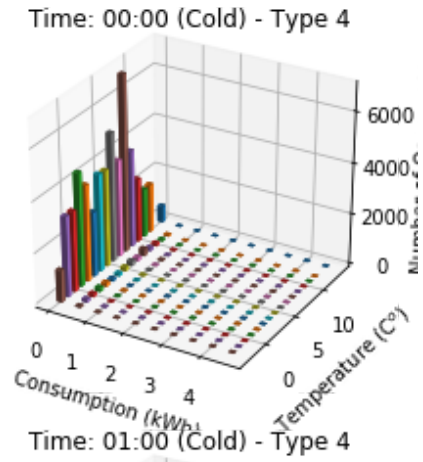

Time: 23:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)

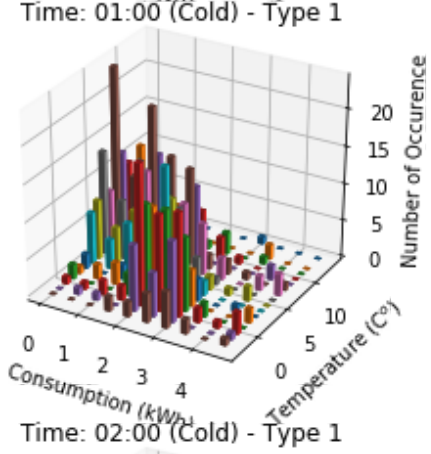
In [379]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 4 # cluster numbers
cons_type = ['Type 1', 'Type 2', 'Type 3', 'Type 4']
fig_all = plt.figure(figsize = (13,72))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            temp_list = list(set(t2['TempC']))
            temp_list.sort(reverse = True)
            for temp in temp_list:
                t3 = t2[t2['TempC'] == temp] # select the data with a specific temperature
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=10, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(10) * temp
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(10) * 0.15
                dy = np.ones(10) * 0.5
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 4 + k + 1
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel(r'Temperature (C$^o$)')
            ax.set_zlabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            temp_list = list(set(t2['TempC']))
            temp_list.sort(reverse = True)
            for temp in temp_list:
                t3 = t2[t2['TempC'] == temp] # select the data with a specific temperature
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=10, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(10) * temp
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(10) * 0.15
                dy = np.ones(10) * 0.5
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 4 + k + 1
            ax.set_title('Time: ' +str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel(r'Temperature (C$^o$)')
            ax.set_zlabel('Number of Occurence')
plt.tight_layout()
```

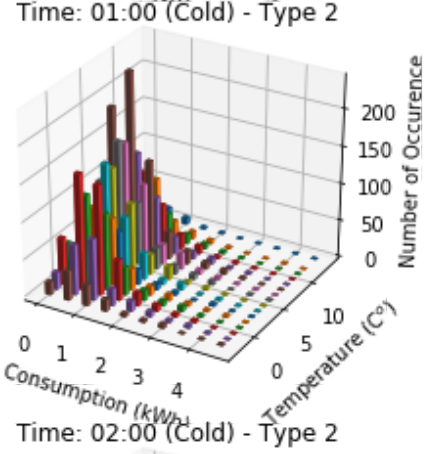

Time: 00:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

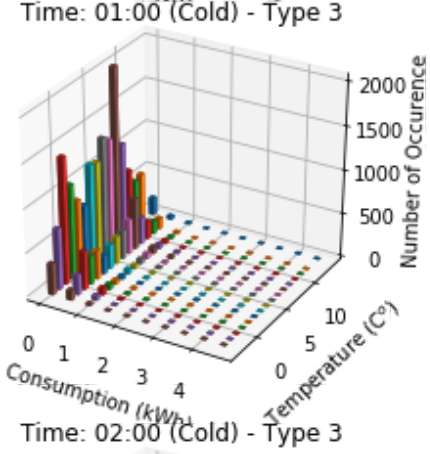

Time: 00:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

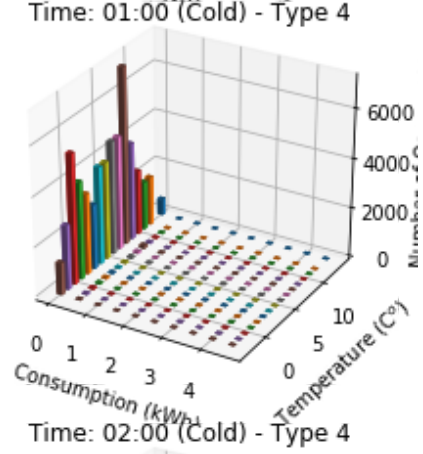

Time: 00:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

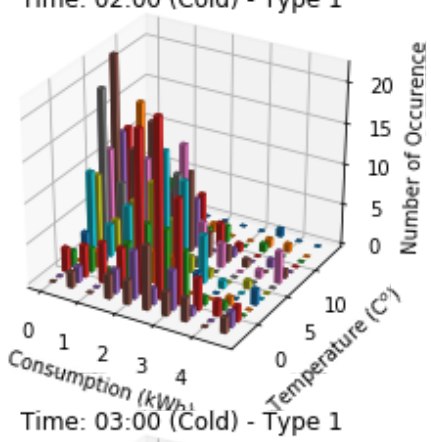

Time: 00:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

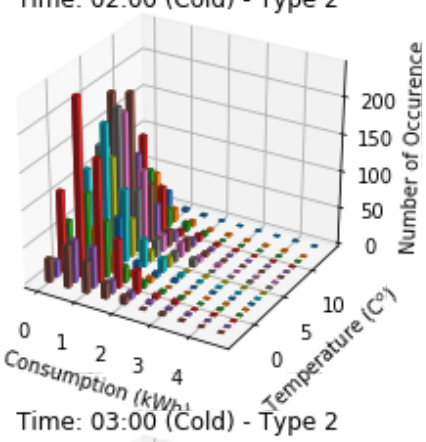

Time: 01:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

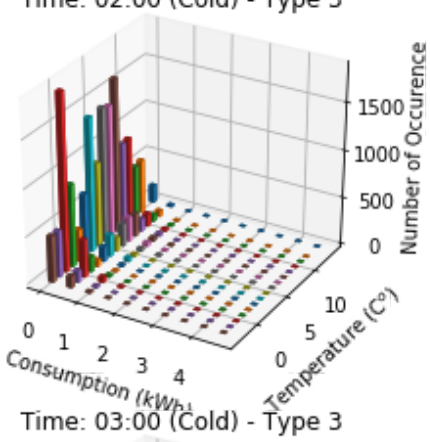

Time: 01:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

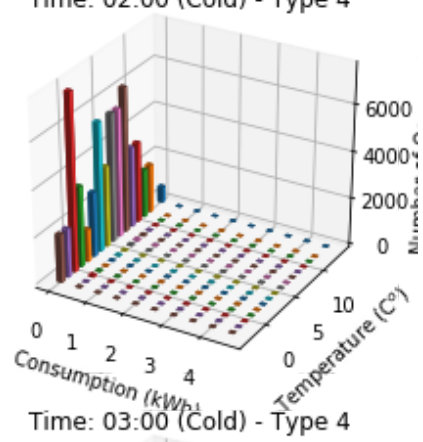

Time: 01:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

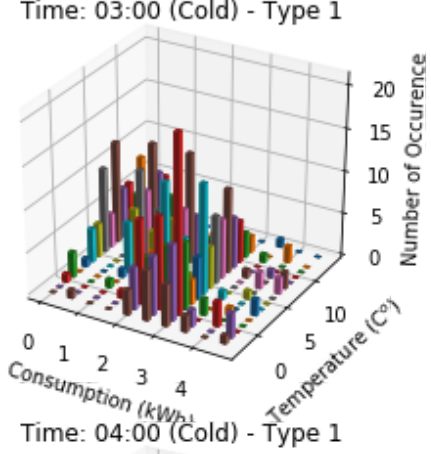

Time: 01:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

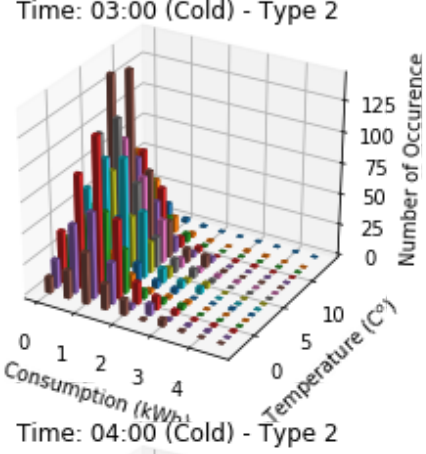

Time: 02:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

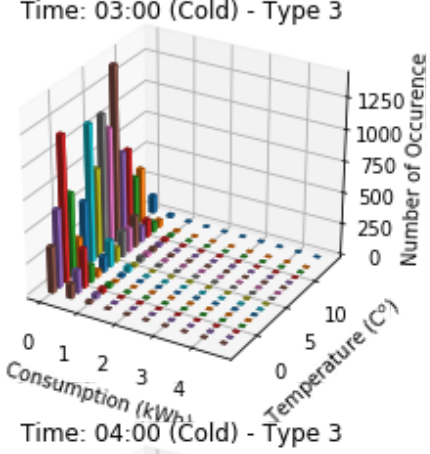

Time: 02:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

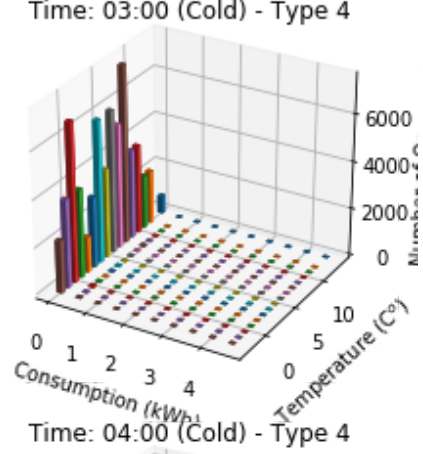

Time: 02:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

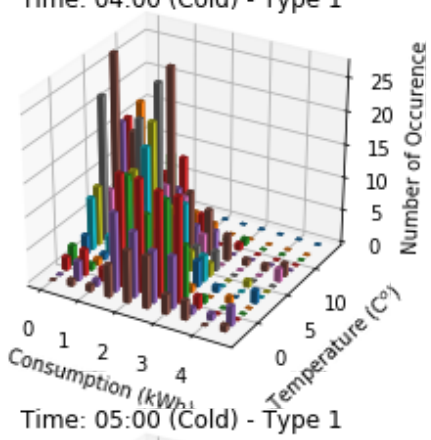

Time: 02:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

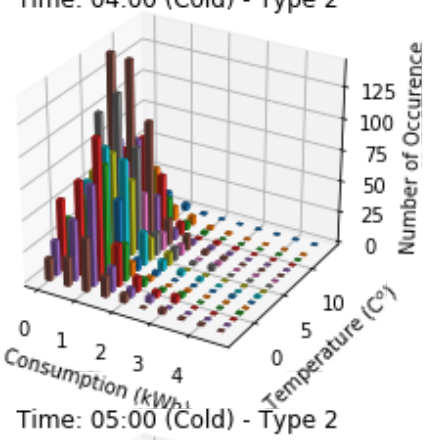

Time: 03:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

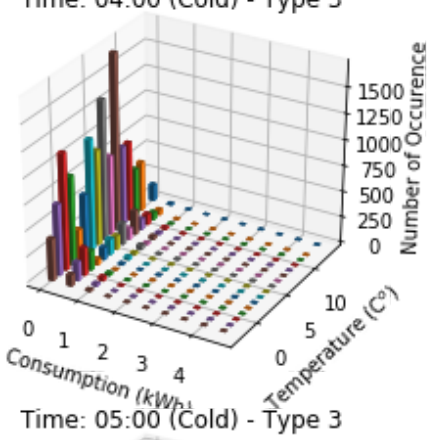

Time: 03:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

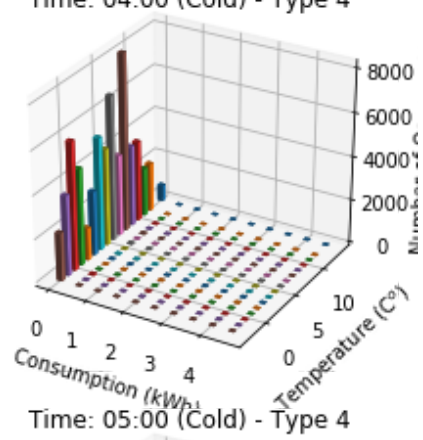

Time: 03:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

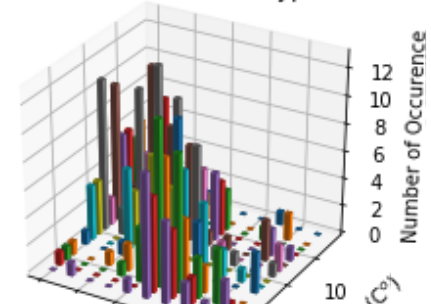

Time: 03:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

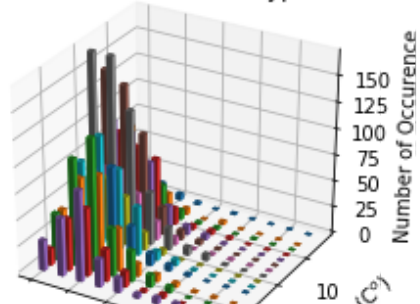

Time: 04:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

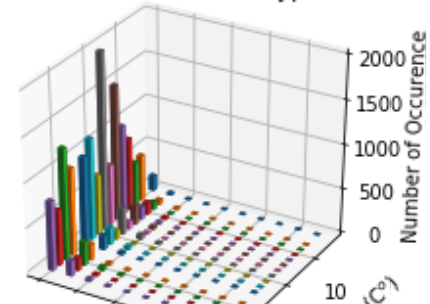

Time: 04:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

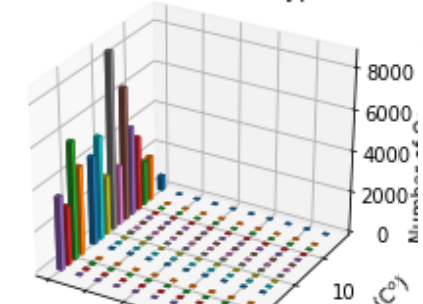

Time: 04:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

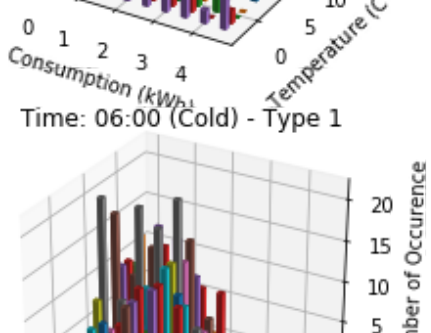

Time: 04:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

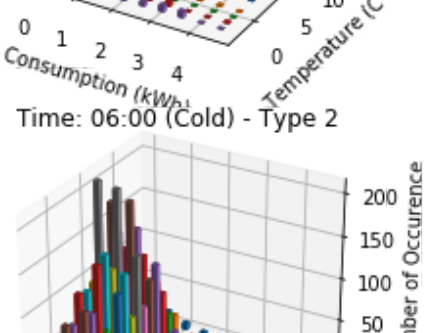

Time: 05:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

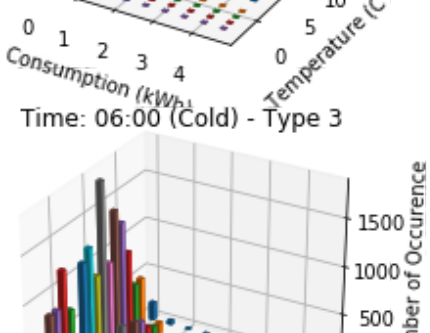

Time: 05:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

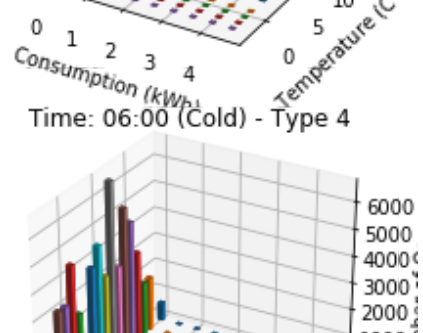

Time: 05:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

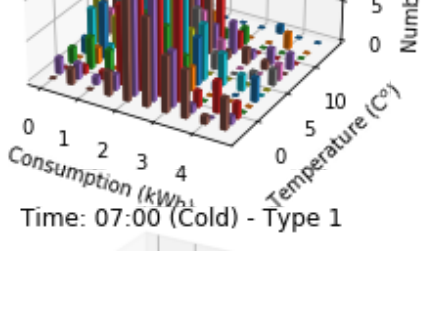

Time: 05:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)

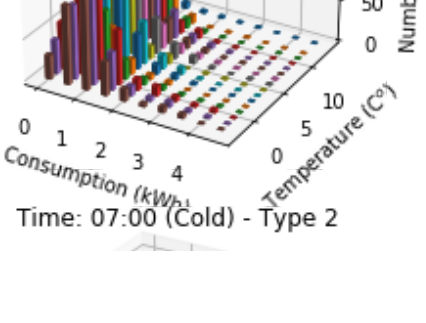

Time: 06:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

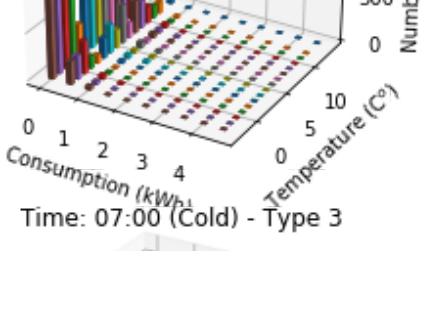

Time: 06:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)

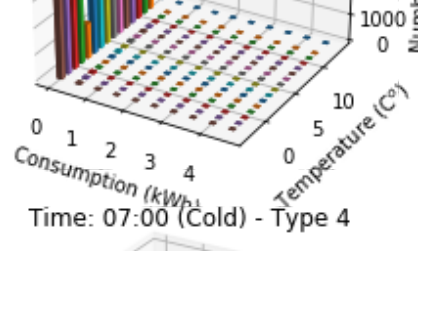

Time: 06:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

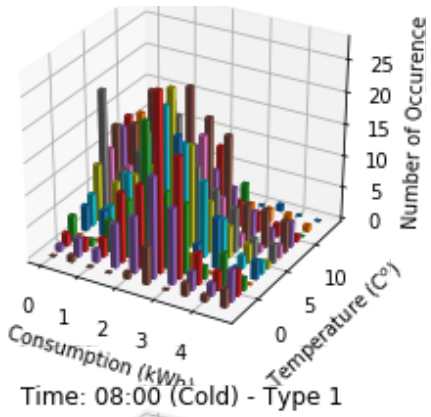

Time: 06:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)


Time: 07:00 (Cold) - Type 1
Number of Occurence
Consumption (kWh)
Temperature (C°)

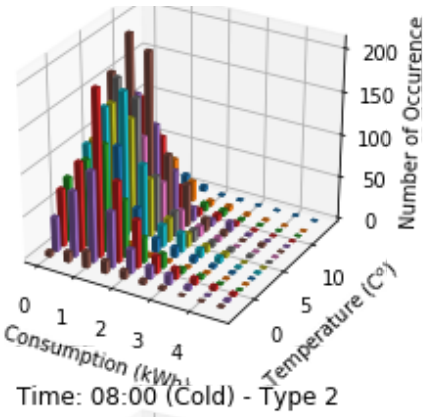

Time: 07:00 (Cold) - Type 2
Number of Occurence
Consumption (kWh)
Temperature (C°)


Time: 07:00 (Cold) - Type 3
Number of Occurence
Consumption (kWh)
Temperature (C°)

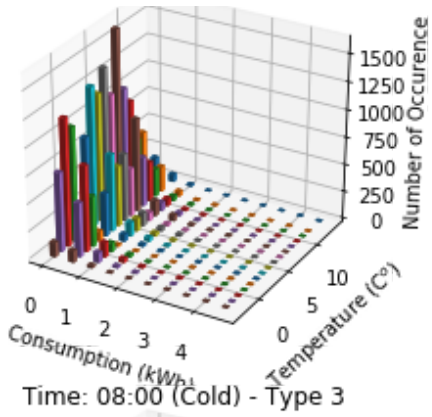

Time: 07:00 (Cold) - Type 4
Number of Occurence
Consumption (kWh)
Temperature (C°)



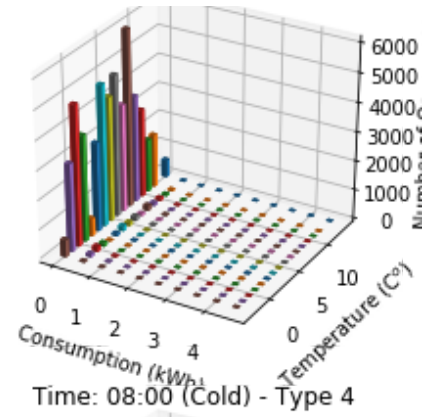
Time: 08:00 (Cold) - Type 1



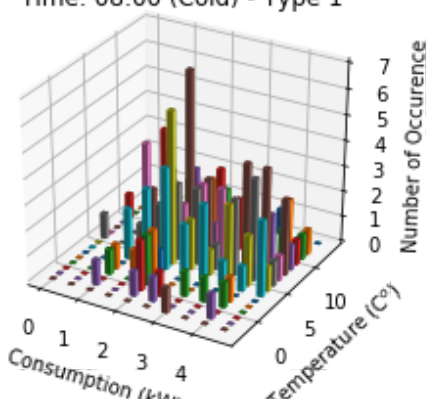
Time: 08:00 (Cold) - Type 2



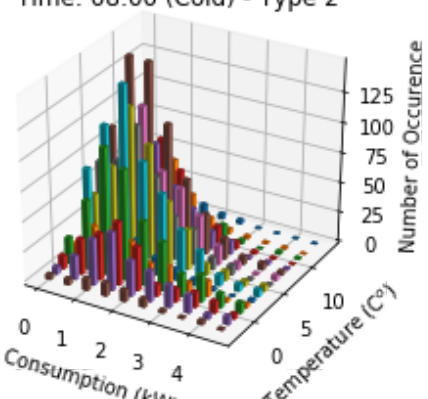
Time: 08:00 (Cold) - Type 3



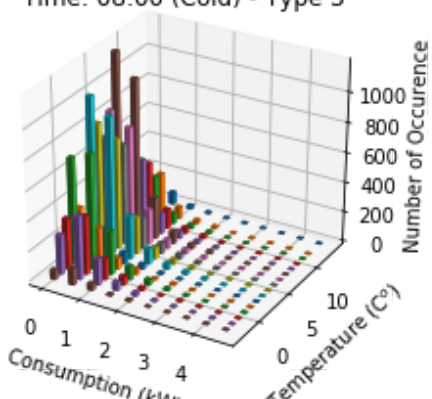
Time: 08:00 (Cold) - Type 4



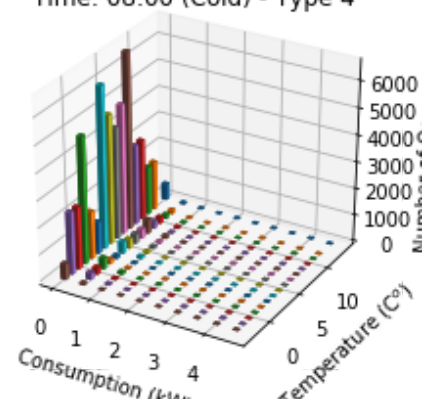
Time: 09:00 (Cold) - Type 1



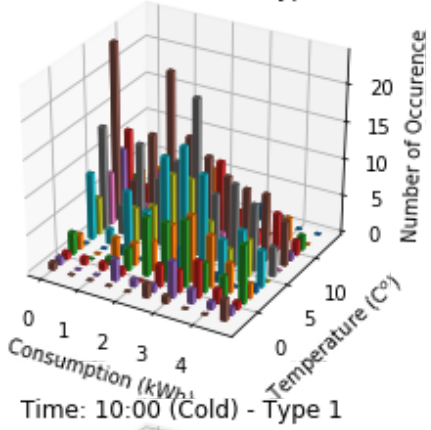
Time: 09:00 (Cold) - Type 2



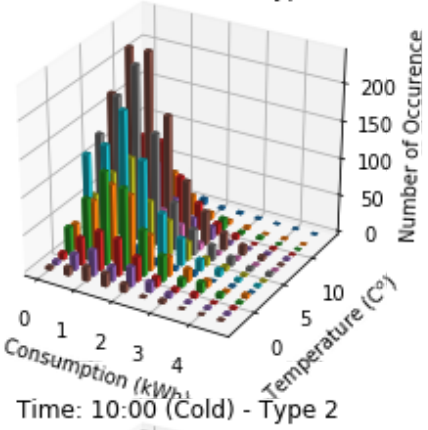
Time: 09:00 (Cold) - Type 3



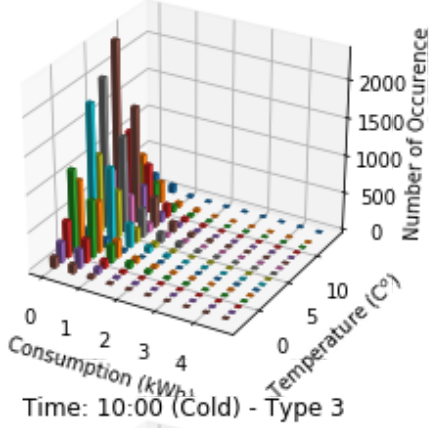
Time: 09:00 (Cold) - Type 4



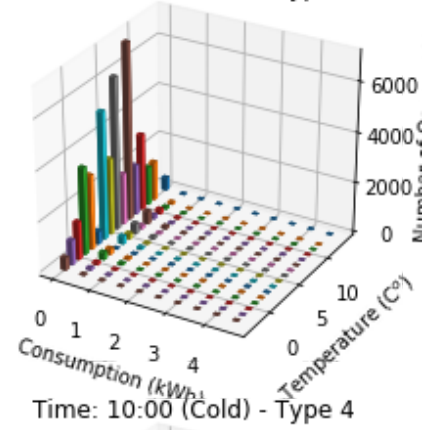
Time: 10:00 (Cold) - Type 1



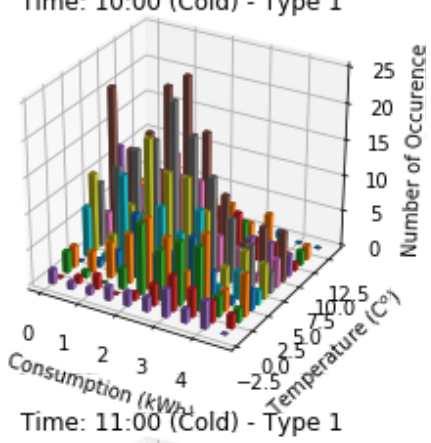
Time: 10:00 (Cold) - Type 2



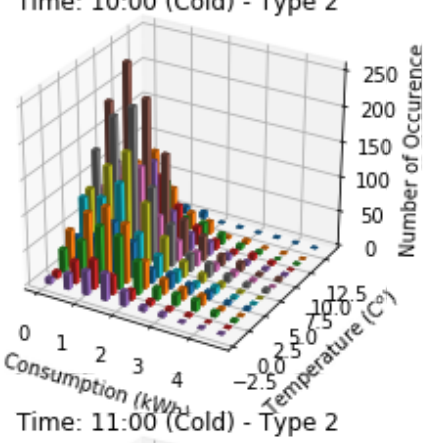
Time: 10:00 (Cold) - Type 3



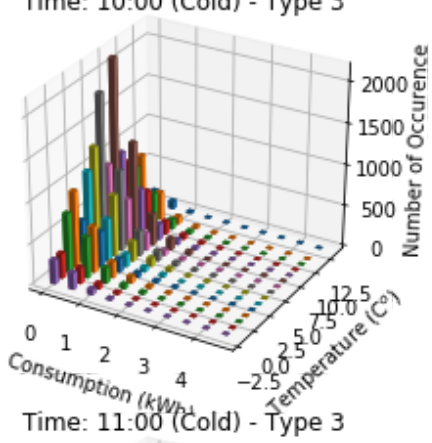
Time: 10:00 (Cold) - Type 4



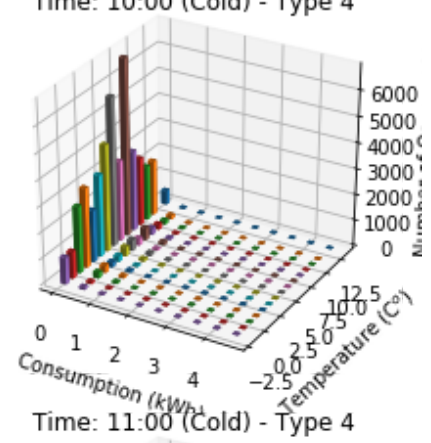
Time: 11:00 (Cold) - Type 1



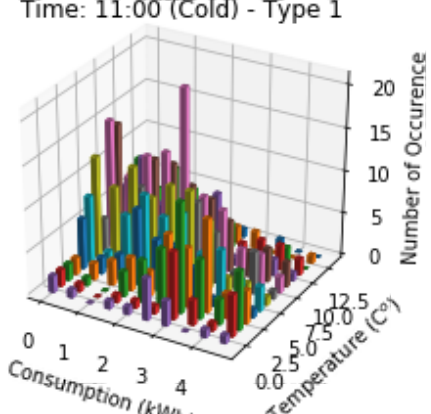
Time: 11:00 (Cold) - Type 2



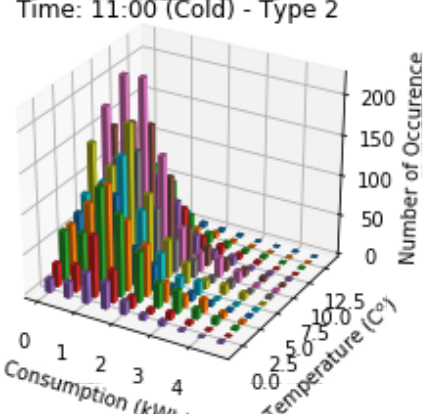
Time: 11:00 (Cold) - Type 3



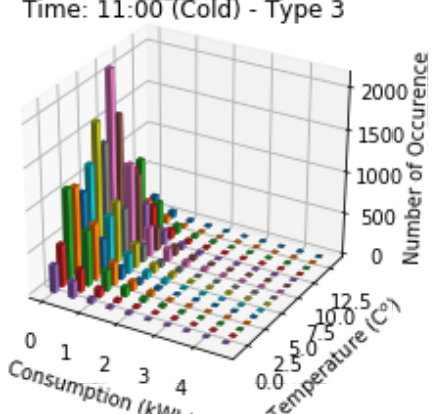
Time: 11:00 (Cold) - Type 4



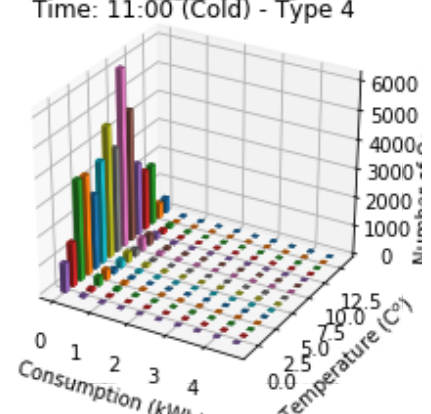
Time: 12:00 (Cold) - Type 1



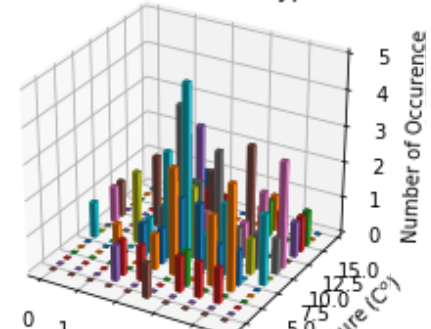
Time: 12:00 (Cold) - Type 2



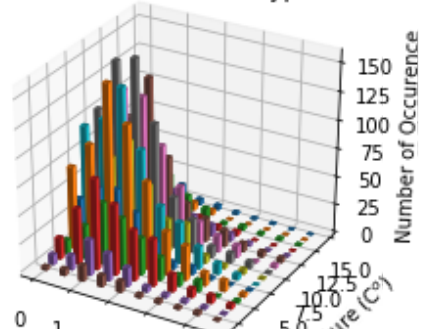
Time: 12:00 (Cold) - Type 3



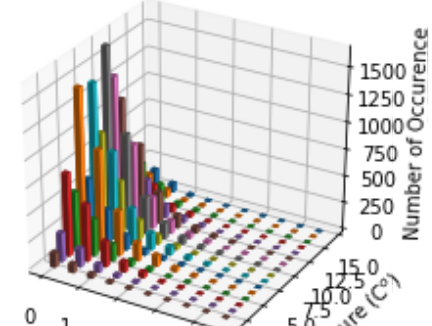
Time: 12:00 (Cold) - Type 4



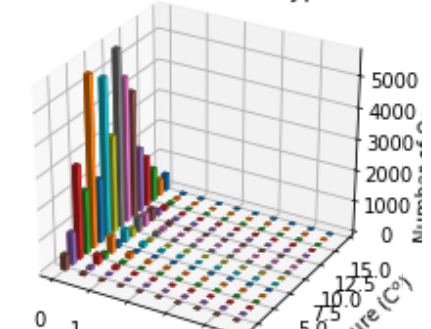
Time: 13:00 (Cold) - Type 1



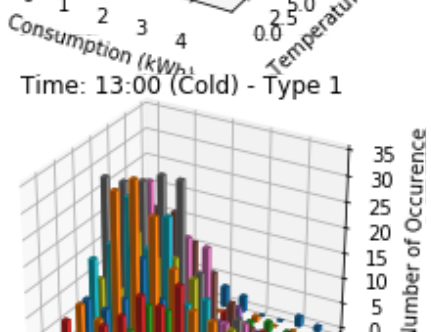
Time: 13:00 (Cold) - Type 2



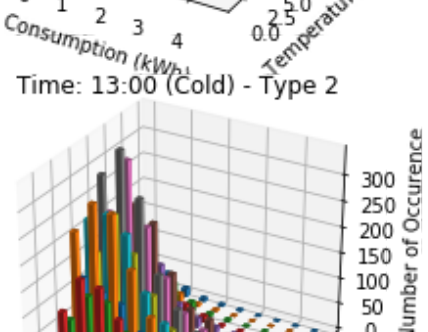
Time: 13:00 (Cold) - Type 3



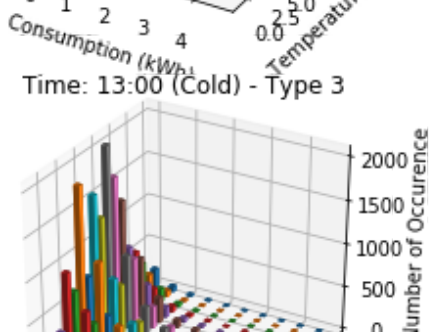
Time: 13:00 (Cold) - Type 4



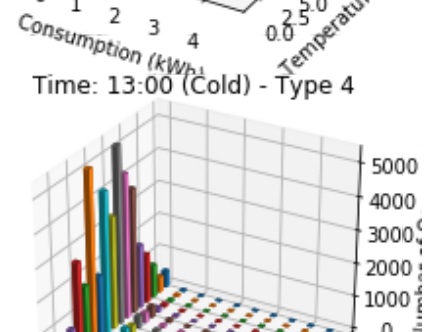
Time: 14:00 (Cold) - Type 1



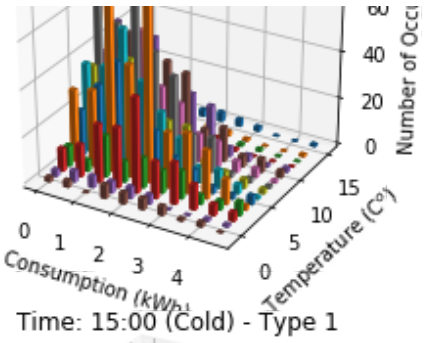
Time: 14:00 (Cold) - Type 2



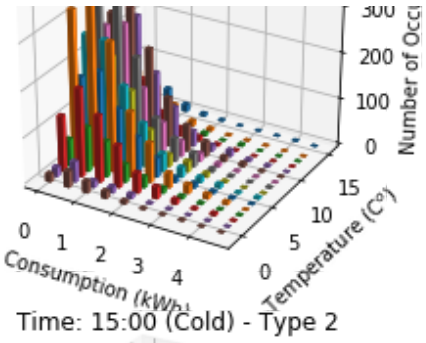
Time: 14:00 (Cold) - Type 3



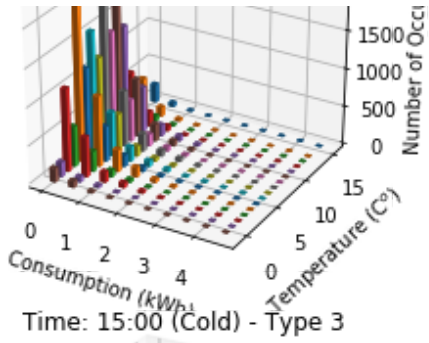
Time: 14:00 (Cold) - Type 4



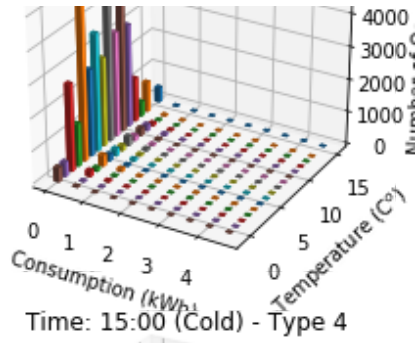
Time: 15:00 (Cold) - Type 1



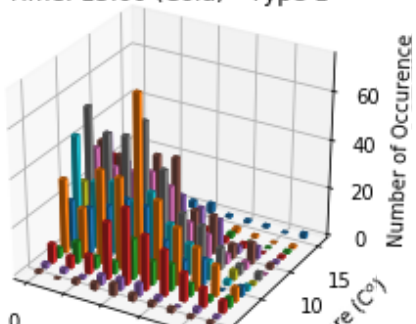
Time: 15:00 (Cold) - Type 2



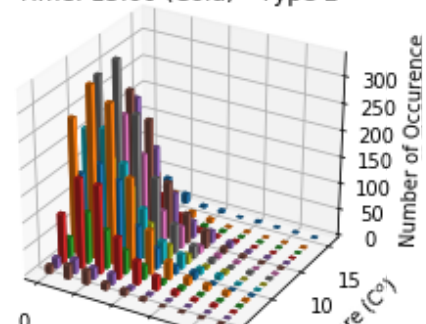
Time: 15:00 (Cold) - Type 3



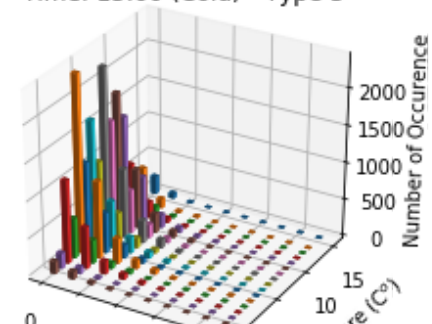
Time: 15:00 (Cold) - Type 4



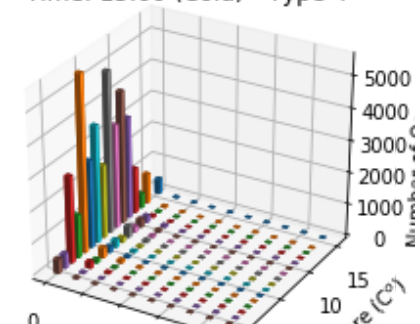
Time: 16:00 (Cold) - Type 1



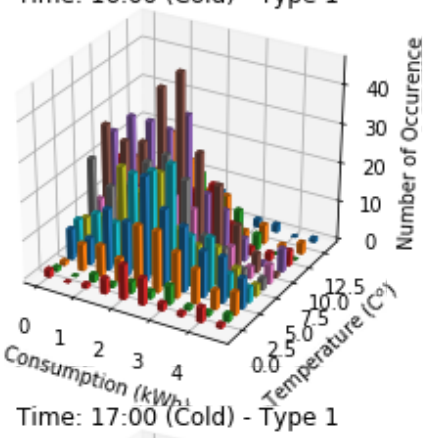
Time: 16:00 (Cold) - Type 2



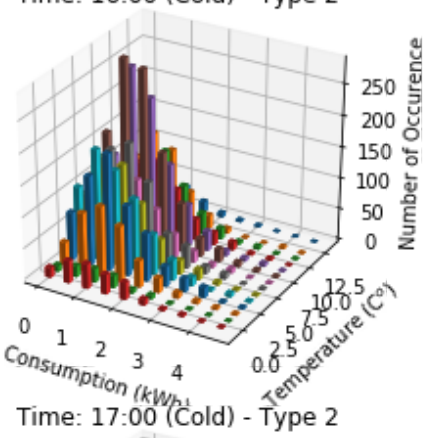
Time: 16:00 (Cold) - Type 3



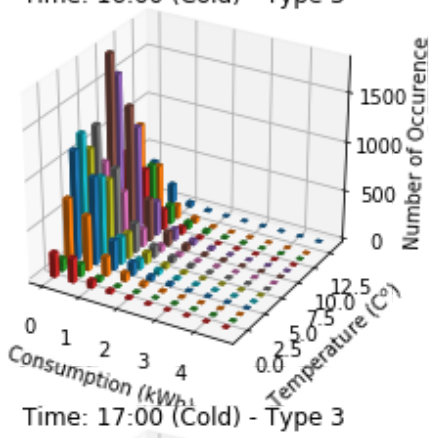
Time: 16:00 (Cold) - Type 4



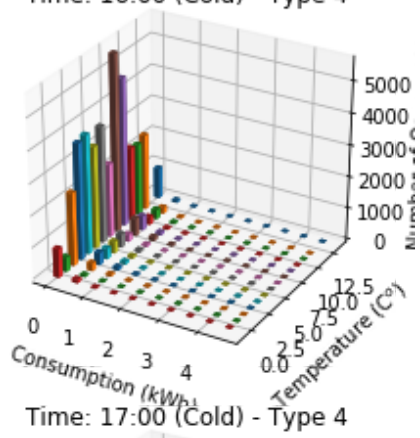
Time: 17:00 (Cold) - Type 1



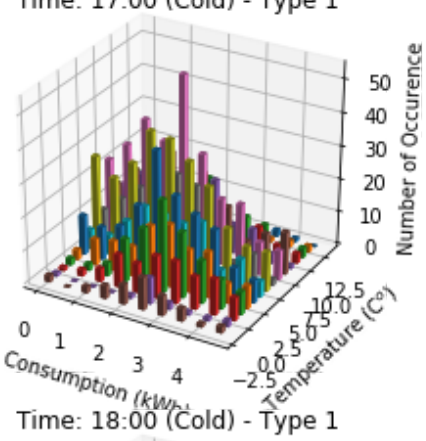
Time: 17:00 (Cold) - Type 2



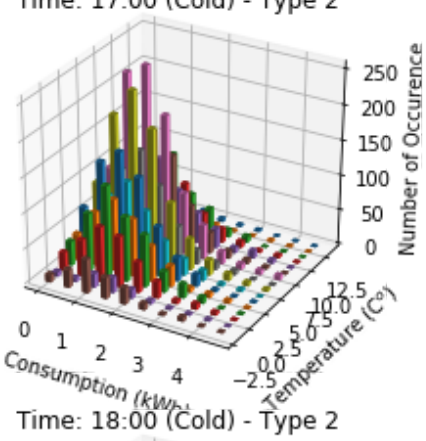
Time: 17:00 (Cold) - Type 3



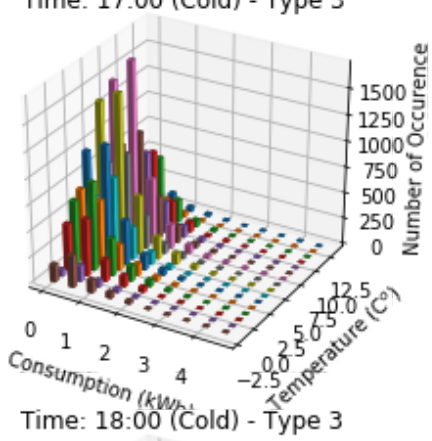
Time: 17:00 (Cold) - Type 4



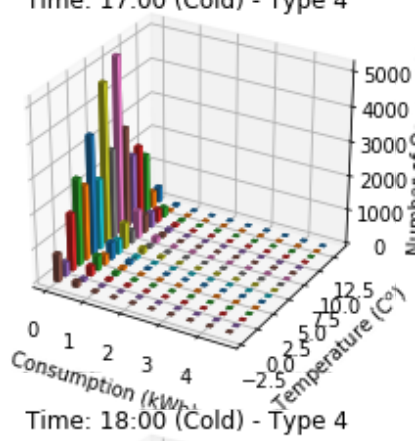
Time: 18:00 (Cold) - Type 1



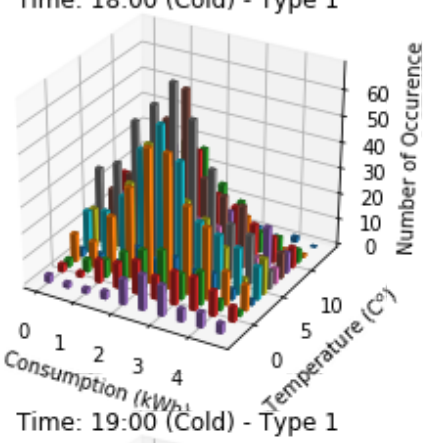
Time: 18:00 (Cold) - Type 2



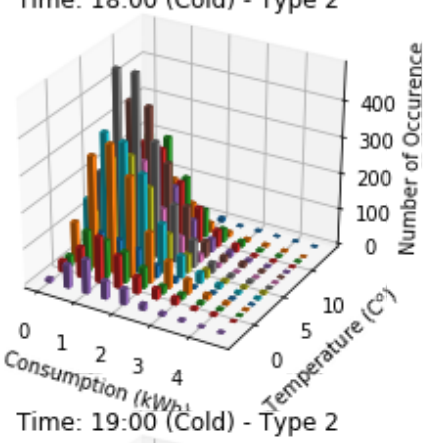
Time: 18:00 (Cold) - Type 3



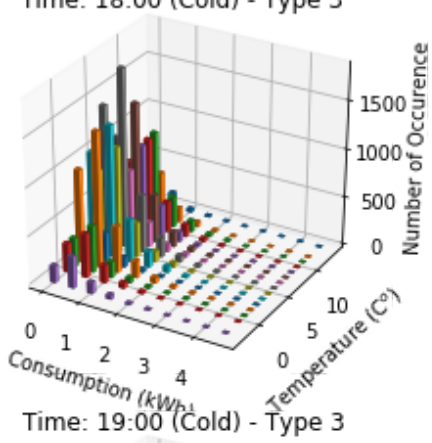
Time: 18:00 (Cold) - Type 4



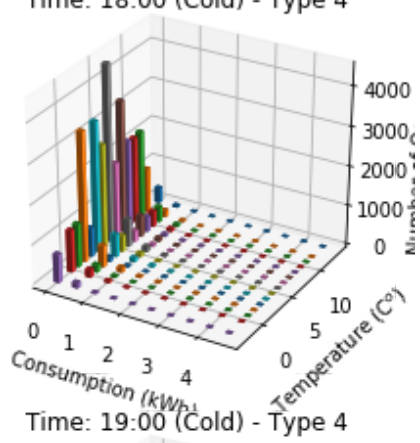
Time: 19:00 (Cold) - Type 1



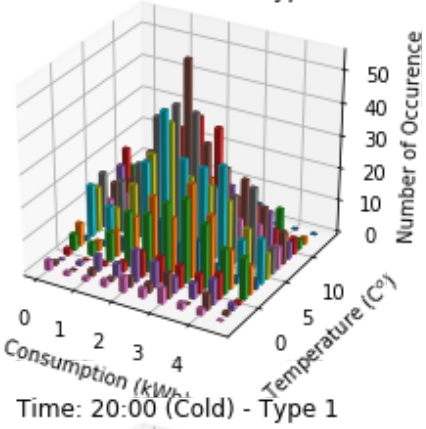
Time: 19:00 (Cold) - Type 2



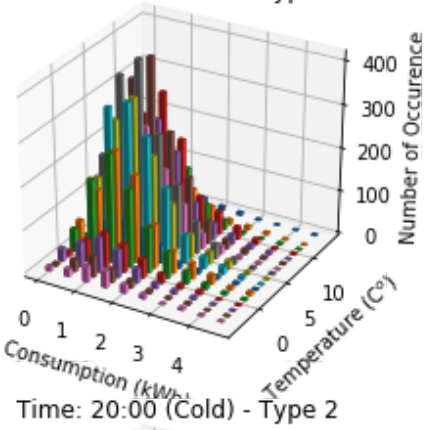
Time: 19:00 (Cold) - Type 3



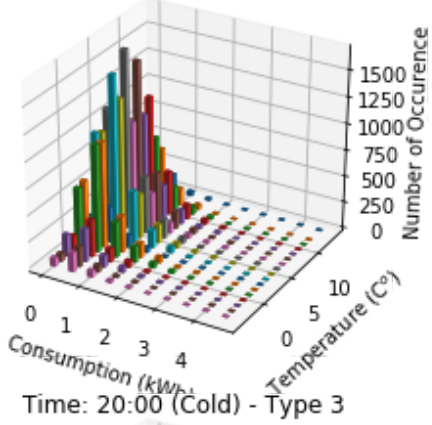
Time: 19:00 (Cold) - Type 4



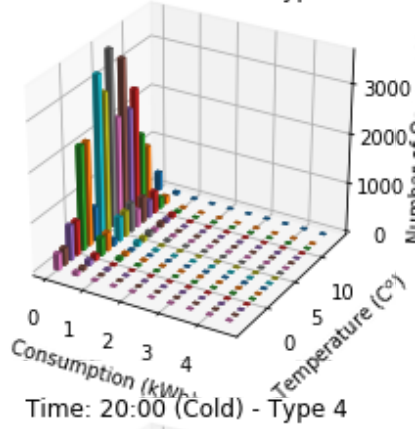
Time: 20:00 (Cold) - Type 1



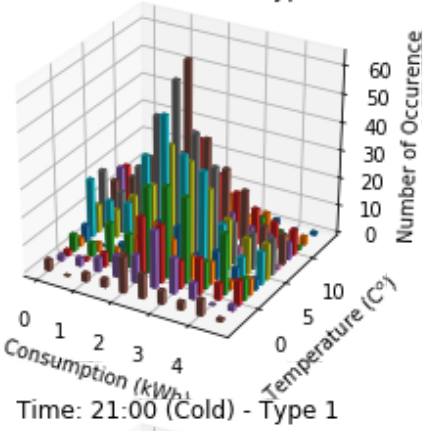
Time: 20:00 (Cold) - Type 2



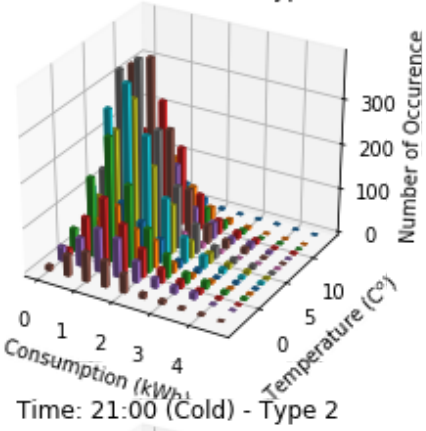
Time: 20:00 (Cold) - Type 3



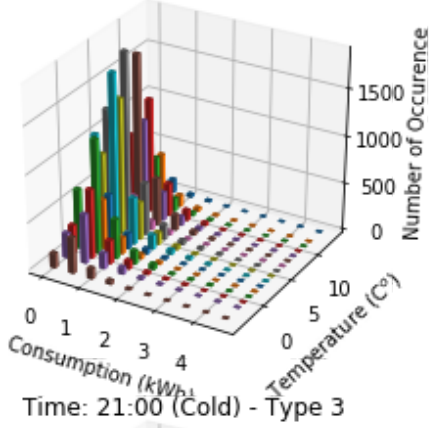
Time: 20:00 (Cold) - Type 4



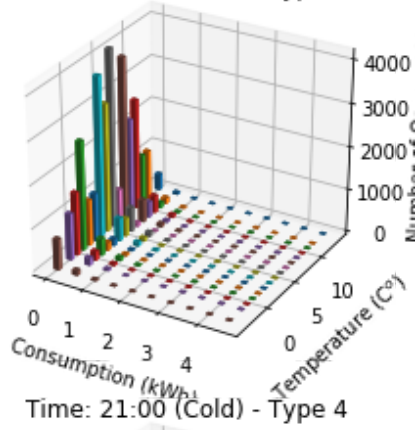
Time: 21:00 (Cold) - Type 1



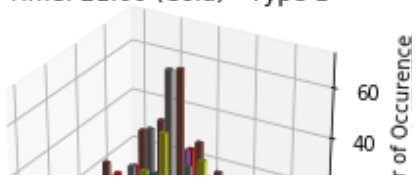
Time: 21:00 (Cold) - Type 2

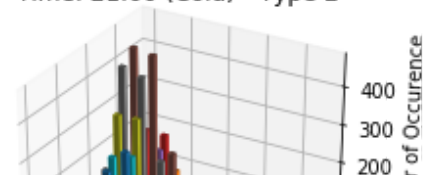


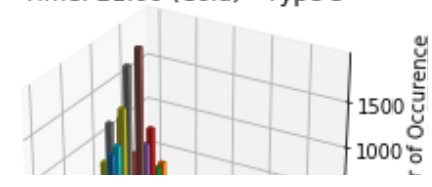
Time: 21:00 (Cold) - Type 3

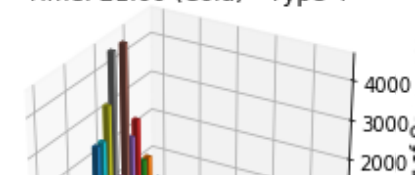


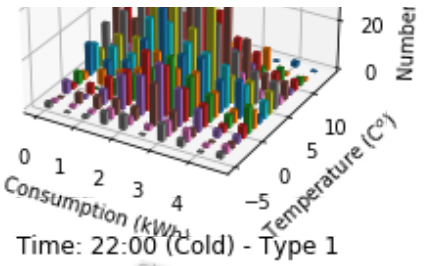
Time: 21:00 (Cold) - Type 4



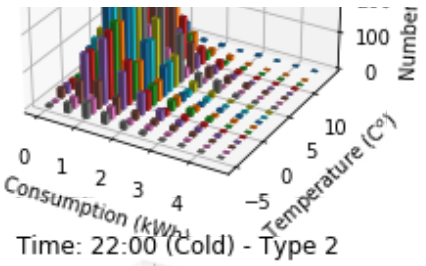




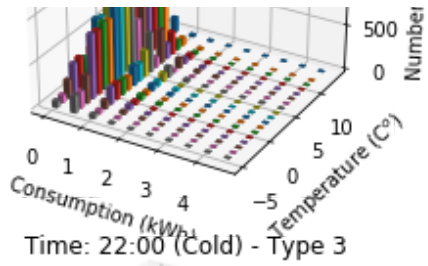




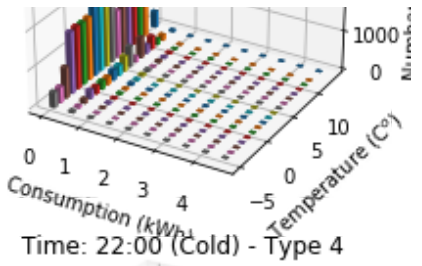
Time: 22:00 (Cold) - Type 1



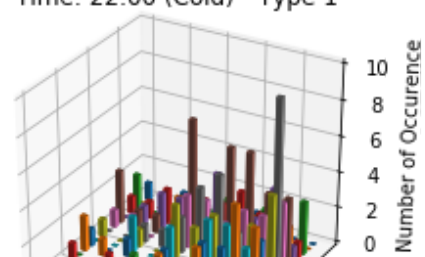
Time: 22:00 (Cold) - Type 2



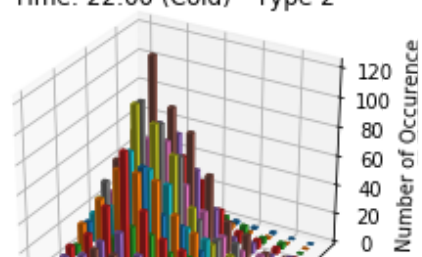
Time: 22:00 (Cold) - Type 3



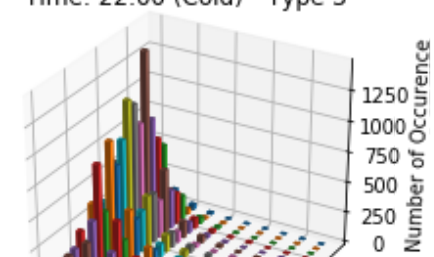
Time: 22:00 (Cold) - Type 4



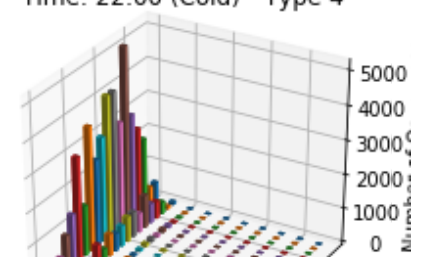
Time: 23:00 (Cold) - Type 1



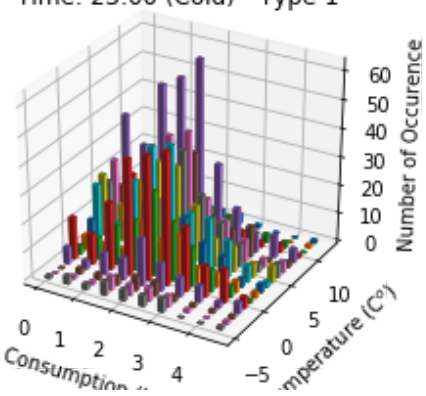
Time: 23:00 (Cold) - Type 2

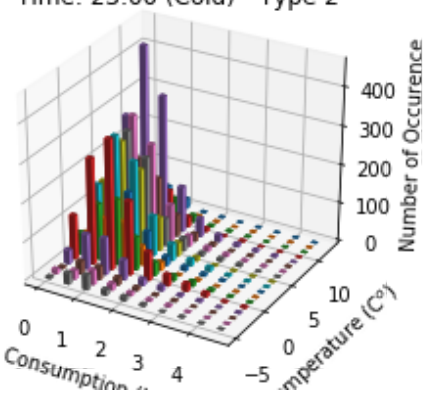


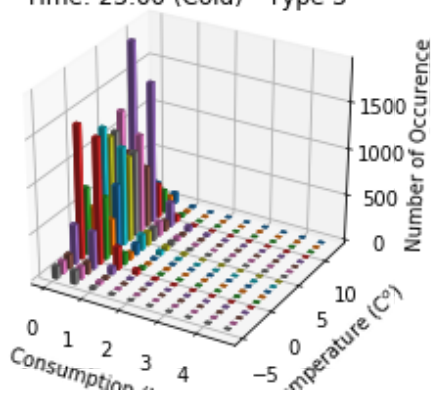
Time: 23:00 (Cold) - Type 3

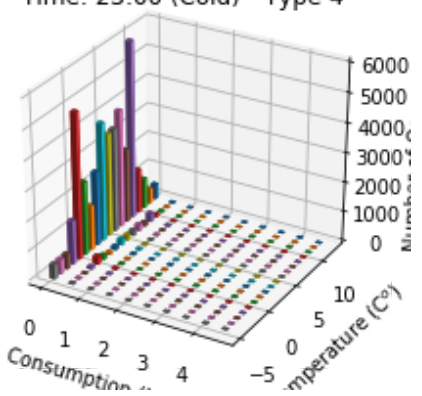


Time: 23:00 (Cold) - Type 4

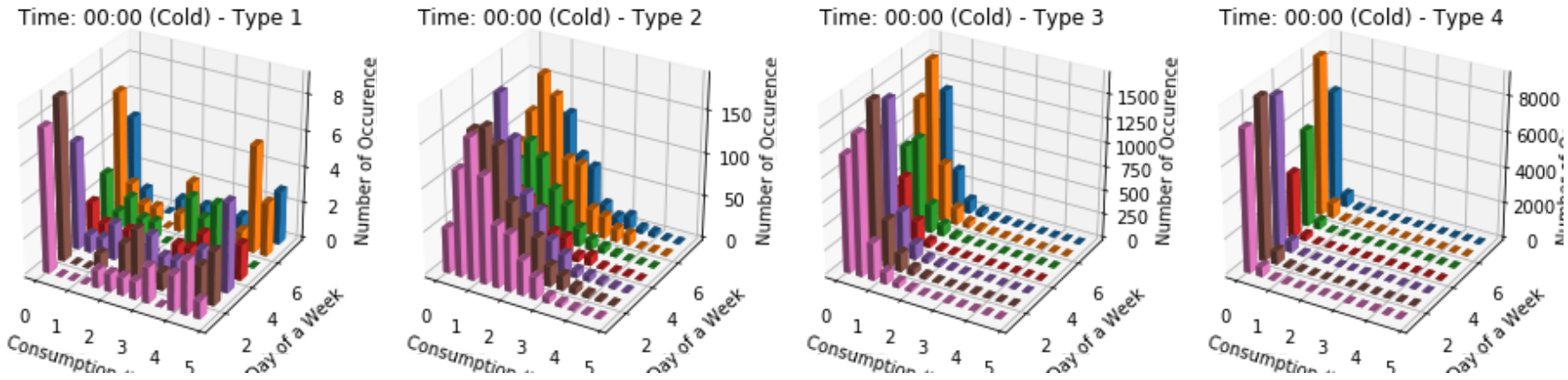






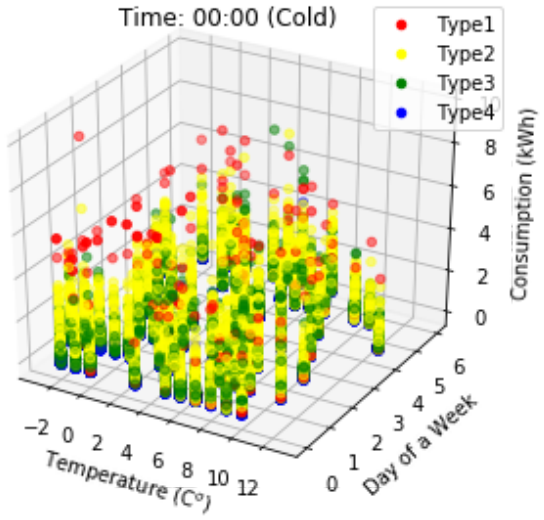


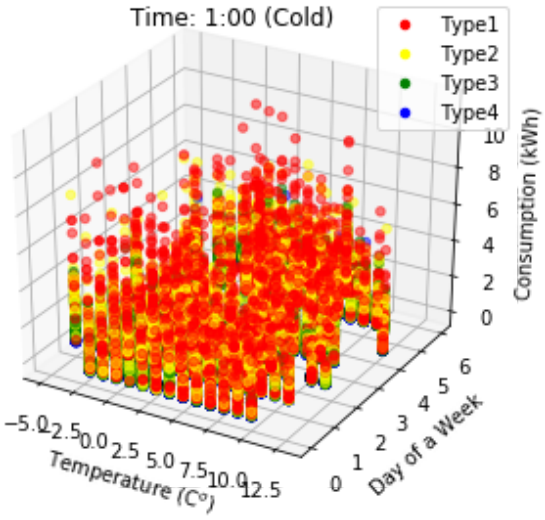
In [34]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 4 # cluster numbers
cons_type = ['Type 1', 'Type 2', 'Type 3', 'Type 4']
fig_all = plt.figure(figsize = (13,72))
for i in range(24):
    if i <= 9:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i)
+ ':00:00')]['GMT']).dt.dayofweek # add temperature column to dataframe
            for day in reversed(range(7)):
                t3 = t2[t2['DayofWeek'] == day] # select the data with a specific day of a week
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=13, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(13) * (day+1)
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(13) * 0.2
                dy = np.ones(13) * 0.5
                # dz = t3[0] / ((kmeans.labels_ == sorted_label[k][0]).sum() * (t2['DayofWeek'] == day).sum
())
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 4 + k + 1
            ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Day of a Week')
            ax.set_zlabel('Number of Occurence')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        for k in range(n):
            ax = fig_all.add_subplot(24, 4, i * 4 + k + 1, projection = '3d')
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':0
0:00')]['GMT']).dt.dayofweek # add temperature column to dataframe
            for day in reversed(range(7)):
                t3 = t2[t2['DayofWeek'] == day] # select the data with a specific day of a week
                t3 = np.histogram(t3[t3.columns[1:-1]].values.flatten(), bins=13, range=(0,5))
                xpos = t3[1][:-1]
                ypos = np.ones(13) * (day+1)
                num_elements = len(xpos)
                zpos = np.zeros(num_elements)
                dx = np.ones(13) * 0.2
                dy = np.ones(13) * 0.5
                # dz = t3[0] / ((kmeans.labels_ == sorted_label[k][0]).sum() * (t2['DayofWeek'] == day).sum
())
                dz = t3[0]
                pl = ax.bar3d(xpos, ypos, zpos, dx, dy, dz, alpha = 0.5, zsort='max')
                pl._sort_zpos = i * 4 + k + 1
            ax.set_title('Time: ' + str(i) + ':00' + ' (Cold) - ' + str(cons_type[k]))
            ax.set_xlabel('Consumption (kWh)')
            ax.set_ylabel('Day of a Week')
            ax.set_zlabel('Number of Occurence')
plt.tight_layout()
```

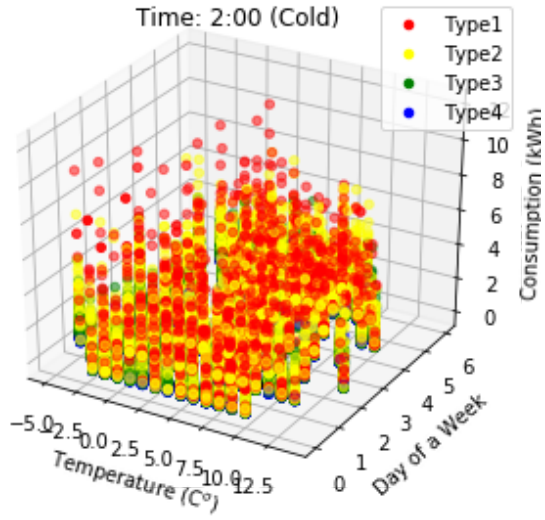


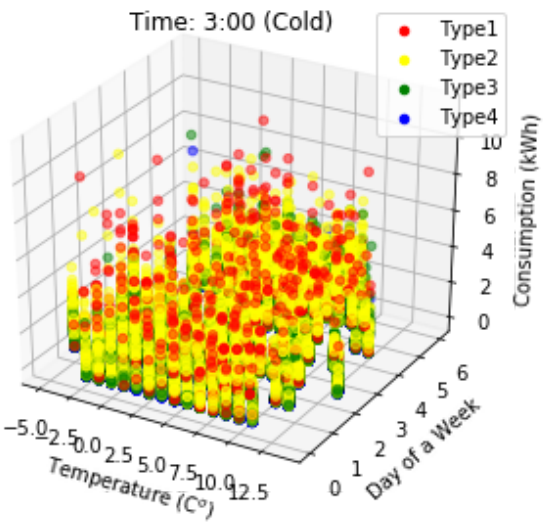
Comparing the 4 clustering with 3 clustering cases, we find that adding one clustering makes the transition between one cluster to the next cluster more slower, i.e., we have one more culstering with a relatively nice distributed shape. From a visualized perspective we have more nice distribution shapes, but it also increase the complexity of labeling a household type. So after consideration, we chose 3 clusters for further analysis, since it's conveniout to handle and can also well describe different clusters.

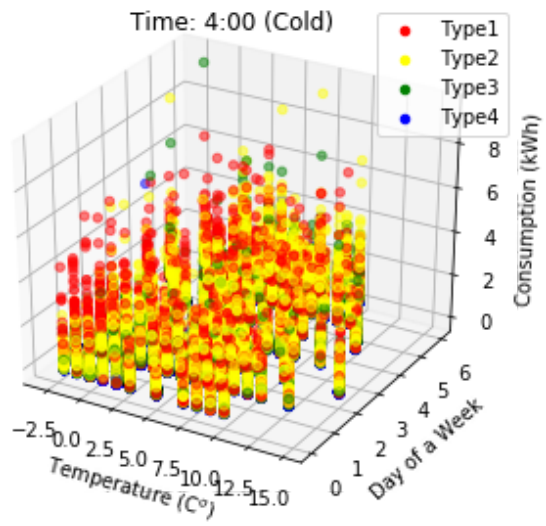
In [456]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
n = 4 # cluster numbers
cons_type = ['Type1', 'Type2', 'Type3', 'Type4']
color_type = ['red', 'yellow', 'green', 'blue']
fig_all = plt.figure(figsize = (13,30))
for i in range(24):
    if i <= 0:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
        for k in range(n):
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i) + ':00:00'
)]['TempC'].values) # add temperature column to dataframe
            t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains('0' + str(i)
+ ':00:00')]['GMT']).dt.dayofweek # add day of a week column to dataframe
            ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[1]], color = color_type[k], labe
l = cons_type[k])
            for j in range(2, 1 + sorted_label[k][1]):
                ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[j]], color = color_type[k],
alpha = 0.5)
                ax.legend()
        ax.set_title('Time: 0' + str(i) + ':00' + ' (Cold)')
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Day of a Week')
        ax.set_zlabel('Consumption (kWh)')
    else:
        x = df_tou1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]
        t = x.dropna(axis = 'columns') # data used for analyzing consumption distribution
        x.set_index('GMT', inplace = True)
        x = x.transpose().dropna() # data for the specific hour, and filtering out null value
        # PCA without standardization
        pca_nstd = PCA()
        principleComponents_nstd = pca_nstd.fit_transform(x)
        principleDf_nstd = pd.DataFrame(data = principleComponents_nstd)
        kmeans = KMeans(n_clusters = n).fit(principleDf_nstd.iloc[:,:3])
        cluster_dict = {}
        for k in range(n):
            cluster_dict[k] = (kmeans.labels_ == k).sum() # store the labels and the number of data in a dict
        sorted_label = sorted(cluster_dict.items(), key=lambda kv: kv[1])
        ax = fig_all.add_subplot(8, 3, i + 1, projection = '3d')
        for k in range(n):
            t2 = t[t.columns[np.insert(kmeans.labels_ == sorted_label[k][0], 0, True)]].copy() # select all d
ata from the n-th cluster
            t2['TempC'] = np.round(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':00:00')]['Te
mpC'].values) # add temperature column to dataframe
            t2['DayofWeek'] = pd.to_datetime(df_wea1h_nf_cold[df_tou1h_nf_cold.GMT.str.contains(str(i) + ':0
0:00')]['GMT']).dt.dayofweek # add day of a week column to dataframe
            ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[1]], color = color_type[k], labe
l = cons_type[k])
            for j in range(2, 1 + sorted_label[k][1]):
                ax.scatter(t2[t2.columns[-2]], t2[t2.columns[-1]], t2[t2.columns[j]], color = color_type[k],
alpha = 0.5)
                ax.legend()
        ax.set_title('Time: ' + str(i) + ':00' + ' (Cold)')
        ax.set_xlabel(r'Temperature (C$^o$)')
        ax.set_ylabel('Day of a Week')
        ax.set_zlabel('Consumption (kWh)')
plt.tight_layout()
```

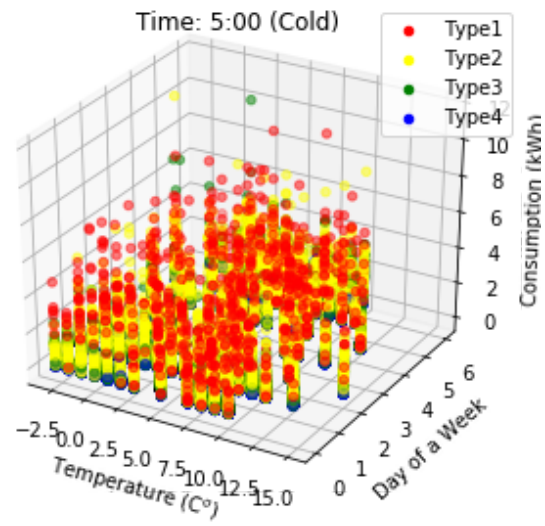

Time: 00:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

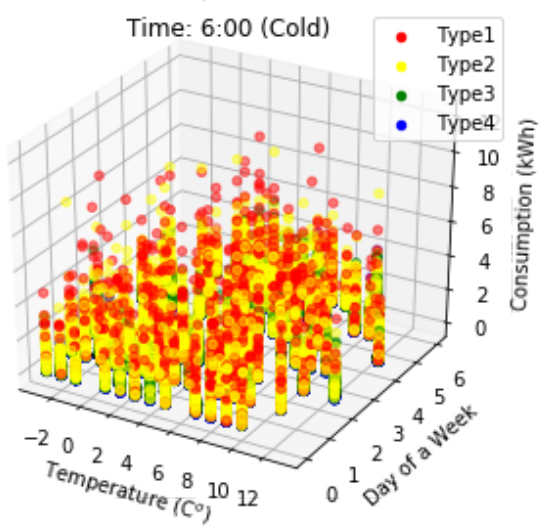

Time: 1:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

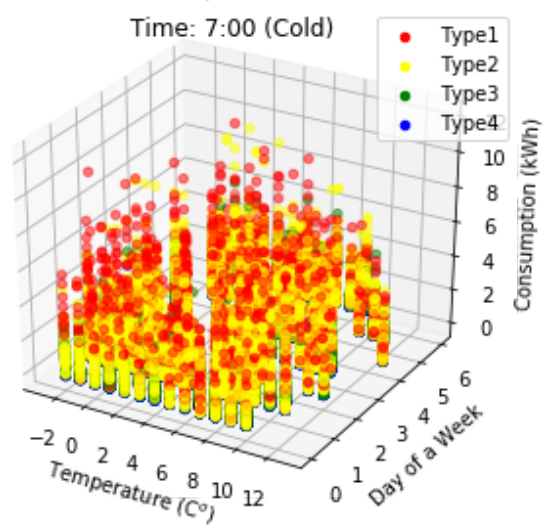

Time: 2:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

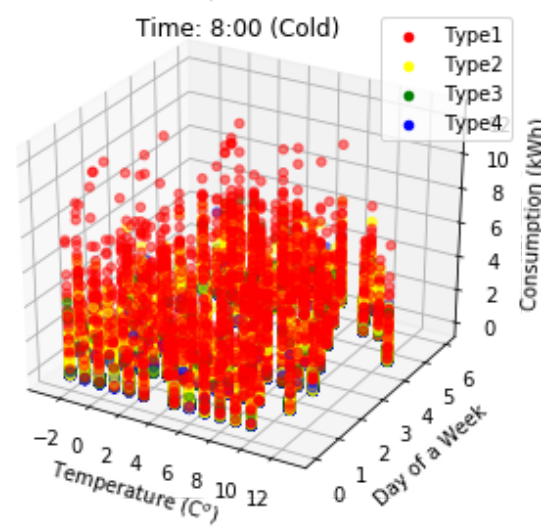

Time: 3:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

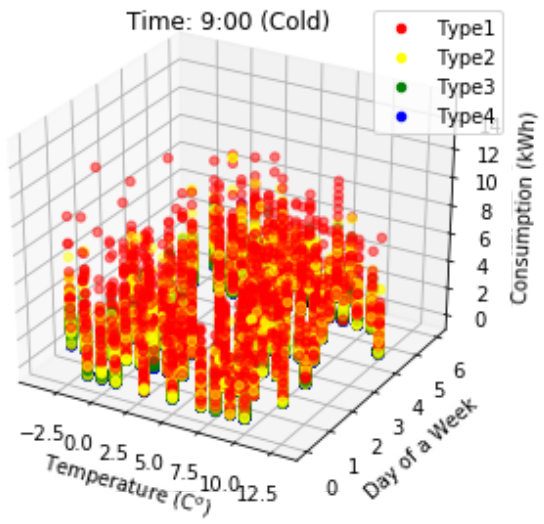

Time: 4:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

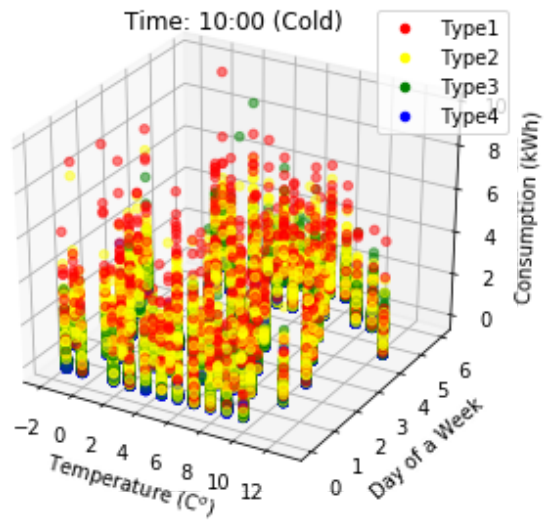

Time: 5:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

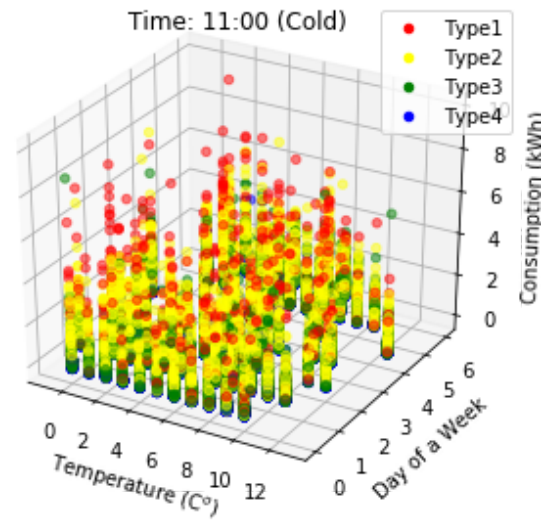

Time: 6:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

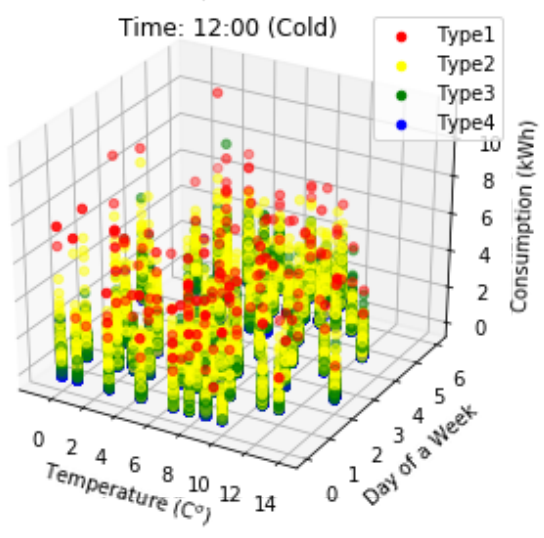

Time: 7:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

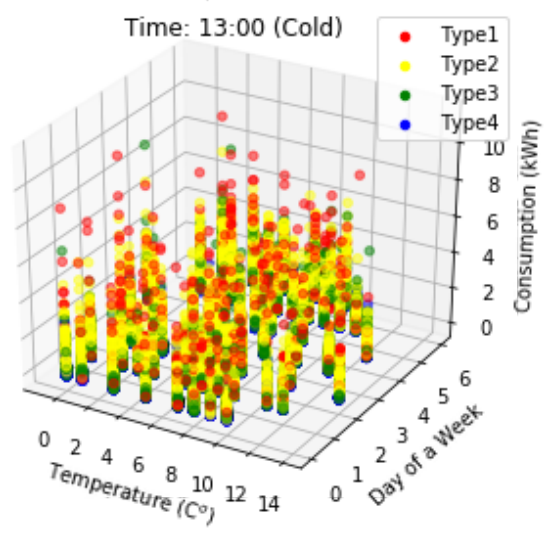

Time: 8:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

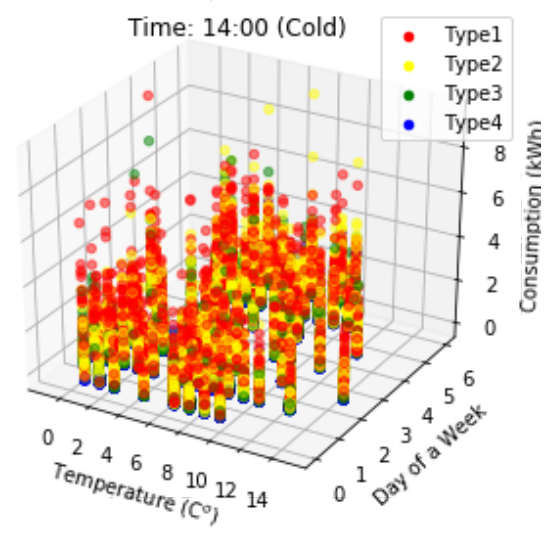

Time: 9:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

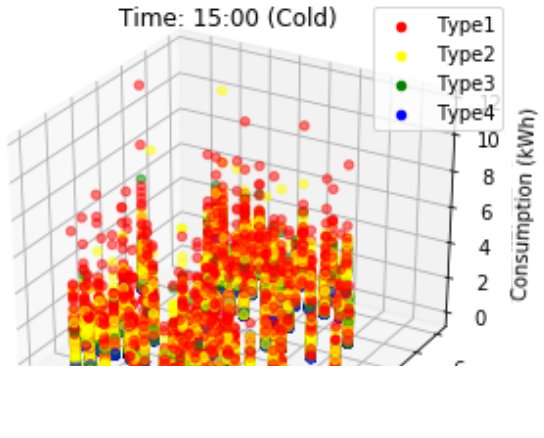

Time: 10:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

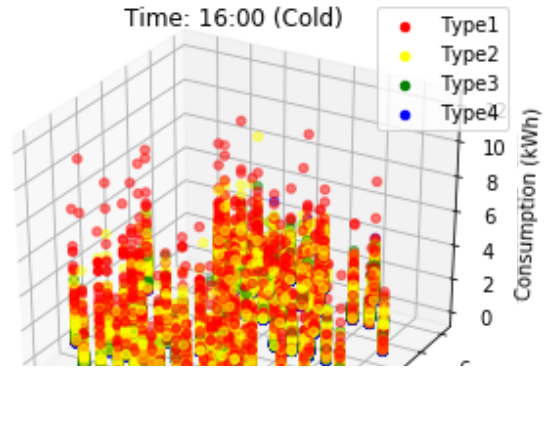

Time: 11:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

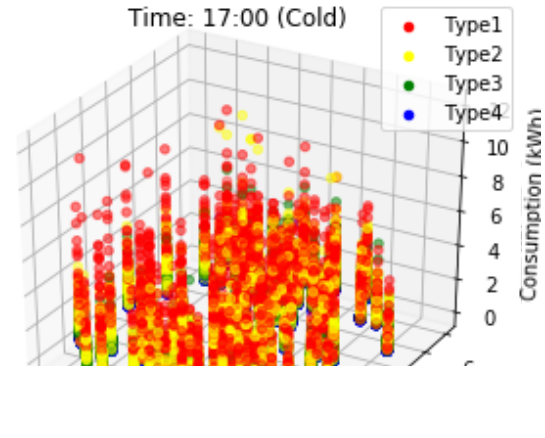

Time: 12:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)

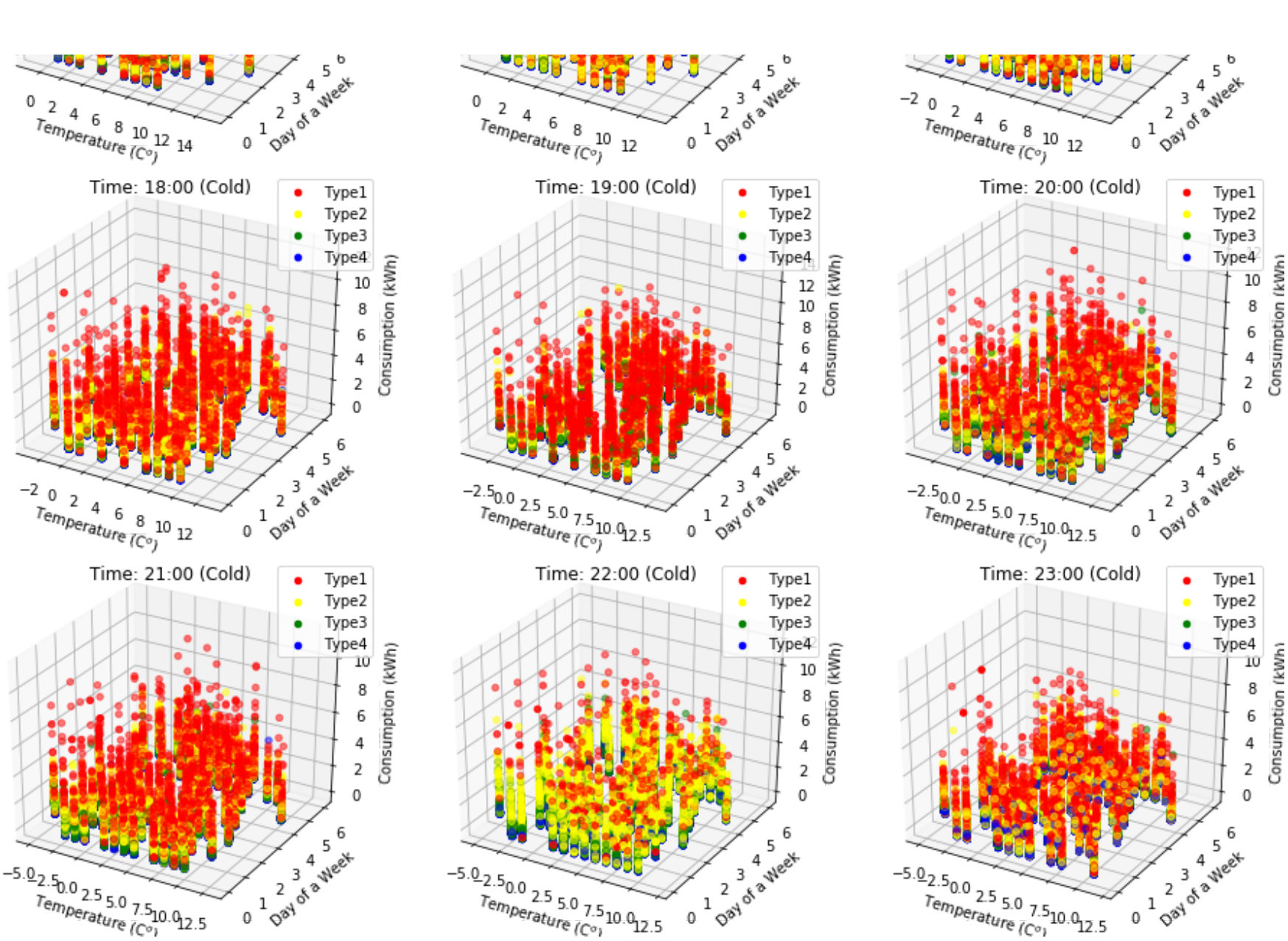

Time: 13:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)


Time: 14:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)


Time: 15:00 (Cold)
Type1
Type2
Type3
Type4
Consumption (kWh)


Time: 16:00 (Cold)
Type1
Type2
Type3
Type4
Consumption (kWh)


Time: 17:00 (Cold)
Type1
Type2
Type3
Type4
Consumption (kWh)


Time: 18:00 (Cold)
Time: 19:00 (Cold)
Time: 20:00 (Cold)
Time: 21:00 (Cold)
Time: 22:00 (Cold)
Time: 23:00 (Cold)
Type1
Type2
Type3
Type4
Temperature (C°)
Day of a Week
Consumption (kWh)