demand$_{a}cepDocumentation$

Chintan Pathak, Yohan Min, Atinuke Ademola-Idowu

Jun 25, 2019

Contents:

CHAPTER 1

Introduction

The project demand_acep aims to make sense of the data collected by power meters at some facilities at the Poker Flat Research Range (PFRR) managed by Alaska Center of Energy and Power (ACEP). The project resulted in an open-source Python 3 package, that implements a data pipeline for high resolution power meter data to easy access to data for data analytics and research. The quick overview of the data pipeline can be seen below:

Data-based approach for demand-charge reduction using meter aggregation

Atinuke A. Ademola-Idowu, Yohan Min, Chintan Pathak

PROJECT OBJECTIVES

- A data-pipeline in an open-source python package for high resolution power meter data to facilitate ease of access for data analytics and research.
- Estimating the potential demand-charge reduction by aggregating the 4 power meters into a virtual meter.

QUICK FACTS ABOUT THE DATA

> Site Owner: University of Alaska, Fairbanks. (UAF)

> Project Sponsor: Alaska Center for Energy and Power (ACEP) at UAF.

> Site Name: Poker Flats Research Range

> Site Function: NASA rocket research and launching

> Electric Utility: Golden Valley Electric Association (GVAE)

> Data size: 900+ GB

> Original data format: NetCDF 3 Classic

> Original data resolution: ~ 7 Hz (about 8 samples/sec)

> Down-sampled data resolution: 0.016 Hz (1 min.)

> Number of meters: 4

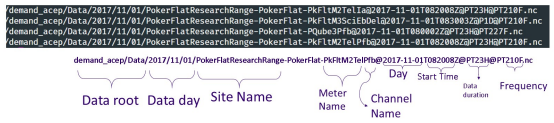
> Meter type: Watts-On and PQube

> Measurement duration: Nov 2017 – April 2019

> Number of measurements per meter: 40

> Measurement type (per phase): energy received, energy delivered, net total energy, active power, reactive power, apparent power, current, voltage L-L, Voltage L-N, frequency, power factor, Sliding average 15 mins real power.

DATA PIPELINE



Extraction

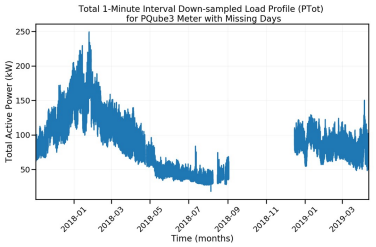
- Get date/dates from user
- Find the files at path
- Parse the channel and meter name from file name.
- Read NetCDF file using xarray in Python.

Transformation

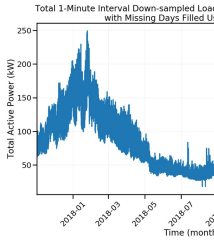
- Down-sample data to (user-specified) 1-min resolution.
- Data imputation to fill missing values.
- Handle edge cases like multiple data dumps per day, anomaly detection.
- Create data-frame by merging channels and save to disk

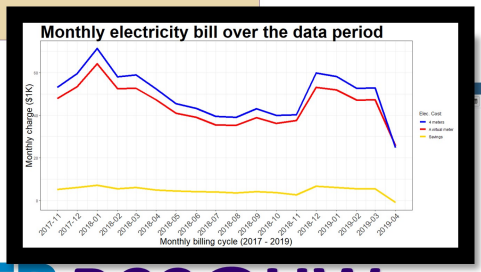
Loading

- Create database schema dynamically based on available meters and years
- Load data into timeseries database TimescaleDB using time as index.

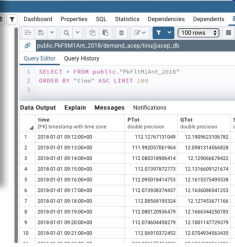


Total down-sampled power profile PQube3 – filled





Database Snap



DIRECT Data Intensive Research Enabling Clean Technologies

CLEAN ENERGY INSTITUTE UNIVERSITY of WASHINGTON

ACEP Alaska Center for Energy and Power

RCC@UW

CHAPTER 2

About the power meters

The data consists of 4 power meters. Three power meters are WattsOnMk2 and one meter is PQube. The meter names and corresponding types are listed in meter_names.txt Each meter measures around 50 channels at a sub-second resolution. The channel names and description can be found in this file. These files will have to updated if there are any changes to meter names or channels, like if more meters are added or more channels are being recorded.

CHAPTER 3

Data Years

At the time of this writing, the ACEP Power meter dataset had data from Nov 2017 to Apr 2019. As data years increase, they need to be added to the data years file.

CHAPTER 4

About the data pipeline

The data pipeline comprises of three steps:

4.1 Extract

This steps takes the data from NetCDF files and creates a dataframe with time as index and values in the NetCDF file as the only column.

4.2 Transform

This step takes the extracted data in the dataframe for each channel and down-samples the data to a lower resolution (1 minute default) and concatenates other channels to the dataframe keeping the same time index for each meter. The transformation step also does data imputation, i.e. fills in the missing values. The section *Data Imputation* describes the data imputation process in more details. The implementation of extraction and transformation is coupled and happens in the function extract_csv_for_date(), which saves the transformed data into a csv for each year. This function also handles edge-cases like days when data download to NetCDF files happens more than once. An example extraction and transformation for a day is shown in the jupyter notebook extract_to_csv.ipynb. An extraction and transformation for multiple days can be done in parallel and is shown in jupyter notebook test_multiprocessing_csv.ipynb. The extraction, transformation and saving of down-sampled data to csv currently takes around 2h 10min on a 28 core, 2.4 GHz system.

Note: Since the "Load" step is dependent on the csv files existing in the path, the "Transform" step should finish before the "Load" step is initiated.

4.3 Load

This step takes the transformed data in csv files and copies it into the timeseries database - TimescaleDB. TimescaleDB installation instructions for Ubuntu are pretty straightforward and it is incredibly straightforward to use with docker, like so:

```
sudo docker run -d -e POSTGRES_USER=<username> -e POSTGRES_PASSWORD=<password> --name
↪<database_name> -p 5432:5432  --restart=always timescale/timescaledb
```

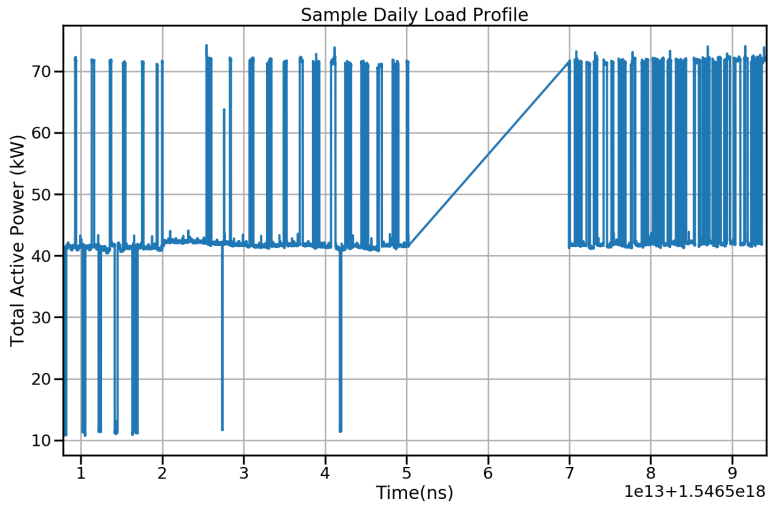
Then create a database named `demand_acep` and enable the TimescaleDB extension as described in the getting started section of the TimescaleDB docs. The database schema for insertion (copy) can be created using the function create_db_schema_from_source_files() which **deletes** all the existing data and tables and creates tables for each meter for each year, with channels as columns and time as primary key. Further, "copy" operation is preferred over the "insert" since it is much faster and can be done for full resolution data too efficiently (Read here about the risks and care in using copy over insert in postgresql database). Further, a Go utility timescaledb-parallel-copy is used to copy the data to the database in parallel. The function parallel_copy_data_for_date() prepares the command for timescaledb-parallel-copy and copies the data. This command is run with the "skip-header" option to ignore the first line of each day csv file, as that date-time is repeated with the previous day. Function parallel_copy_data_for_dates() is a wrapper around the parallel_copy_data_for_date() function and does the copying for a date range. An example application of the copy operation can be seen in the jupyter notebook timescale_parallel_copy.ipynb. The parallel copy takes 6min 18s on a 28 core, 2.4 GHz system.

CHAPTER 5

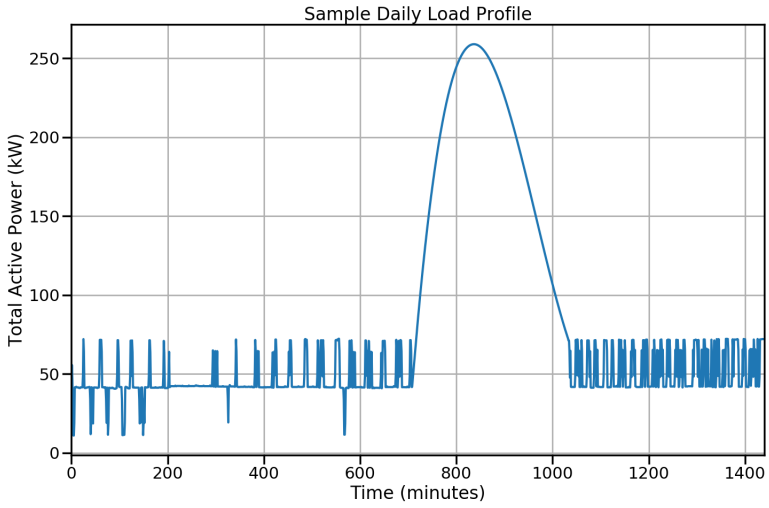
Data Imputation

Data Imputation is a process by which missing data points are filled. The ACEP measurement data is in the form of a time series, with missing data points ranging from seconds to months. Several python modules for data imputation were reviewed but none could cater to the specific demands of the ACEP time series data. In particular, using the linear, quadratic or cubic 1-d interpolation function interp1d() from the scipy module give unsatisfactory results as shown below:

Linear Interpolation.



Spline Interpolation.



This is because the function interp1d() function uses the a single data point before and after the missing data points to interpolate. To fix this, a new interpolation function data_impute() was created to extend the capabilities of the interp1d() function by selecting multiple data points before and after the missing data points with the number of data points selection equal to the number of missing data points. The data_impute() function also handles edge cases such as when the number of data points before and after the missing data points is less than the number of missing data points. In this case, the function selects the available data points.

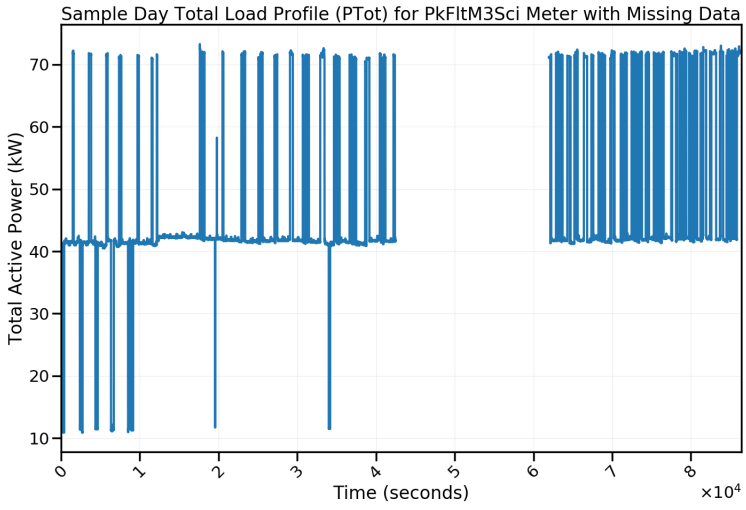
Depending on the duration of the missing data points, the data_impute() can be used as it is or coupled with other functions as described below.

5.1 Short Duration Missing Data Points

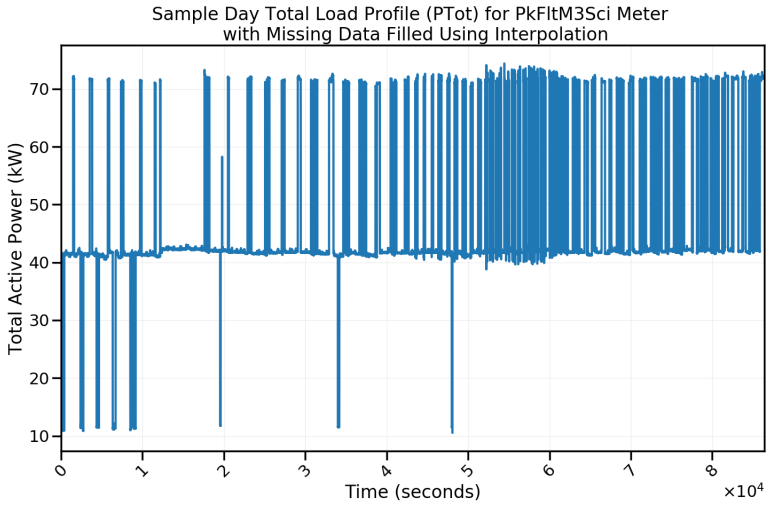
Missing data points are termed short duration if the number of consecutive missing points sums up to less than 24hrs. The pandas DataFrame containing this data can be passed to the interpolation function data_impute() for filling.

An example of a short duration missing data imputation for a sample is shown in the jupyter notebook test_data_impute.ipynb and the results displayed in the figures below. In this example, about 3 hrs of data is missing.

Before Filling



After Filling



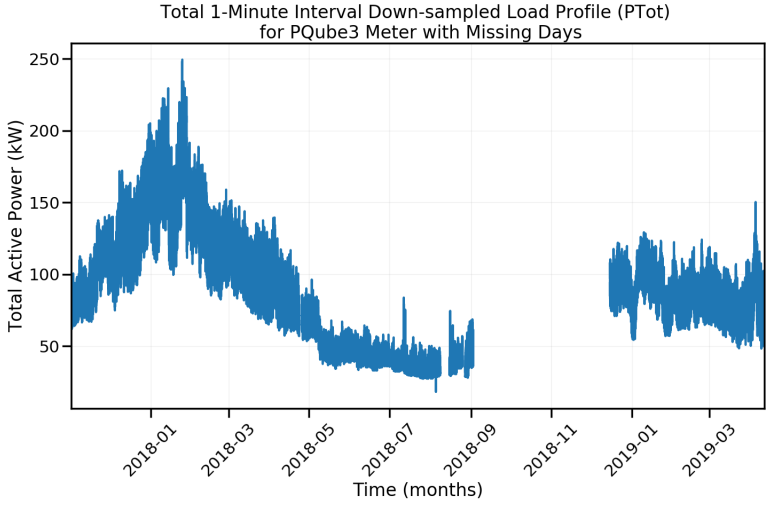
5.2 Long Duration Missing Data Points

When the number of consecutive missing data points sum up to more than 24 hrs, that is, ranges from days to months, it is termed a long duration missing data. This scenario is handled a bit differently from the short duration missing data as the data imputation is performed after the data has been inserted into the database. The reason for this is that when processing the measurement data through the data pipeline, the days with the missing data is unknown.

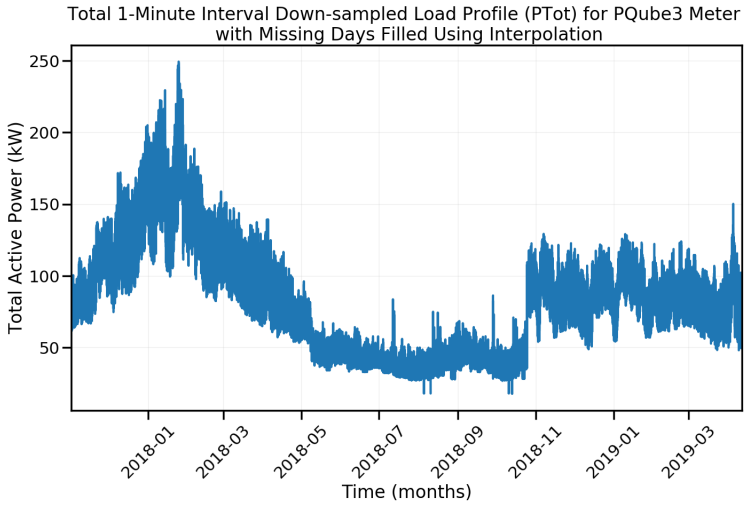
The function long_missing_data_prep() is used to pre-process the full data downloaded from the time series data base by inserting the missing data points time index and assigning a value of NaN. This preprocessed data can then be passed into the interpolation function data_impute() for filling.

An example of a long duration missing data imputation is shown in the jupyter notebook test_large_missing_data.ipynb and the results displayed in the figures below. In this example, he larges duration of consecutive missing points sums up to about 3 months.

Before Filling



After Filling



CHAPTER 6

Configuration

The configuration is specified in one file and then called in functions, as it makes project maintainence easier and promotes re-use. A separate configuration file can be created for production and test environments. A sample configuraiton file is as below:

> **Warning:** The example config file below shows the database username password being declared here. Therefore, this file if used as is, should never be committed to git/github/version control system and not be exposed to the internet.
>
> Bad things can happen otherwise!

6.1 Sample `config.py`

```
"""
This file has the global configuration so we do not have to repeat those in every␣
↪function.

We can create separate configuration files for production and test system, but they␣
↪should atleast define the following:

Attributes
----------
DATA_ROOT: string
    This is the absolute path of data.

METADATA_PATH: string
    This is the absolute path of the metadata for our data. The metadata contains␣
↪information about channels per meter, names and years of meters etc.
    Sample metadata files are found in `https://github.com/demand-consults/demand_
↪acep/tree/master/demand_acep/data/properties`.
```

(continues on next page)

(continued from previous page)

```
METER_CHANNEL_FILE: string
    This is the absolute path of the files containing the names of channels for each␣
↪meter. Each meter can have different channels it reads.
    An example `METER_CHANNEL_FILE` is here: https://github.com/demand-consults/
↪demand_acep/blob/master/demand_acep/data/properties/NetCDF%20Meter%20File
↪%20Generation%20Matrix%20Copy%20Poker%20Flats.xlsx
    If the format of this file changes, then the parsing logic below will have to␣
↪change accordingly.

DATA_YEARS_FILE: string
    This is the absolute path of the data years file. This file lists all the years␣
↪that we have the data for. The code then creates a new table for each meter for␣
↪each year.
    A sample data years file is here: https://github.com/demand-consults/demand_acep/
↪blob/master/demand_acep/data/properties/data_years.txt

METER_NAMES_FILE: string
    This is the absolute path of the meter names file. This file lists the names of␣
↪the all the meters and their type.
    A sample `METER_NAMES_FILE` is here: https://github.com/demand-consults/demand_
↪acep/blob/master/demand_acep/data/properties/meter_names.txt

METER_NAMES: list
    This list contains the names of the meters.

METER_CHANNEL_DICT: dictionary
    This dictionary should contain the available meter names as keys and the channels␣
↪for that meter as values for the corresponding.
    The generation of this dictionary is related to the structure of the `METER_
↪CHANNEL_FILE`.

DATA_YEARS: list
    This list contains the years of the data.

DATA_START_DATE: datetime.datetime
    The start date for the data to process.

DATA_END_DATE: datetime.datetime
    The end date for the date to process.

DB_NAME: string
    The name of the database to create the schema in. This database will store the␣
↪processed data, with one table per meter per year.

tsdb_pc_path: string
    The absolute path of the timescaledb-parallel-copy Go executable.

DB_USER: string
    The username to connect to the TimescaleDB with

DB_PASSWORD: string
    The password for the TimescaleDB database.

SAMPLE_TIME: string
    The argument needed to downsample the data, 1T = 1 min etc. Please refer to the␣
↪pandas `resample` documentation here: https://pandas.pydata.org/pandas-docs/stable/
↪reference/api/pandas.DataFrame.resample.html
```

(continues on next page)

(continued from previous page)

```
"""

import pandas as pd
import os, sys
import datetime

# This is for production environment
# The tests will define these paths separately
print("Config imported")
DATA_ROOT = "/gscratch/stf/demand_acep/Data"

dirname = os.path.dirname(__file__)
METADATA_PATH = os.path.join(dirname, "data/properties/")
METER_CHANNEL_FILE = os.path.join(METADATA_PATH, "NetCDF Meter File Generation Matrix␣
↪Copy Poker Flats.xlsx")
DATA_YEARS_FILE = os.path.join(METADATA_PATH, "data_years.txt")
METER_NAMES_FILE = os.path.join(METADATA_PATH, "meter_names.txt")

# Get meter names from meter names file
meter_names_df = pd.read_csv(METER_NAMES_FILE)
print(meter_names_df)
METER_NAMES = meter_names_df['meter_name'].tolist()

# Read in files containing channel names for WattsOn
meter_channel_metadata_WattsOnMetadata = pd.read_excel(METER_CHANNEL_FILE, sheet_name=
↪"WattsOnMk2")
meter_channel_metadata_PQube = pd.read_excel(METER_CHANNEL_FILE, sheet_name="PQube3PF
↪")

# Extract channel names

##########################################################################
# TODO: Create a metadata file with channel names for every meter
# ---------------------------------------------------------------
# this is hardcoded to read upto line 48,
# as the file contains other lines at the end. This should be changed to a
# file containing channels per meter, as different meters can have different
# channels and that file should be a source of truth across applications,
# the new database schema is created when this file changes
##########################################################################

# # Additional column name time is added to store the timestamp of measurement
# channel_names = ['time'] + list(meter_details['Channels'][:48])
# # Extract name of meters
# # TODO: Change this part to remmove the hardcoding
# #  when the metadata files are sorted
# # ---------------------------------------------------------
# meter_names = list(meter_details.columns.values)[-4:]

# Create a dictionary to store the channels per meter. So keys are the meter
# names and the values are a list of channel per meter
METER_CHANNEL_DICT = {}
# Loop across the meter_names list to add channels for each meter
for index, row in meter_names_df.iterrows():
    if row['meter_type'] == 'WattsOnMk2':
```

(continues on next page)

(continued from previous page)

```
        channel_names = meter_channel_metadata_WattsOnMetadata['Filename'][0:48]
        METER_CHANNEL_DICT[row['meter_name']] = ['time'] + list(channel_names)
    elif row['meter_type'] == 'PQube':
        channel_names = meter_channel_metadata_PQube['Filename'][0:46]
        METER_CHANNEL_DICT[row['meter_name']] = ['time'] + list(channel_names)

# Get years from the years file
years_df = pd.read_csv(DATA_YEARS_FILE)

DATA_YEARS = years_df['years'].values.tolist()


# Data start and end date
DATA_START_DATE = datetime.datetime(2017, 11, 1)
DATA_END_DATE = datetime.datetime(2019, 4, 30)

#########################################################
############ Database related configuration ############
#########################################################


# Datebase IP address
DB_ADDRESS = "localhost"
# Database port
DB_PORT = 5432
#####################################################################
############### !!!! NEVER COMMIT THE CREDENTIALS TO GIT !!!!!!!
################# Only demonstrated here as an example #########
#####################################################################
# DB username
DB_USER = "cp84"
# DB password
DB_PWD = "neotao123"
# Database name
DB_NAME = 'demand_acep'
#  path of timescaledb-parallel-copy
tsdb_pc_path = "/gscratch/stf/demand_acep/go/bin"

# Downsampling duration
# sample_time allows the user determine what time interval the data should be␣
↪resampled at
# For 1 minute - 1T, 1 hour - 1H, 1 month - 1M, 1 Day - 1D
SAMPLE_TIME = '1T'
```

CHAPTER 7

Data plots for the 4 meters

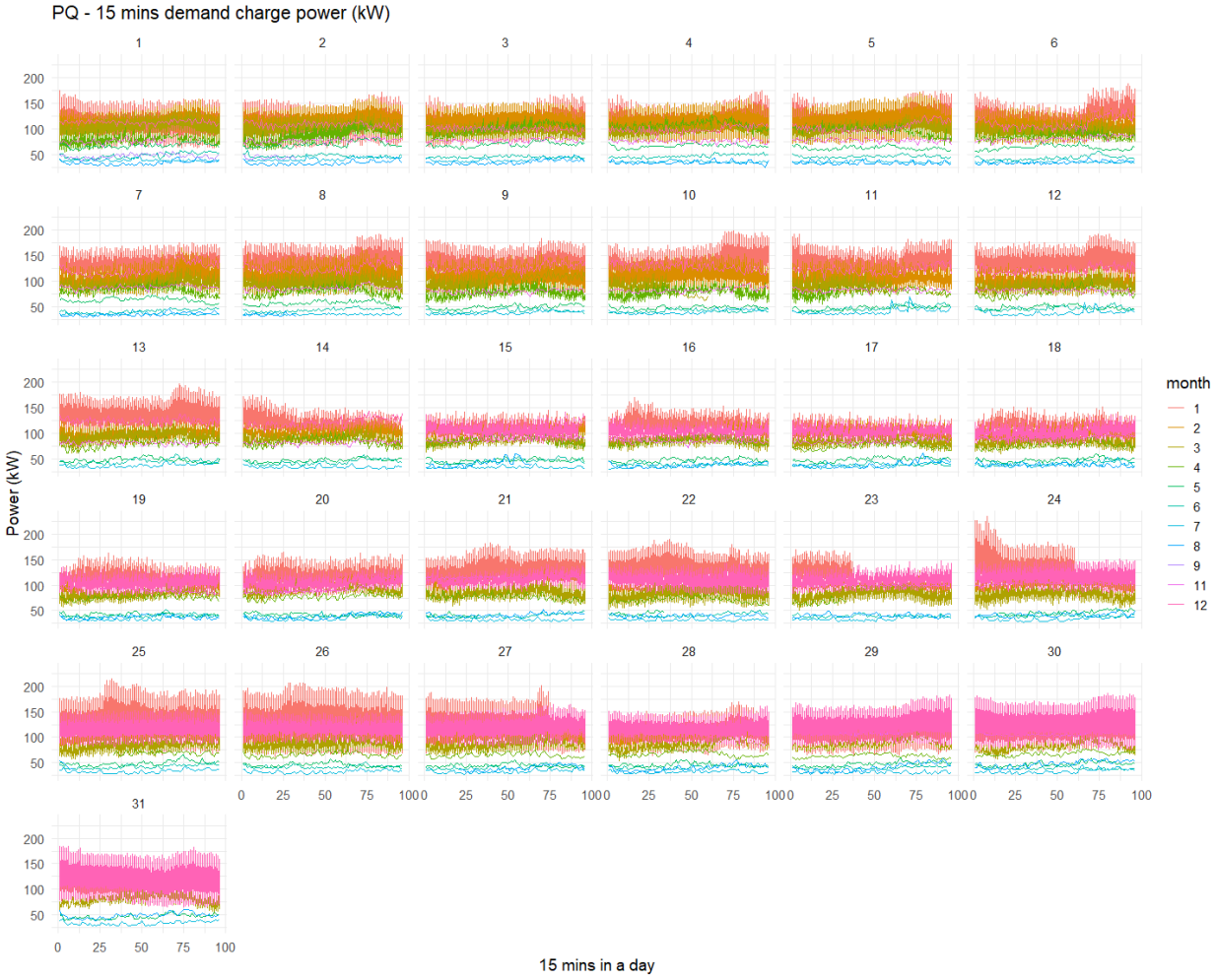
For each meter (PQ, Wat1, Wat2, and Wat3), there are four plots showing the power (kW) trends per years, months, weekdays and days to help comparing the trends (Nov. 2017 to Apr. 2019).

For each meter (PQ, Wat1, Wat2, and Wat3), there are four plots showing the power (kW) trends per years, months, weekdays and days to help comparing the trends (Nov. 2017 to Apr. 2019)

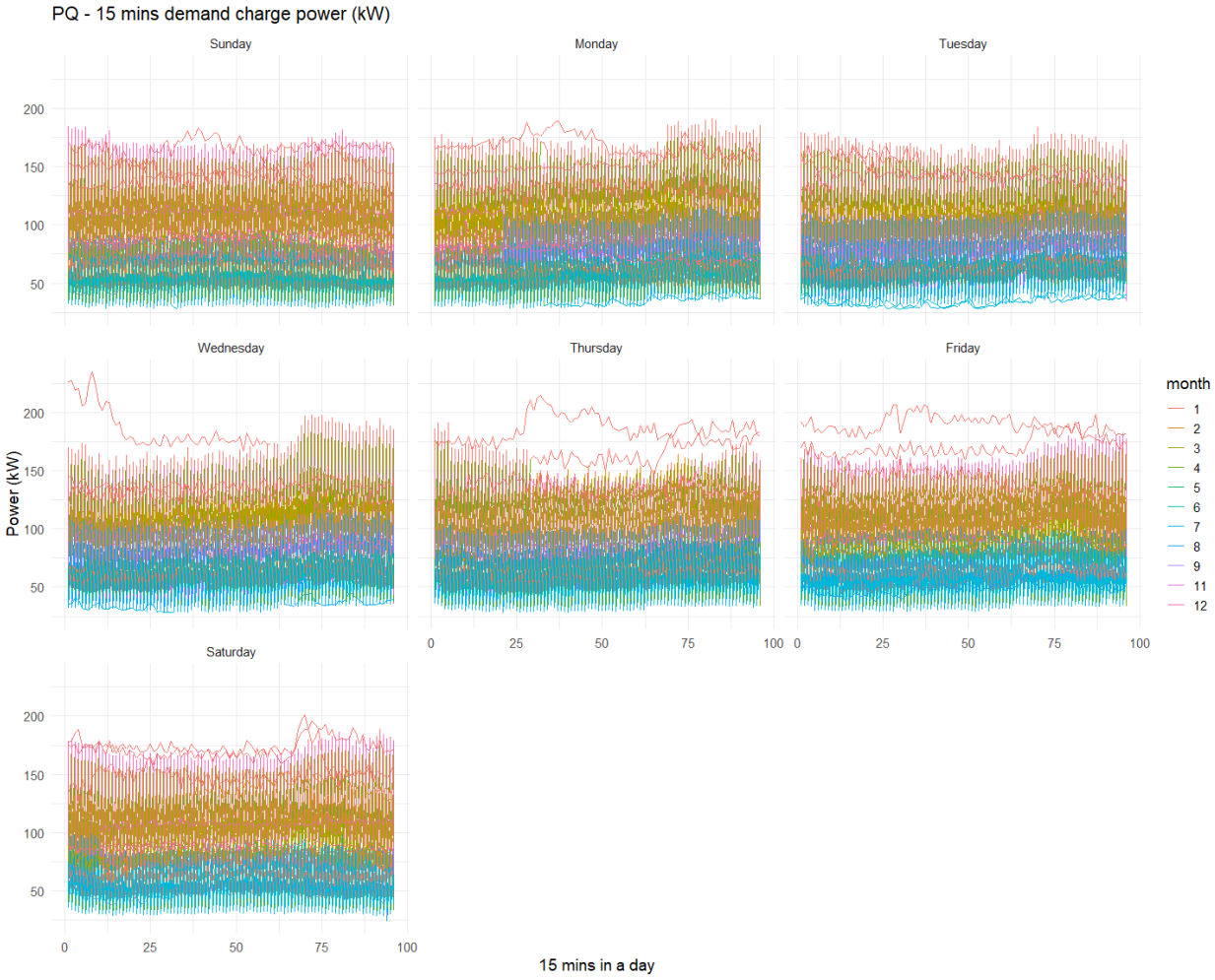
- Power (kW) trend of a day for days (Day 1 to Day 31) per month
- Power (kW) trend of a day for weekdays (Sunday to Saturday) per month
- Power (kW) trend of a day for months (Jan. to Dec.) per year
- Power (kW) trend of a month for months (Jan. to Dec.) per year

7.1 PQ

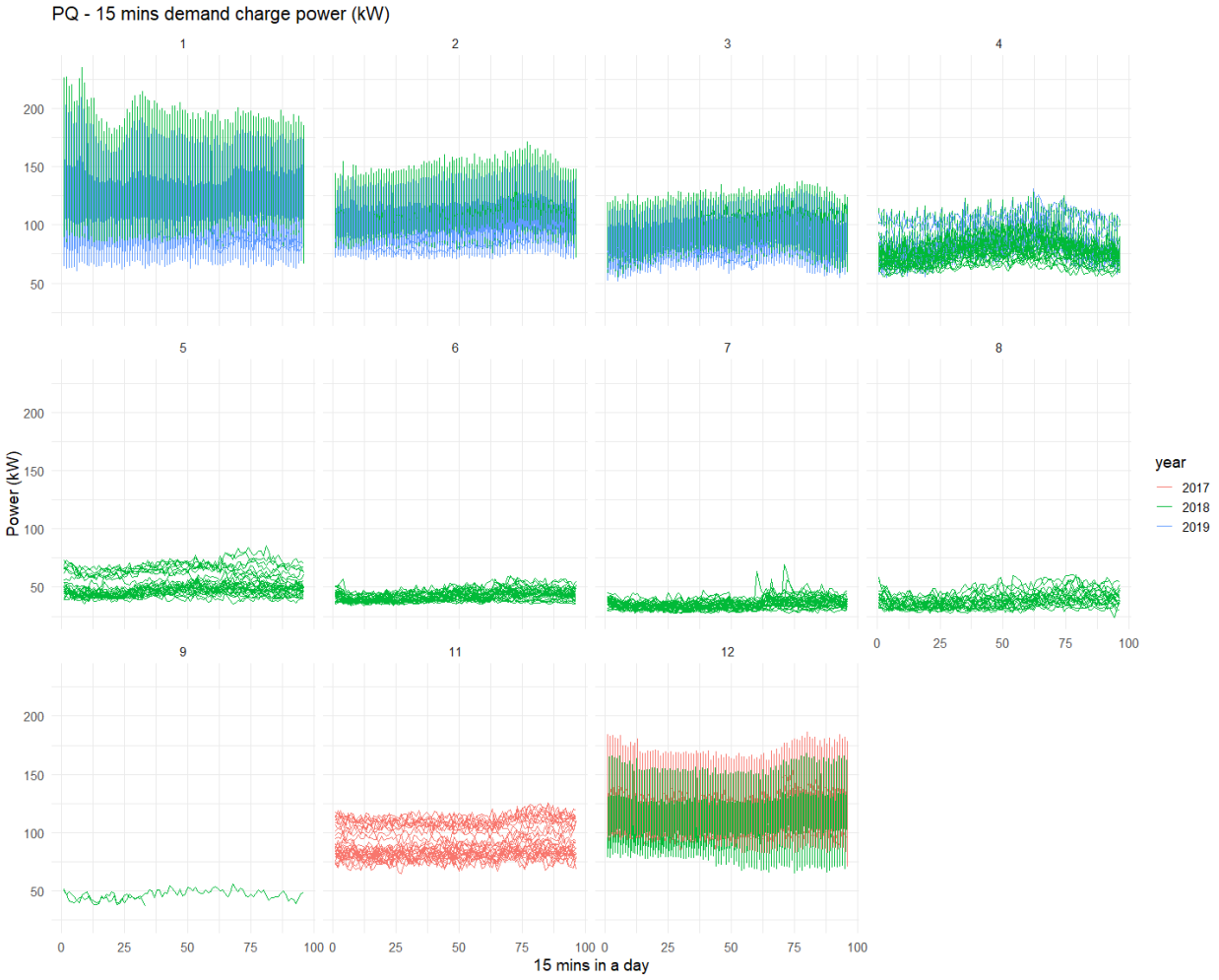
7.1.1 Power (kW) trend of a day for days (Day 1 to Day 31) per month



7.1.2 Power (kW) trend of a day for weekdays (Sunday to Saturday) per month



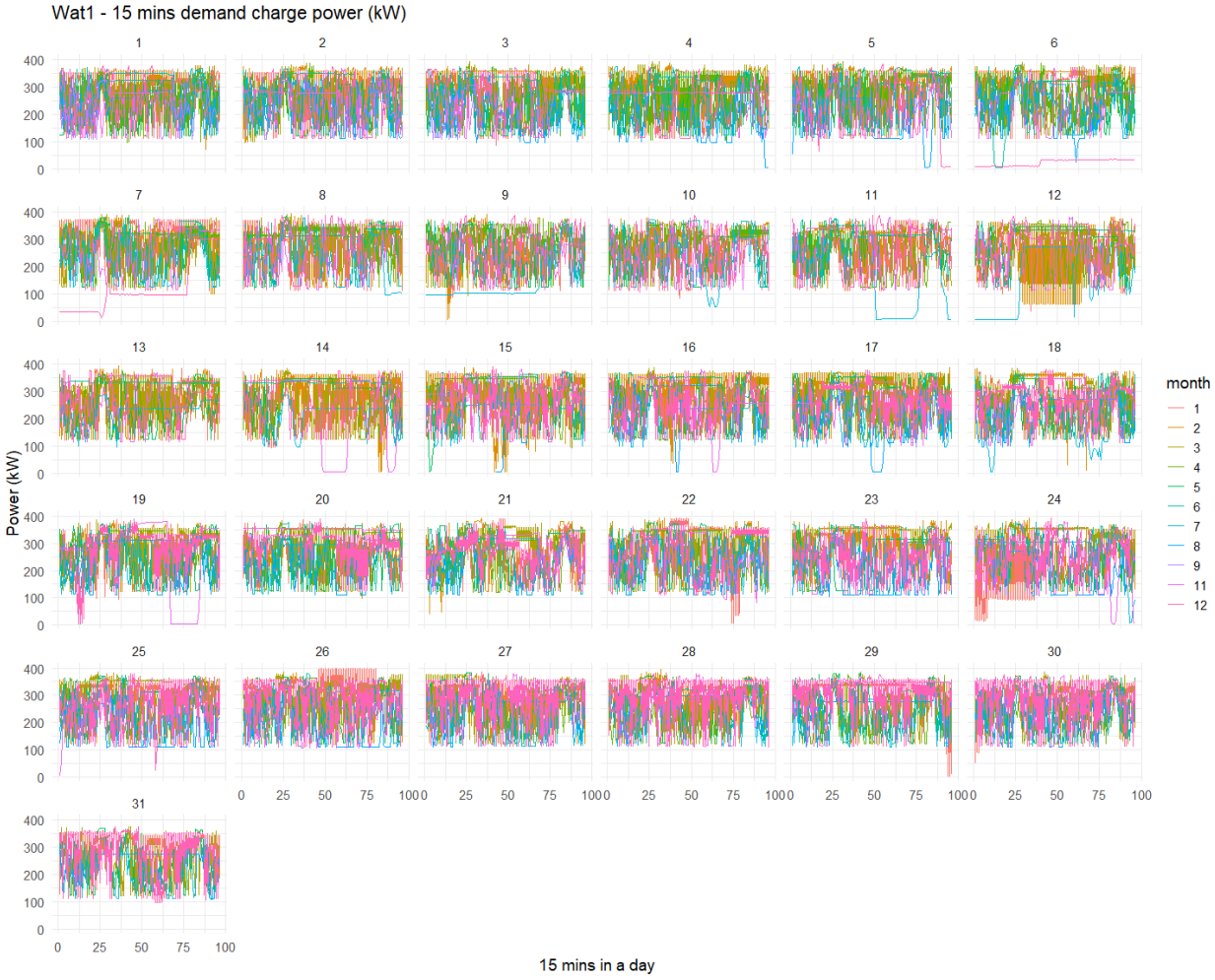
7.1.3 Power (kW) trend of a day for months (Jan. to Dec.) per year



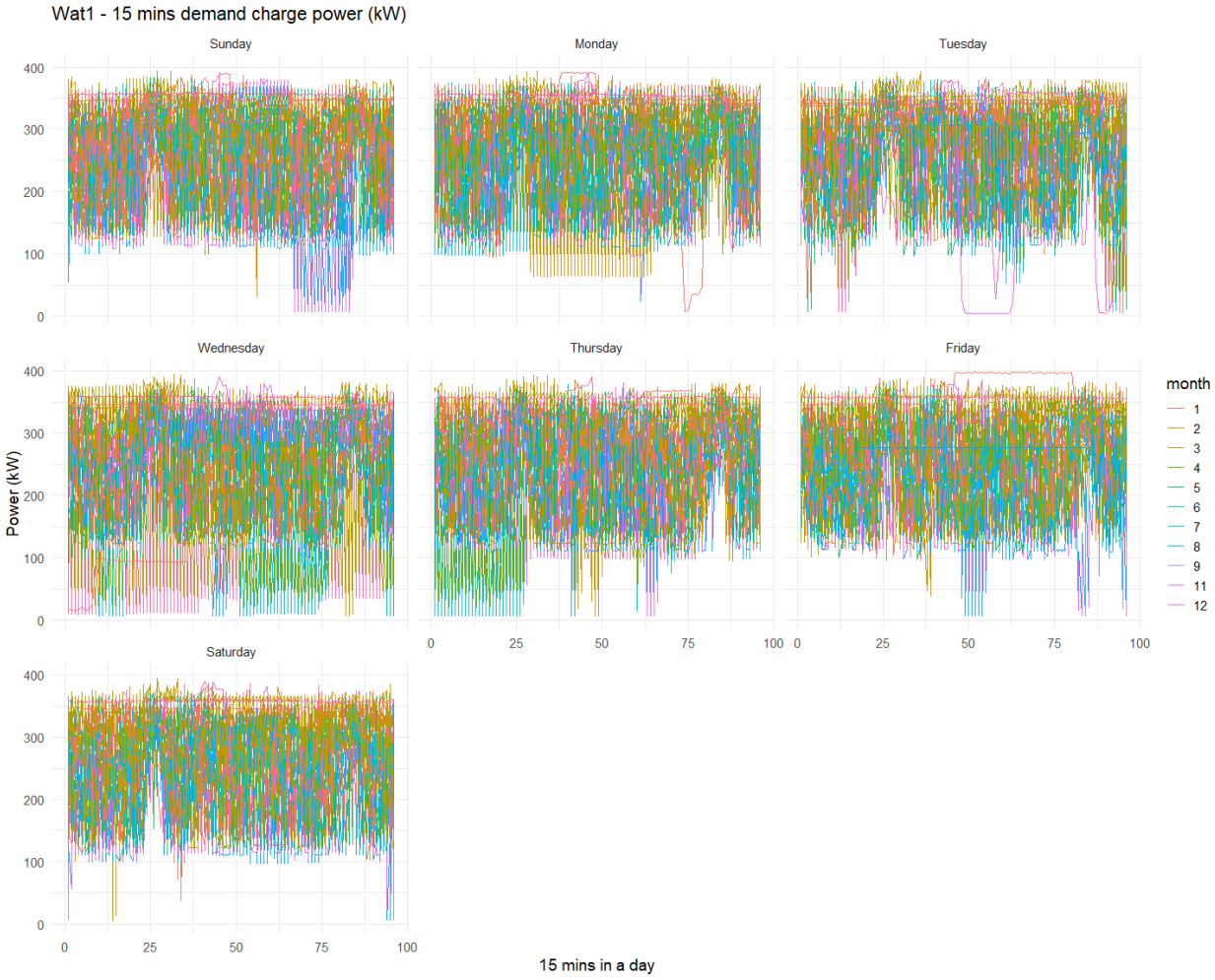
7.1.4 Power (kW) trend of a month for months (Jan. to Dec.) per year

7.2 Wat 1

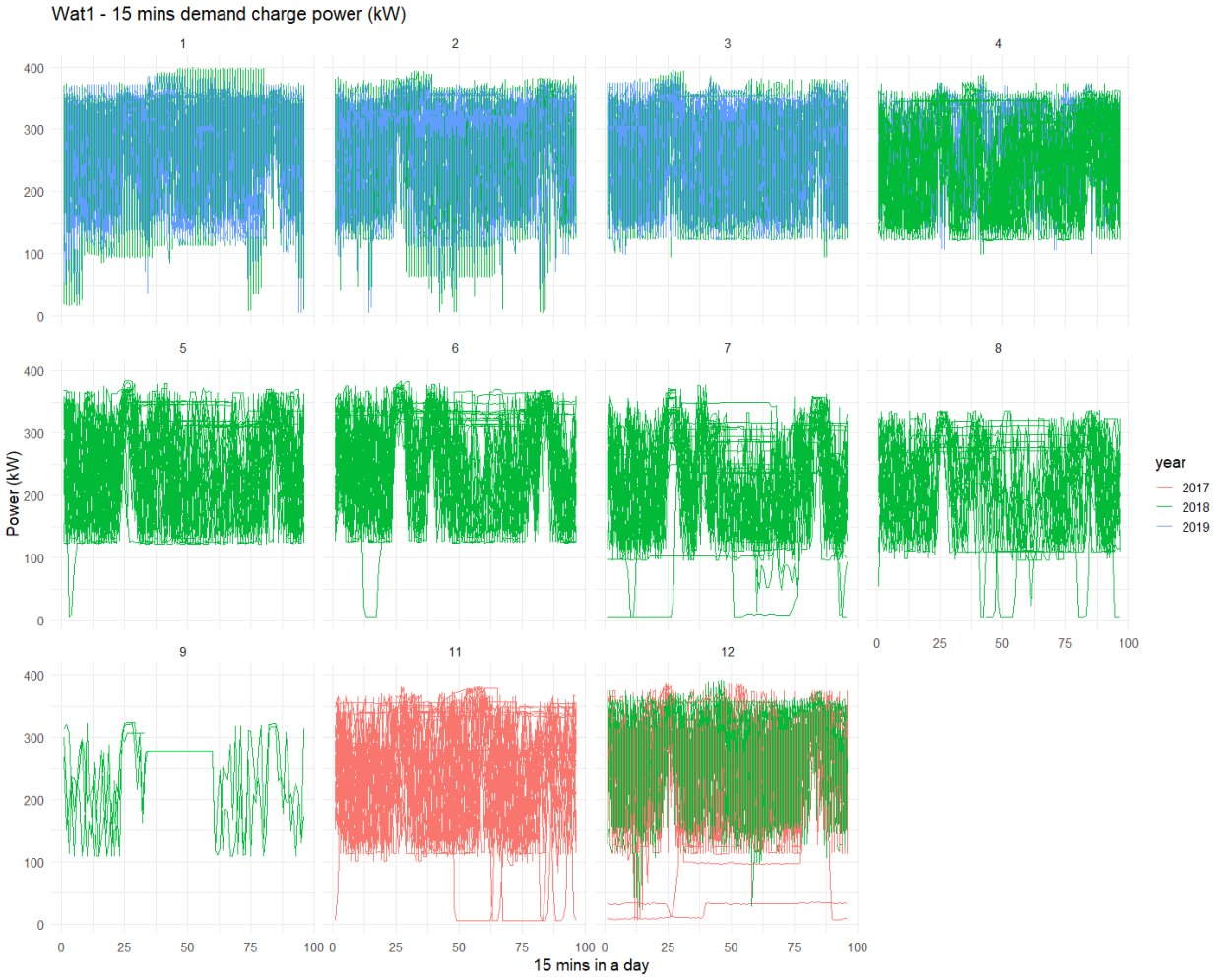
7.2.1 Power (kW) trend of a day for days (Day 1 to Day 31) per month



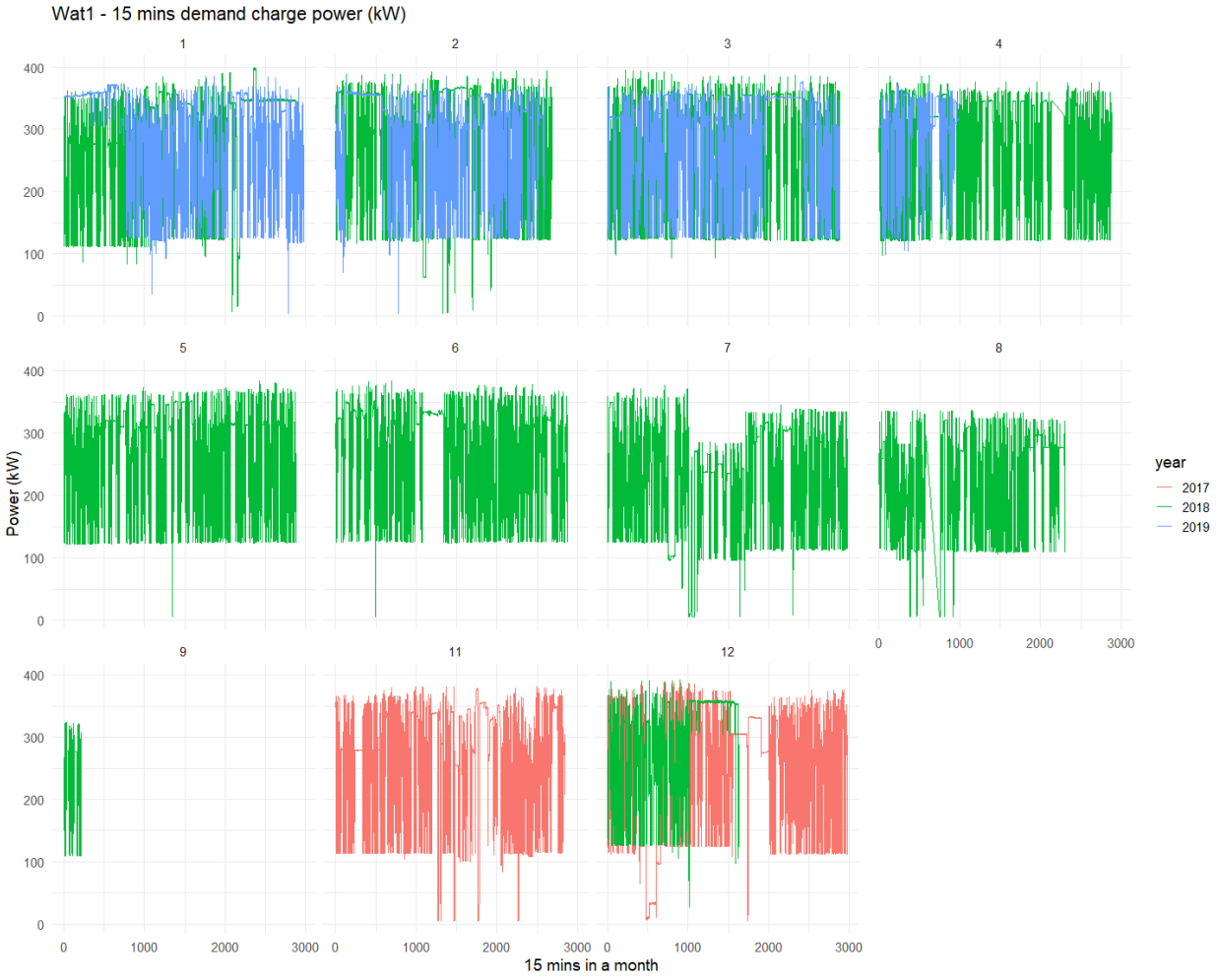
7.2.2 Power (kW) trend of a day for weekdays (Sunday to Saturday) per month



7.2.3 Power (kW) trend of a day for months (Jan. to Dec.) per year



7.2.4 Power (kW) trend of a month for months (Jan. to Dec.) per year

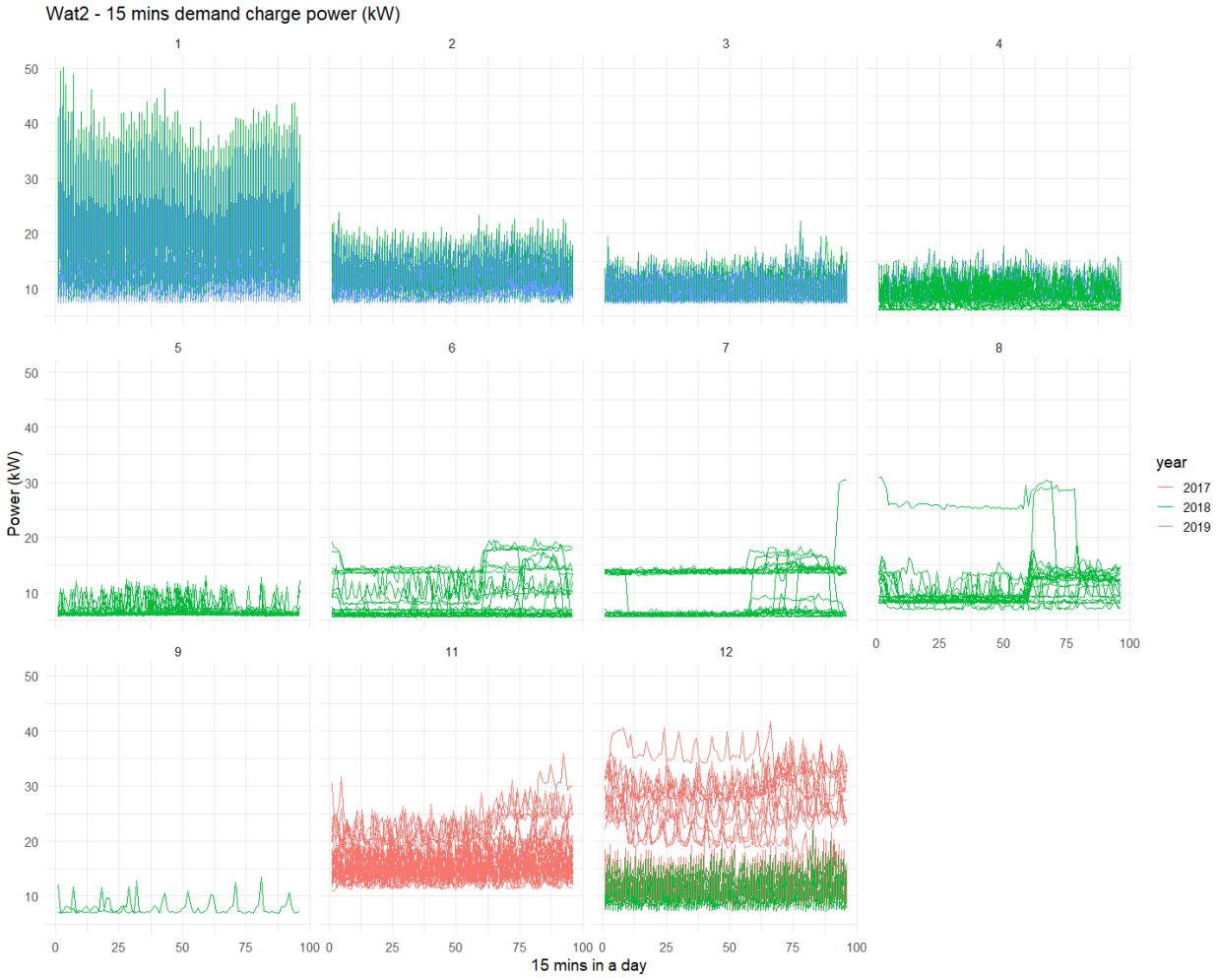


7.3 Wat 2

7.3.1 Power (kW) trend of a day for days (Day 1 to Day 31) per month

7.3.2 Power (kW) trend of a day for weekdays (Sunday to Saturday) per month

7.3.3 Power (kW) trend of a day for months (Jan. to Dec.) per year

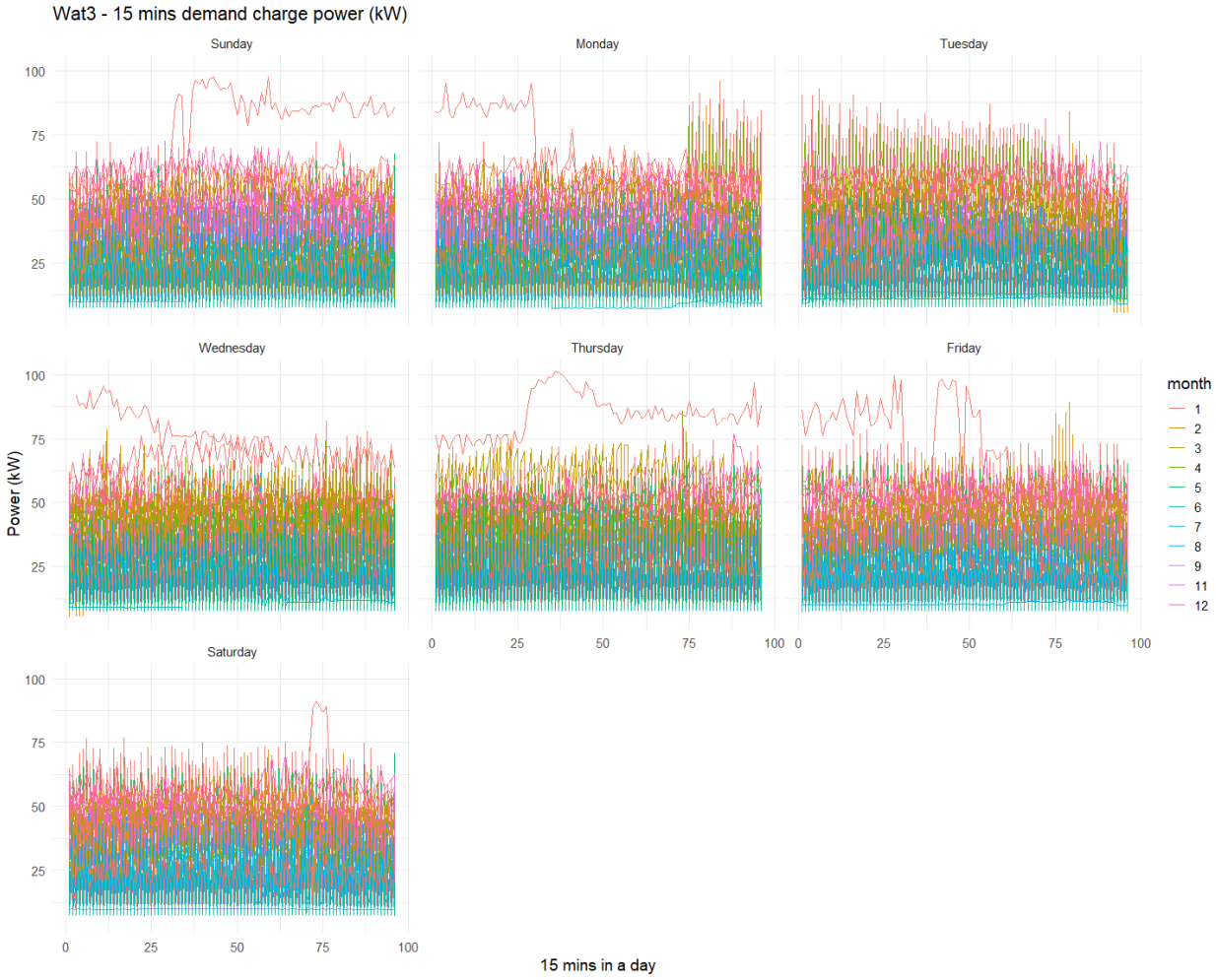


7.3.4 Power (kW) trend of a month for months (Jan. to Dec.) per year

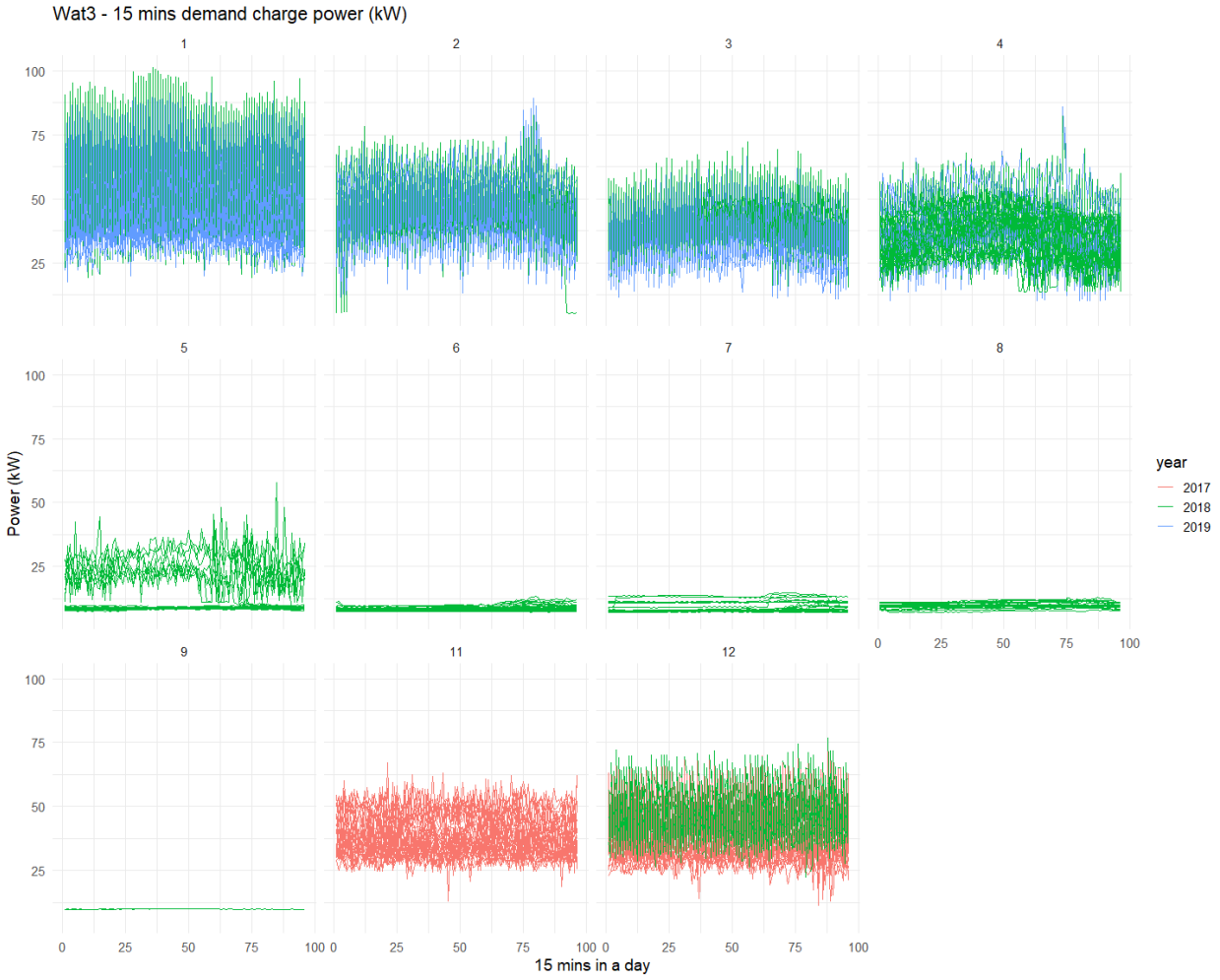
7.4 Wat 3

7.4.1 Power (kW) trend of a day for days (Day 1 to Day 31) per month

7.4.2 Power (kW) trend of a day for weekdays (Sunday to Saturday) per month



7.4.3 Power (kW) trend of a day for months (Jan. to Dec.) per year



7.4.4 Power (kW) trend of a month for months (Jan. to Dec.) per year

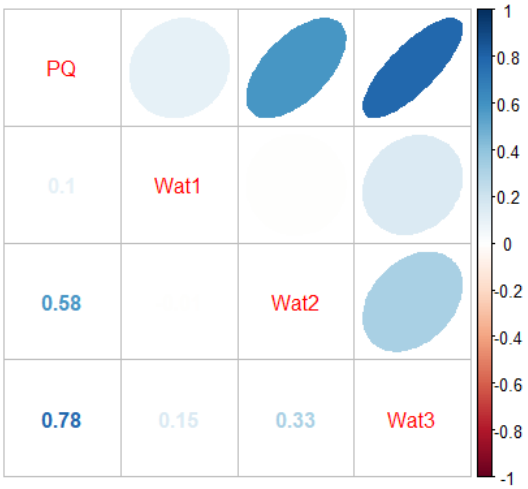
CHAPTER 8

Power (kW) correlation and forecast

These 4 meters have power (kW) correlated each other for the past 3 years. The correlations are various depending upon the comparisons. In general, power in *PQ* meter is mostly correlated with power in *Wat3*. This founding is interesting as the more correlated, the more dependency resulting in less effective to address load reduction of having a virtual meter, are expected. Using ARIMA, power trends of each meter were forecasted based on month, which is the billing cycle for demand charge. The power trends were plotted with maximum value of the peak power during the month because the peak power decides the billing cost. For the comparison, forecasts based on day, were also plotted. The range between the upper and lower bound of forecast shows narrower than the one based on month.

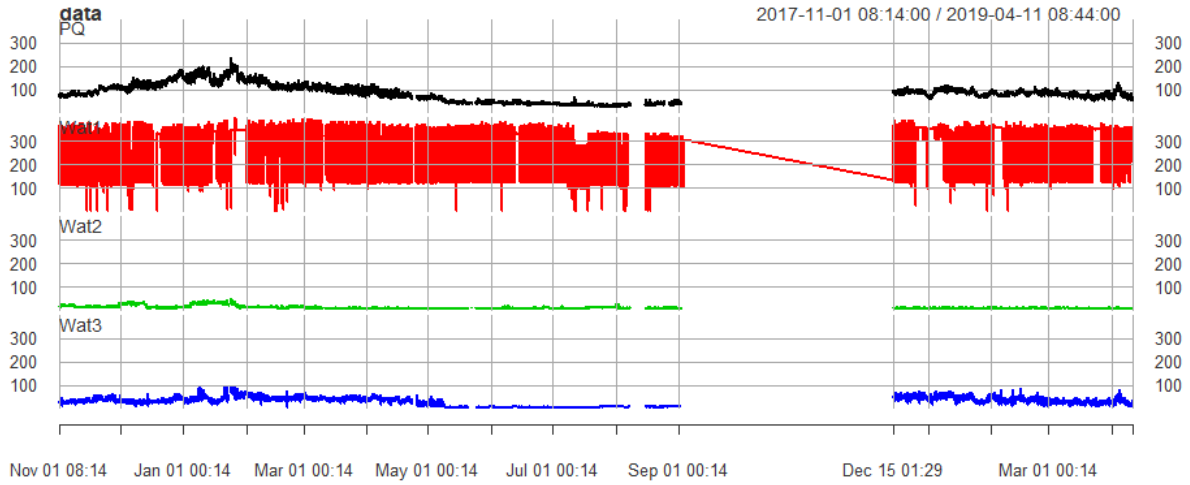
8.1 Power (kW) for demand charge correlation

These 4 meters have power (kW) correlated each other for the past 3 years. The correlations are various depending upon the comparisons. In general, power in `PQ` meter is mostly correlated with power in `Wat3`. This founding is interesting as the more correlated, the more dependency resulting in less effective to address load reduction of having a virtual meter, are expected.



8.2 Past 3 years power trends of each meter (Nov. 2017 to Apr. 2019)

Variances of power (kW) for each meter, verify the correlations found in the previous correlation plot.



A regression model for power of `PQ` shows `Wat3` is the most significant followed by `Wat2` resulting in 0.72 in adjusted R-squared. They are higly correlated.

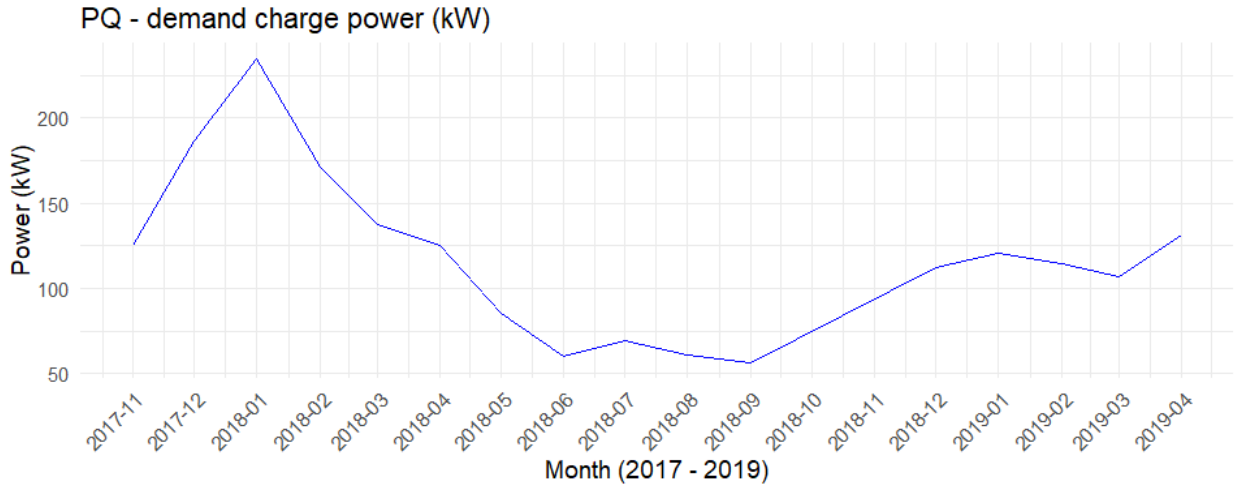
8.3 Forecast based on month

Using ARIMA, power trends of each meter were forecasted based on month, which is the billing cycle for demand charge. The power trends were plotted with maximum value of the peack power during the month because the peak power decides the billing cost.

8.3.1 PQ

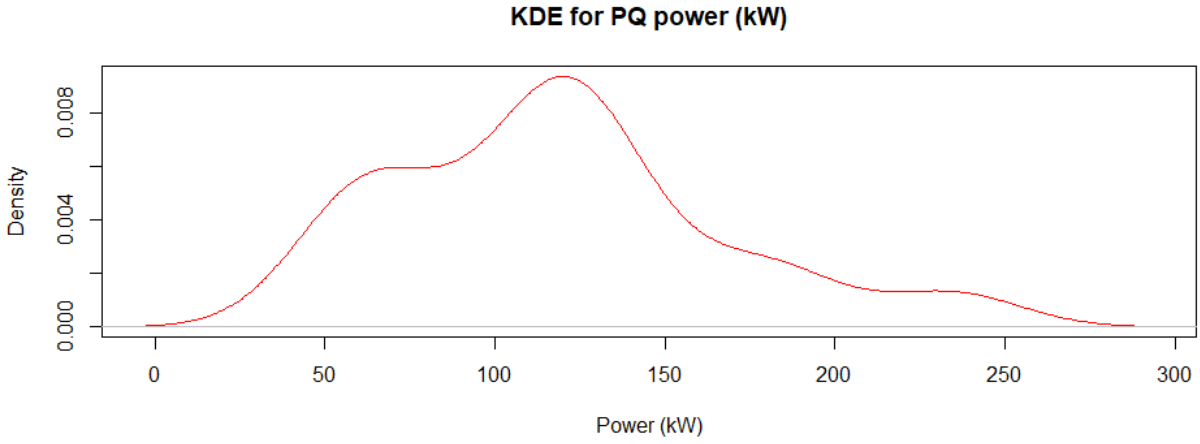
It was expected that power for PQ would be around 125 kW for the last 3 months and the real values for these months were 114, 107, and 131 kW respectively.

Monthly maximum peak power(kW) trend



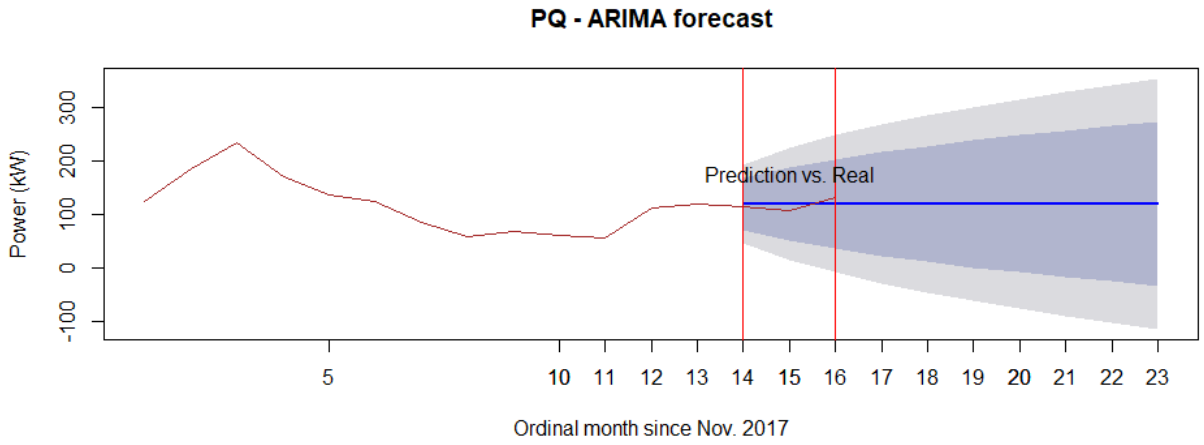
Density of maximum peak power (kW)

The density plot shows the distribution of the monthly peak power for each meter. For `PQ`, peak power around 125 kW is mostly prevalent, which means there is more probability that `PQ` peak power would be around 125 kW.



Prediction performance for the last 3 months

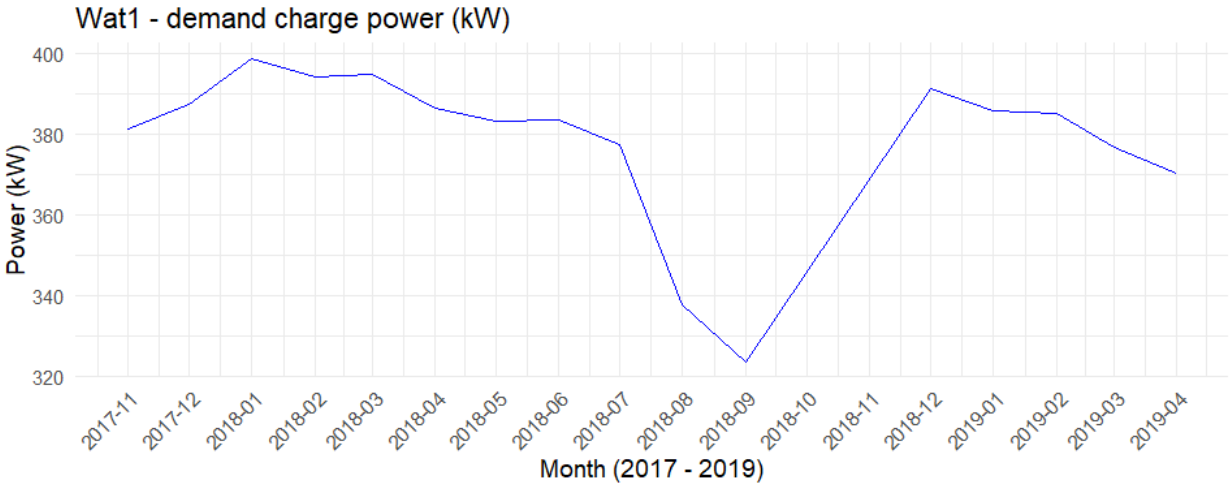
ARIMA predicted the peak power for the last 3 months, which is February, March, and April given the previous peak power data from November, 2017 to January, 2019. Since there are 2 months missing (November and December of 2018), the months used for the prediction were 13 months. Given the historical data of 13 months, the model predicts the coming 3 months and it is quite accurate as around 125 kW compared to the real values for these months, 114, 107, and 131 kW respectively.



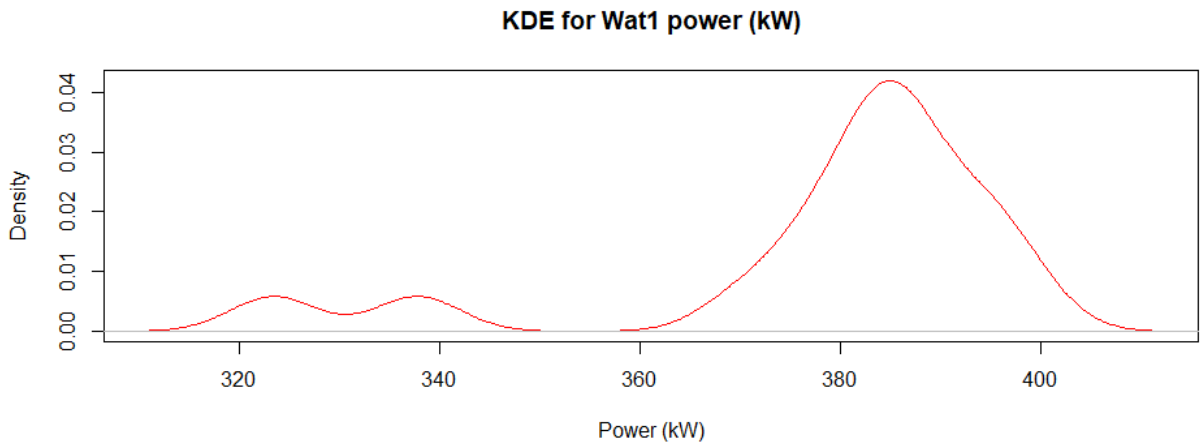
8.3.2 Wat1

It was expected that power for Wat1 would be around 372 kW for the last 3 months and the real values for these months were 385, 377, and 370 kW respectively.

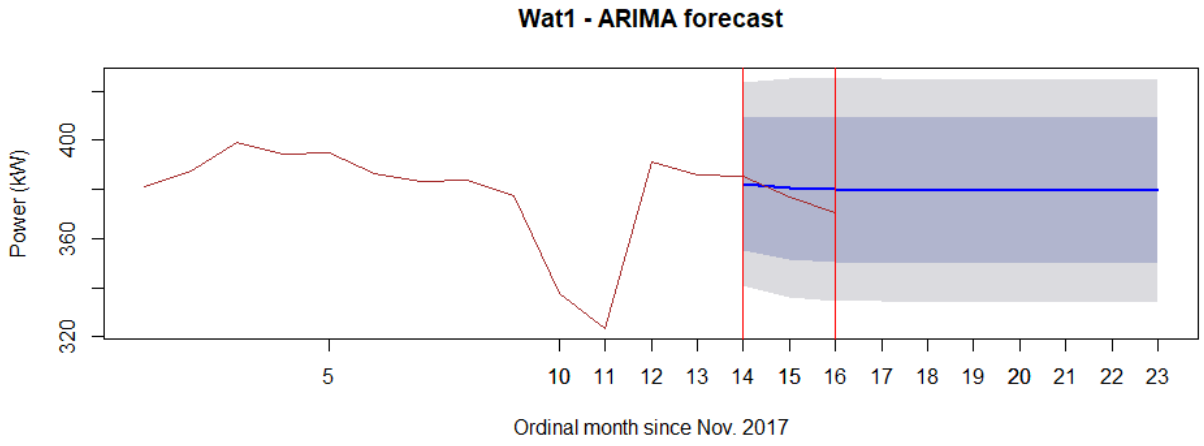
Monthly maximum peak power(kW) trend



Density of maximum peak power (kW)



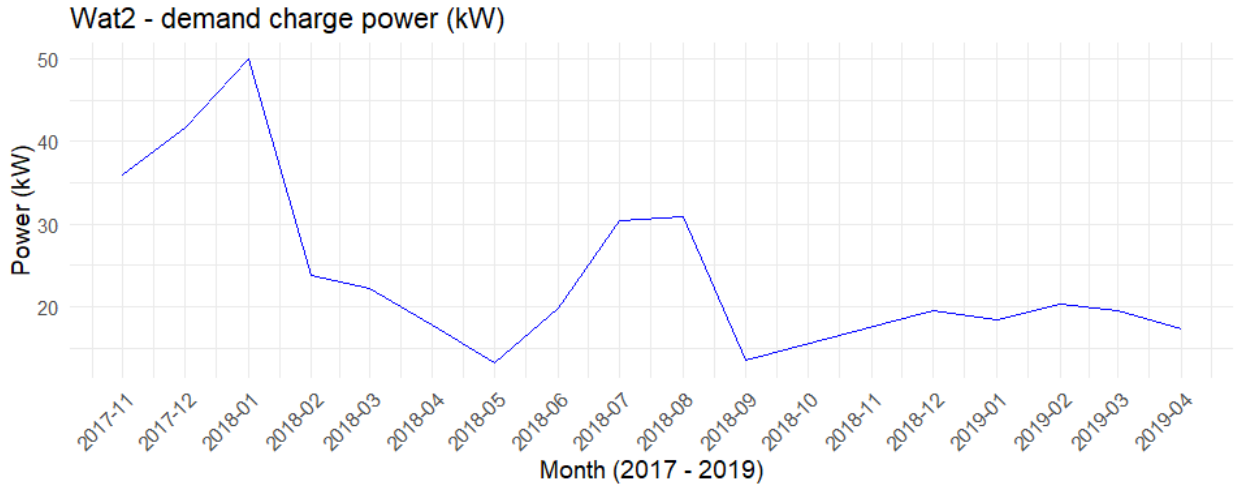
Prediction performance for the last 3 months



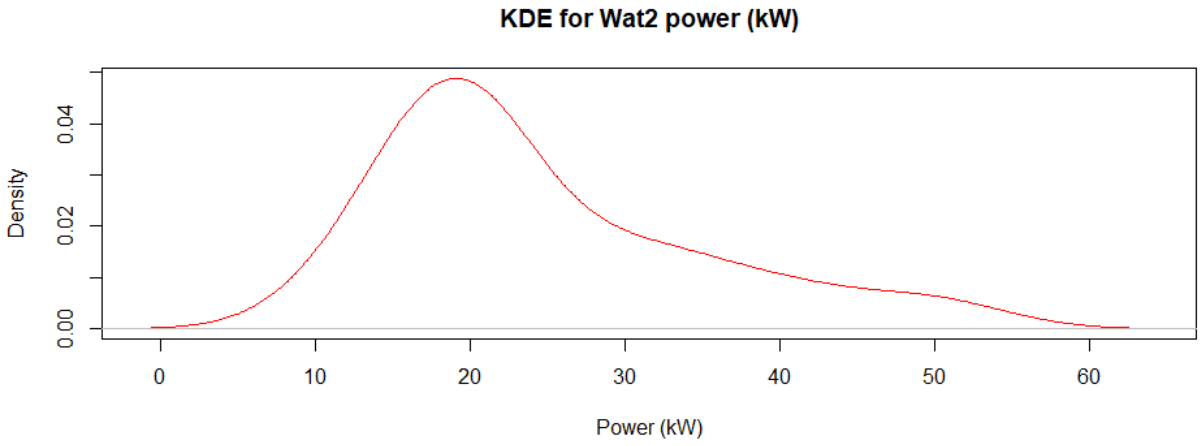
8.3.3 Wat2

It was expected that power for Wat2 would be around 18 kW for the last 3 months and the real values for these months were 20, 19, and 17 kW respectively.

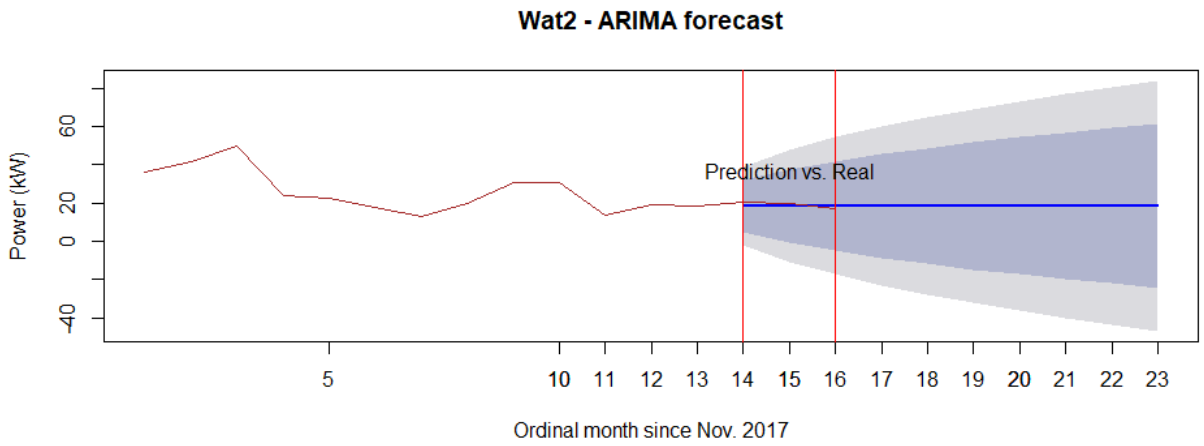
Monthly maximum peak power(kW) trend



Density of maximum peak power (kW)



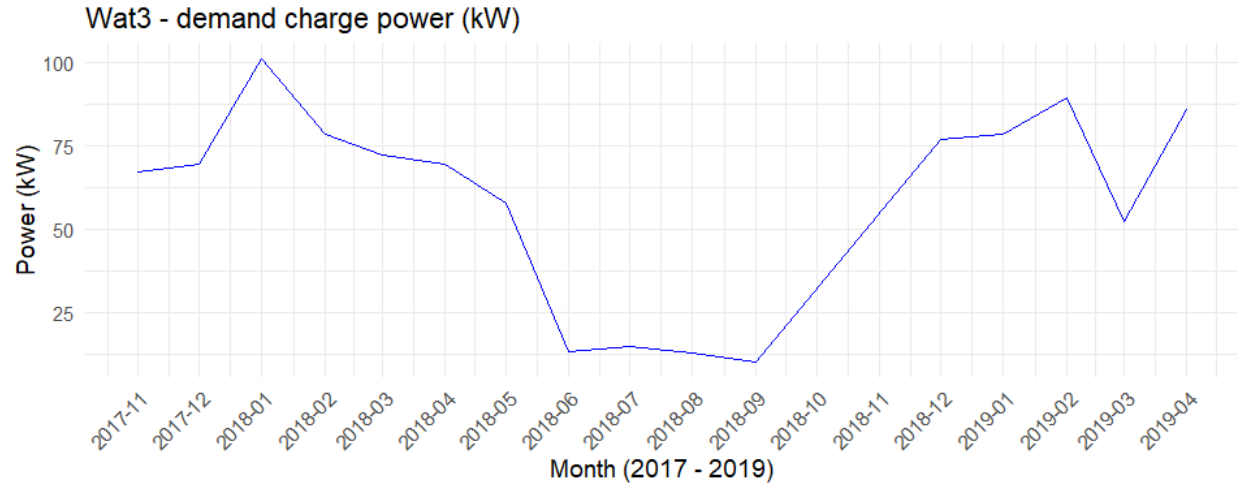
Prediction performance for the last 3 months



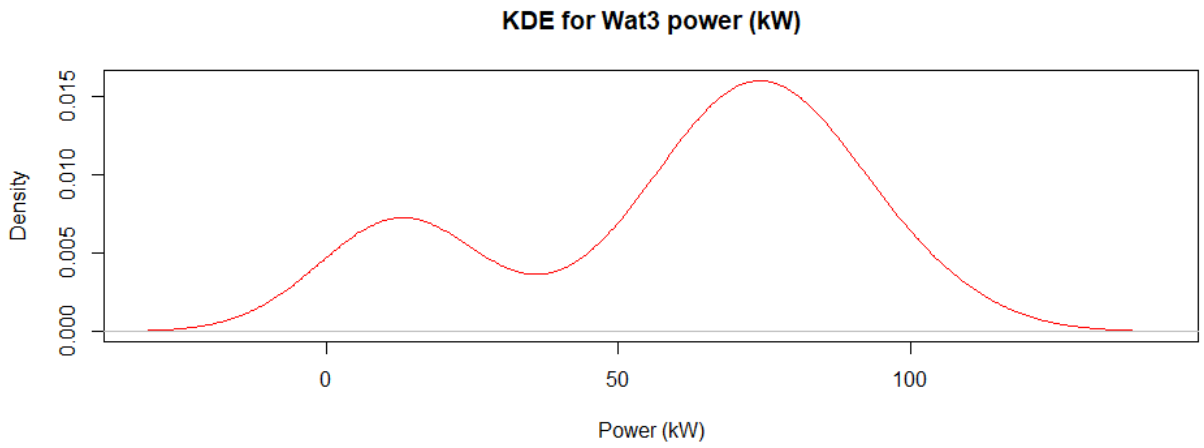
8.3.4 Wat3

It was expected that power for Wat3 would be around 68 kW for the last 3 months and the real values for these months were 89, 52, and 85 kW respectively.

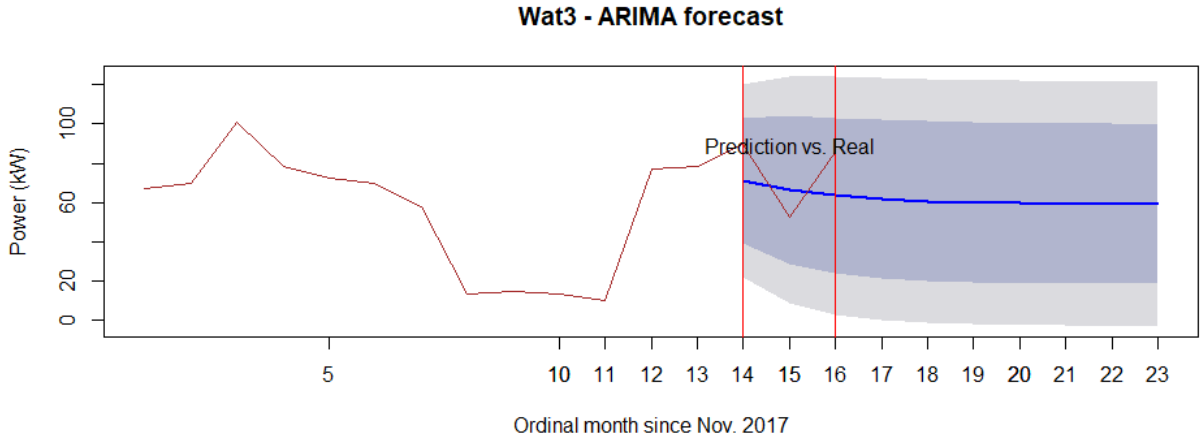
Monthly maximum peak power(kW) trend



Density of maximum peak power (kW)



Prediction performance for the last 3 months

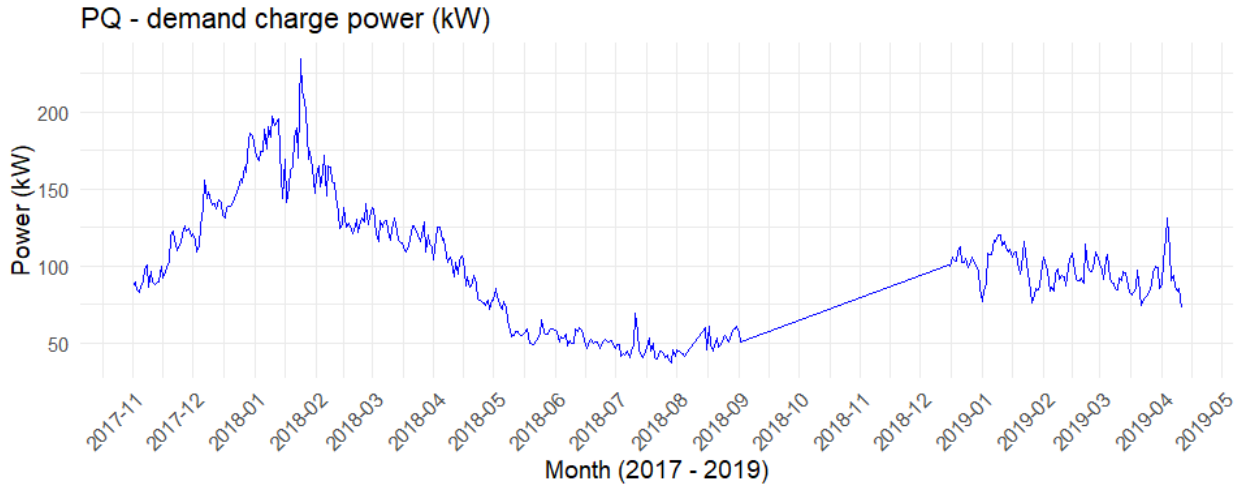


8.4 Forecast based on day

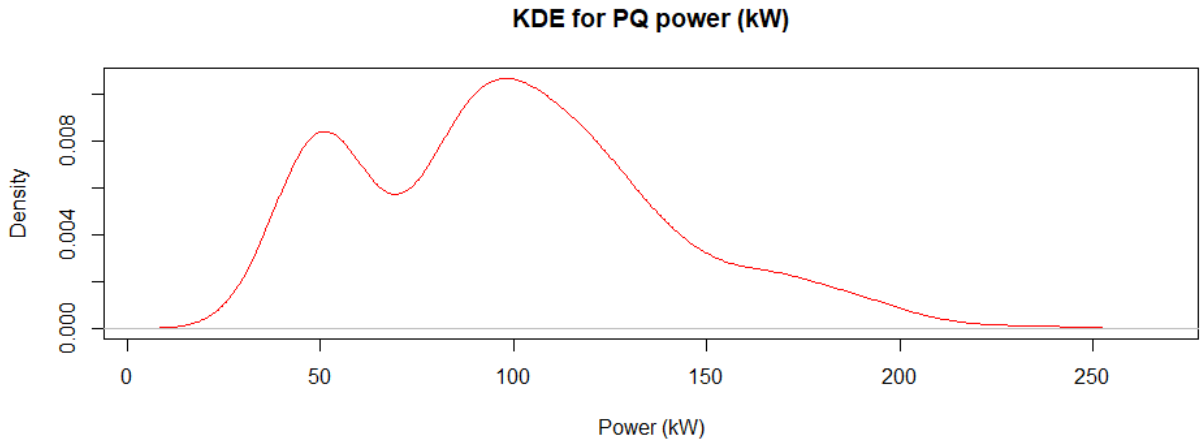
For the comparison, forecasts based on day, were also plotted. The range between the upper and lower bound of forecast shows narrower than the one based on month.

8.4.1 PQ

Daily maximum peak power(kW) trend

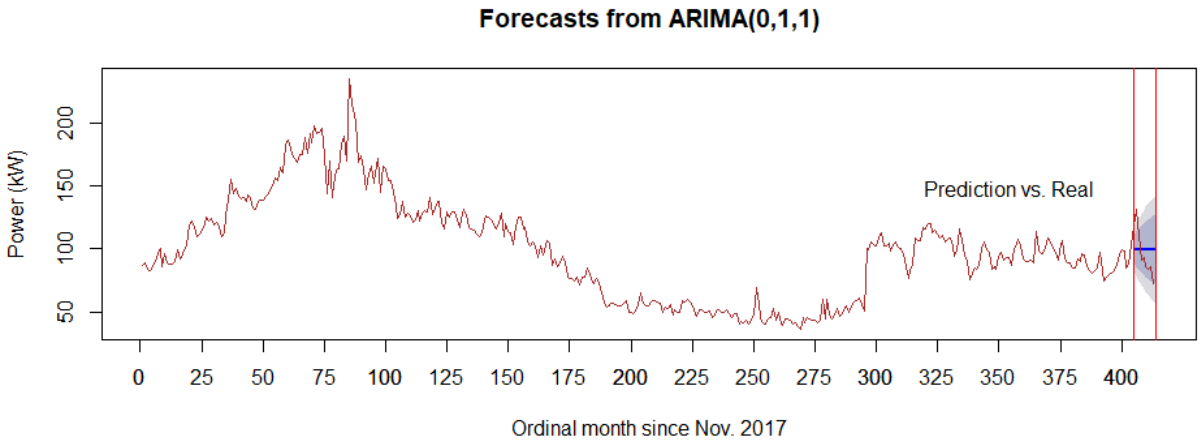


Density of maximum peak power (kW)



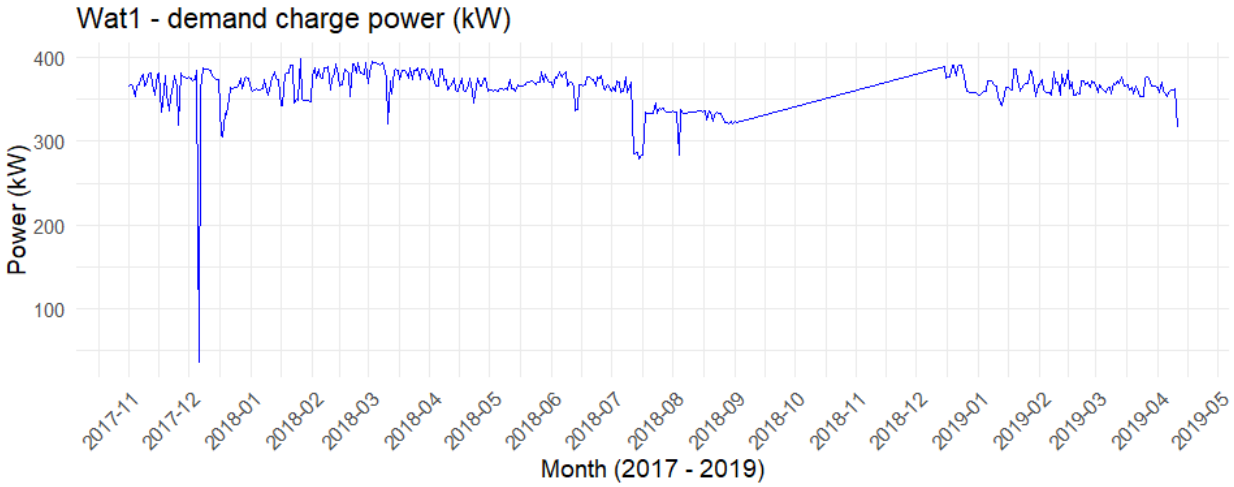
Prediction performance for the last 10 days

There are total 413 days previously available since November 2017 till April 2019. Given the 404 days, rest of 10 days were predicted from 405th day to 413th day as below in the figure.

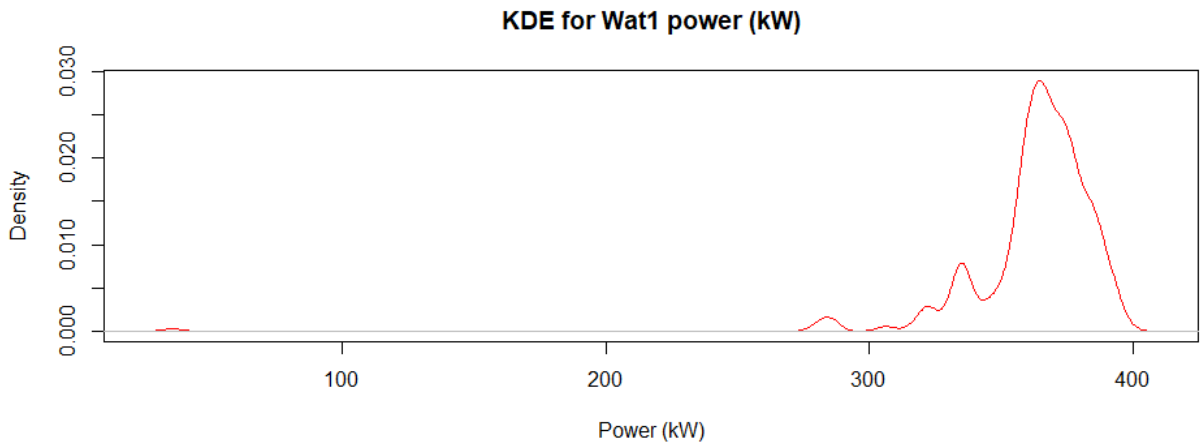


8.4.2 Wat1

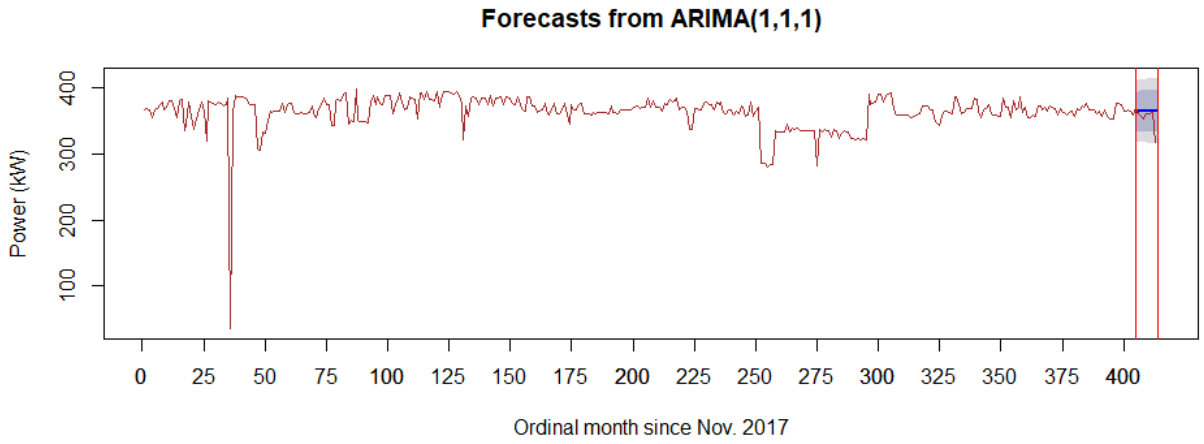
Daily maximum peak power(kW) trend



Density of maximum peak power (kW)

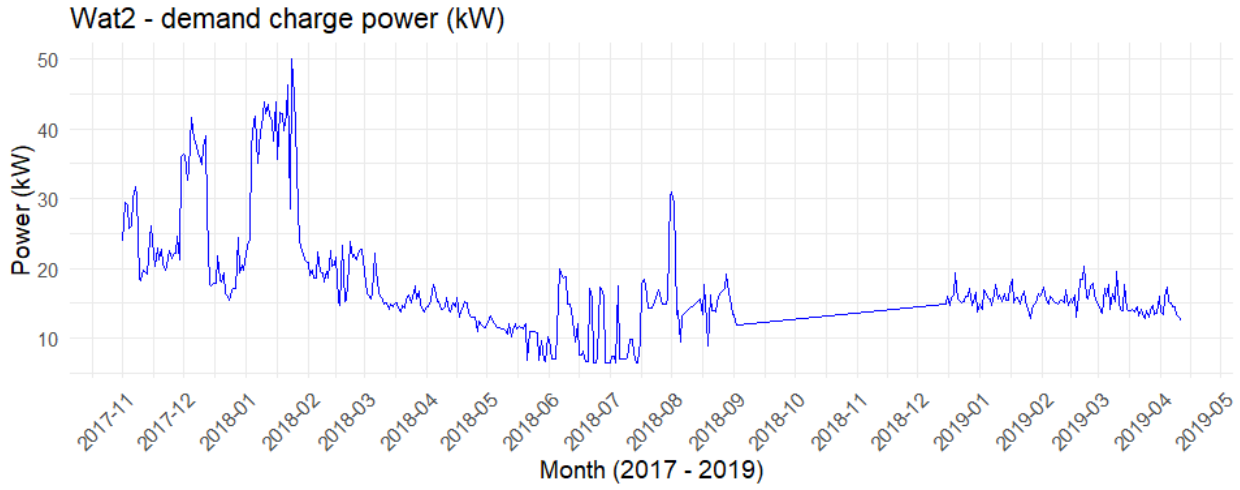


Prediction performance for the last 10 days

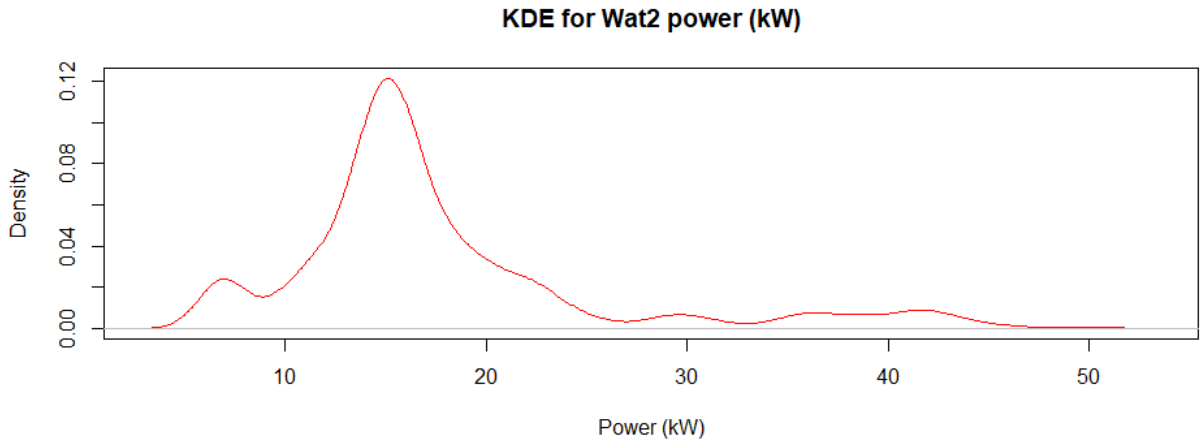


8.4.3 Wat2

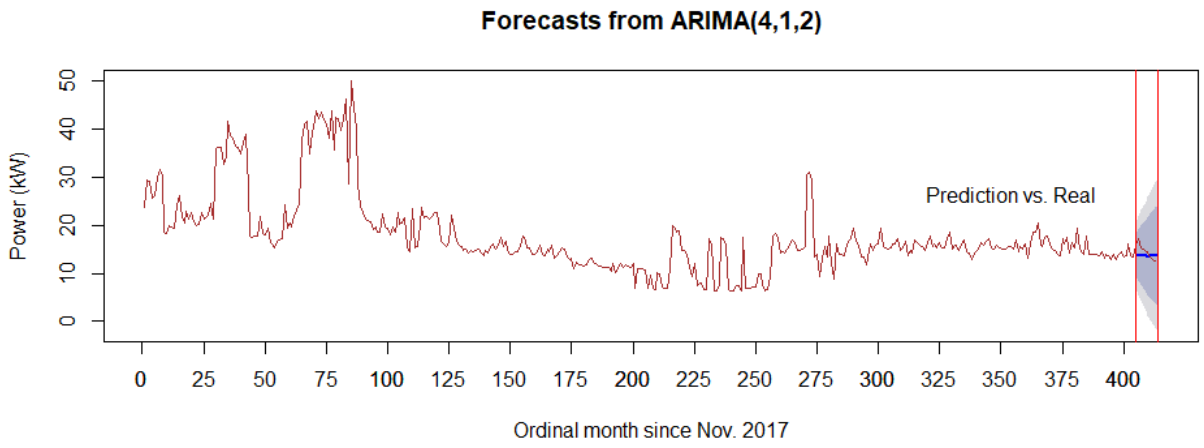
Daily maximum peak power(kW) trend



Density of maximum peak power (kW)

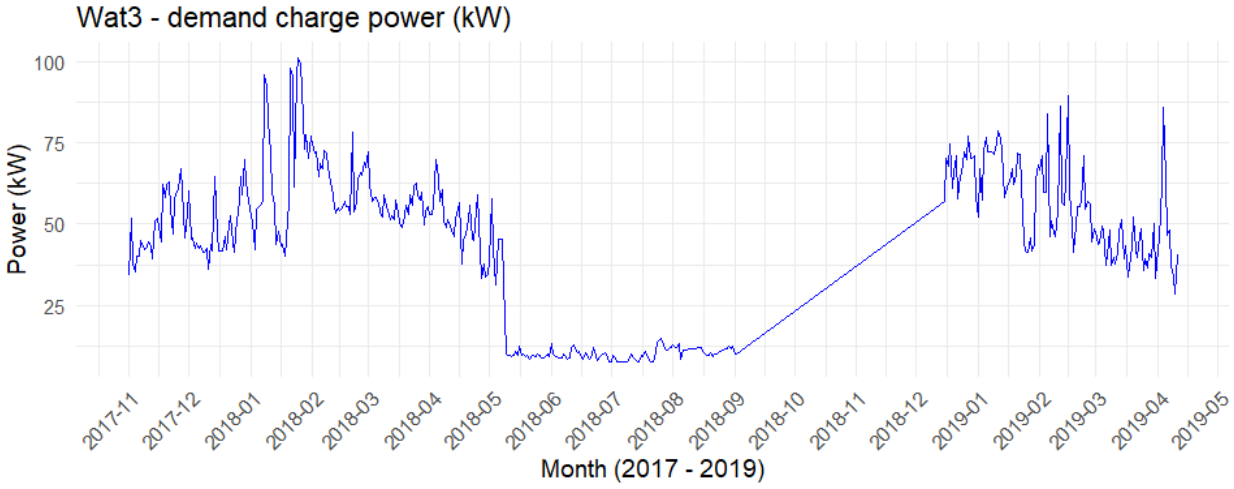


Prediction performance for the last 10 days

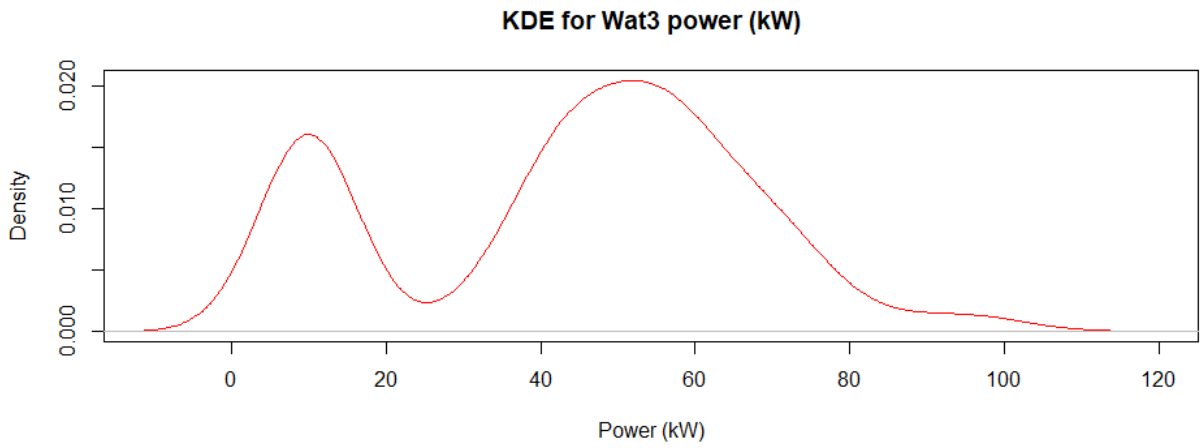


8.4.4 Wat3

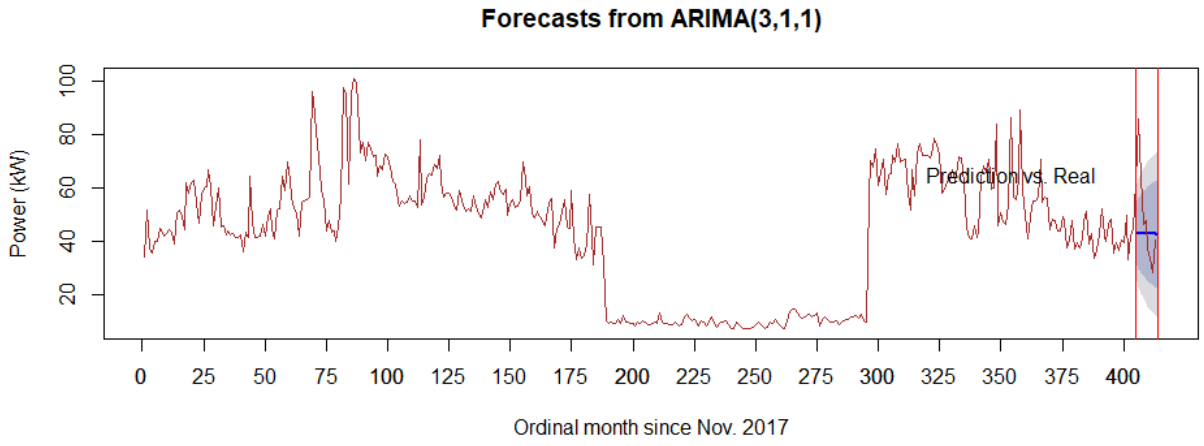
Daily maximum peak power(kW) trend



Density of maximum peak power (kW)



Prediction performance for the last 10 days



CHAPTER 9

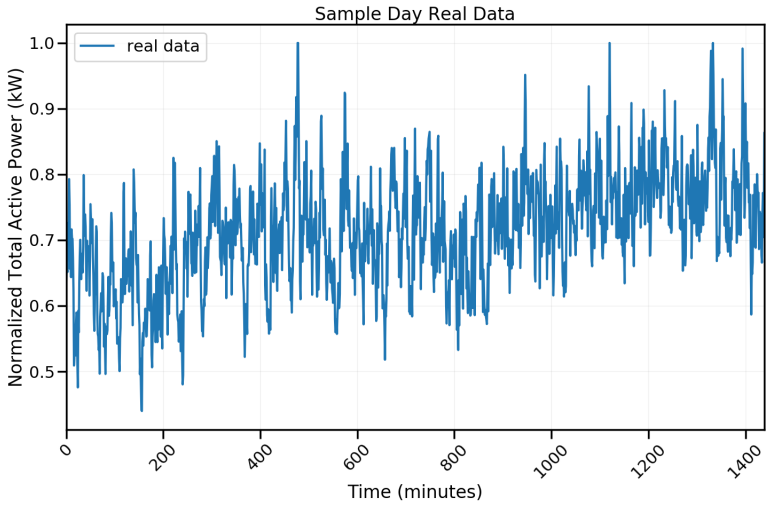
Load Synthesis

The benefits of having the data in an accessible form provided by the data pipeline is the ease of using the data for various analysis. Some potential analysis such as evaluating the energy storage potentials requires a large amount of data which is not available as the ACEP measurement data is ranges from November 2017 to March 2019.

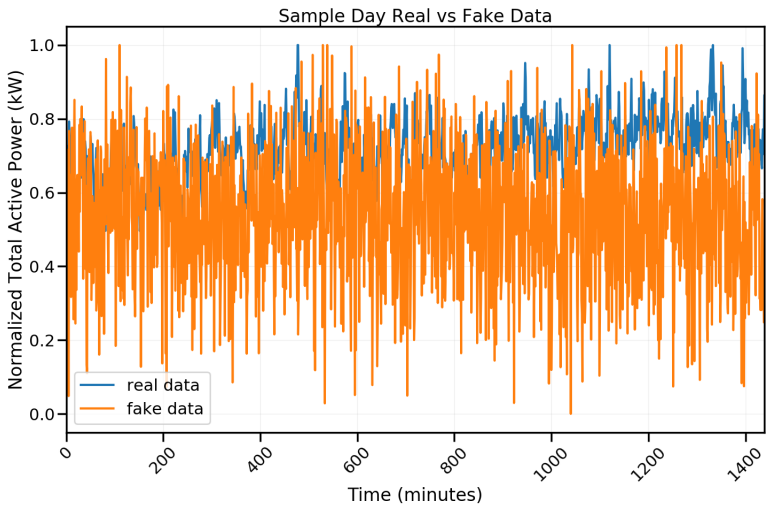
One way of mitigating this problem is to create synthetic load profiles, that is, load profiles which have similar structures to the actual data. A common way of generating synthetic data is using a method called generative adversarial network (GAN).

Unfortunately, the performance of GAN on the ACEP measurement data was quite poor. A possible reason for this could be due to the nature and structure of the measurement data as seen in the figure below. The data does not follow a particular time series trend and spikes at irregular intervals, which makes it difficult for the generator to produce a look-alike data set that can pass the discriminators test.

Sample Day Data.



Sample Day Data and Synthesized Data.



The code used in generating the results above can be found in load_synthesis.ipynb

CHAPTER 10

Virtual meter impact on demand charge

Having a virtual meter by aggregating all the power consumptions of all four meters, reduces the peak demand as shown in the Figure below. Every month has less peak demand by a virtual meter compared to summing up all the peak demand of the individual meters during a billing cycle. A virtual meter could save money by aggregating all the meters resulting in a payment once during a billing cycle as opposed to several billed payments, in this case four times. On the other hand, virtual meter may lead to higher rate per unit kW for the demand charger. For example, the utility has charged Poker flat $14.29 per kW for GS-2 service while GS-3 would involve $22.89 per kW. So it is necessary to cost-benefit analysis to find saving could happen by implementing a virtual meter. On the other hand, GS-3 service has higher utility charge of $0.0294 per kWh while GS-2 has $0.06256 per kWh. The monthly power consumption for each meter and the total are shown in the figure below.

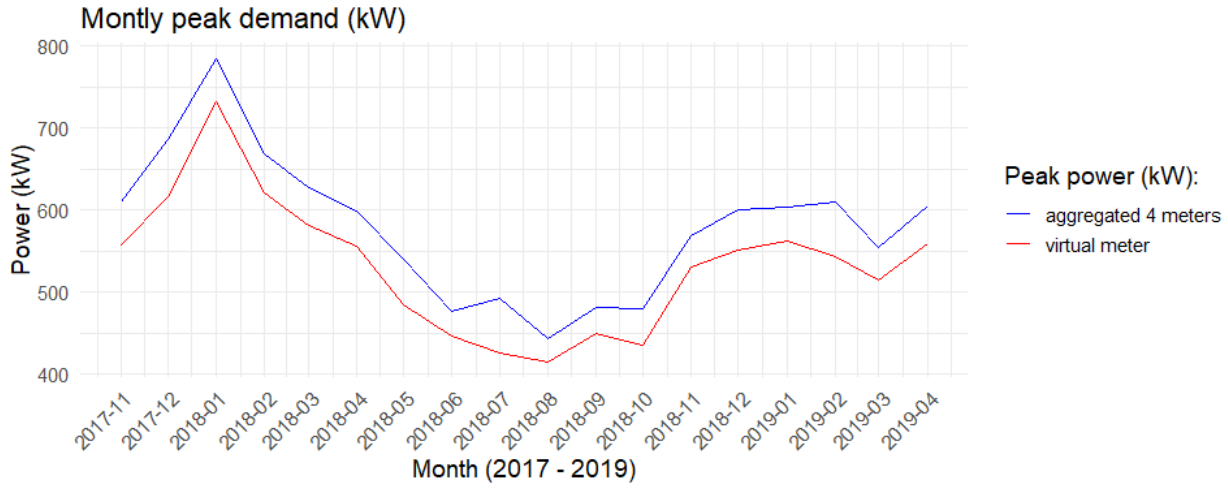
10.1 Total aggregated power (kW) for demand charge based on month

The aggregated total power (kW) on a monthly basis will show the trend of peak demand power over the past 3 years. This will help to analyze the effect of a virtual meter by comparing the separate billing of the four meters and the aggregated billing of the virtual meter.

Using ARIMA, the trend of aggregated meter powers were forecasted based on month, which is the billing cycle for demand charge. The power trends were plotted with maximum value of the peak power during the month because the peak power decides the billing cost.

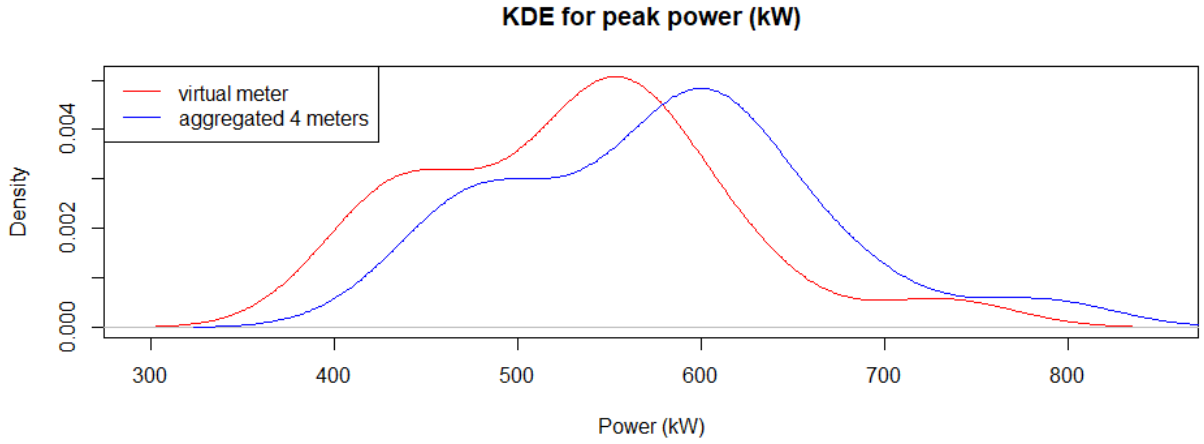
It turns out having a virtual meter by aggregating all the power consumptions of all four meters, reduces the peak demand as shown in the Figure below. Every month has less peak demand by a virtual meter compared to summing up all the peak demand of the individual meters during a billing cycle.

10.1.1 Monthly maximum peak power(kW) trend



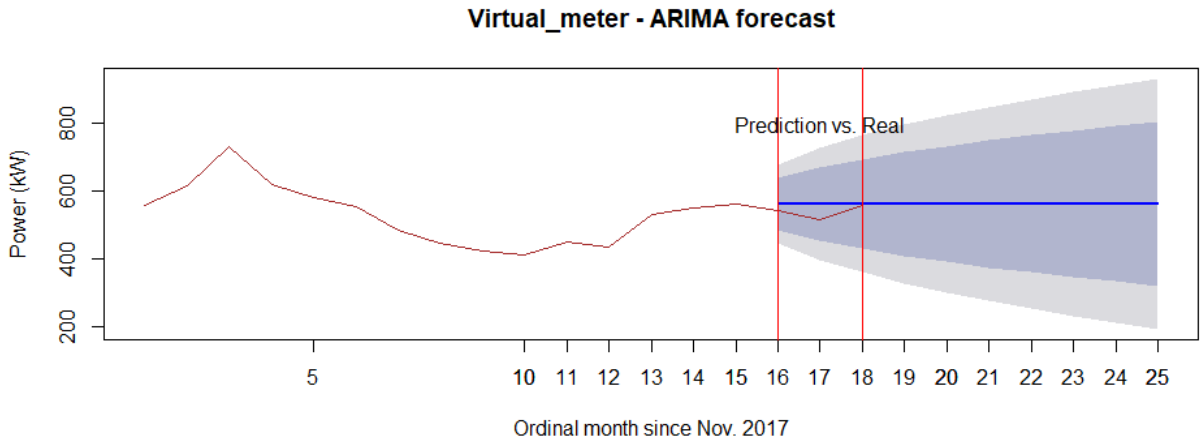
10.1.2 Density of maximum peak power (kW)

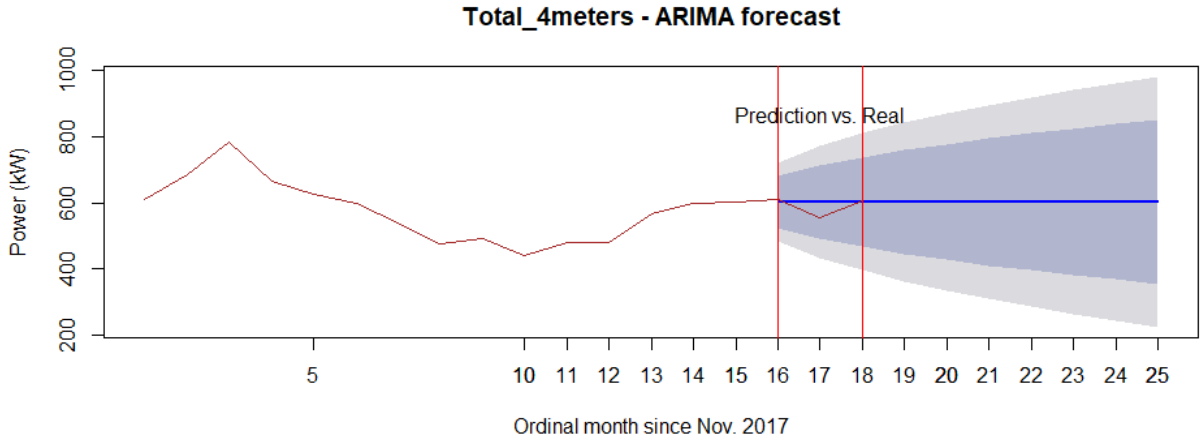
The figure below shows by having a virtual meter, the peak demand density becomes lower. It shows the distribution of peak power or probability that what value of peak power (kW) is highly expected for the cases of a virtual meter scenario and the total of the individual 4 meters. It shows that having a virtual meter reduced the probability of higher peak power across months.



10.1.3 Prediction performance for the last 3 months

There are slightly difference between monthly peak demand forecasts of a virtual meter and aggregated four meters.

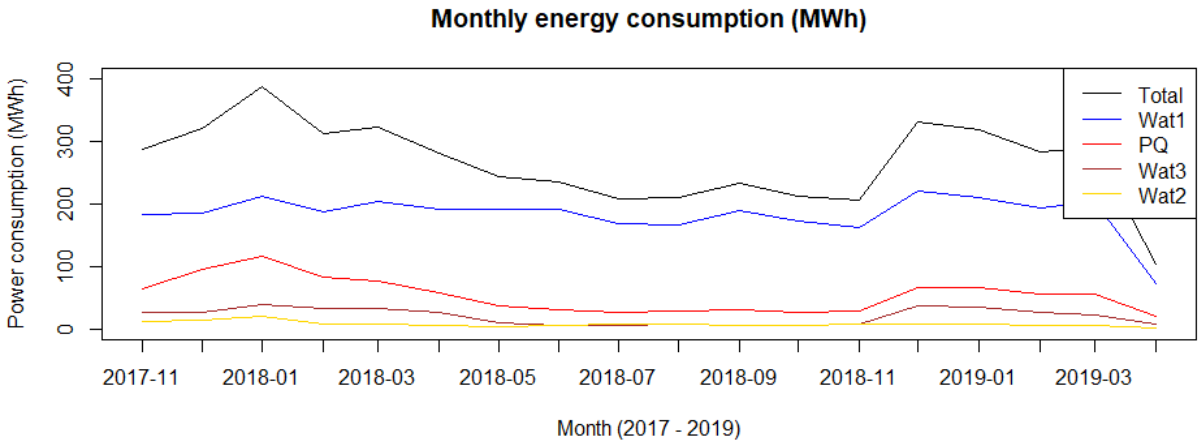




10.2 Benefit-cost analysis of involving a virtual meter

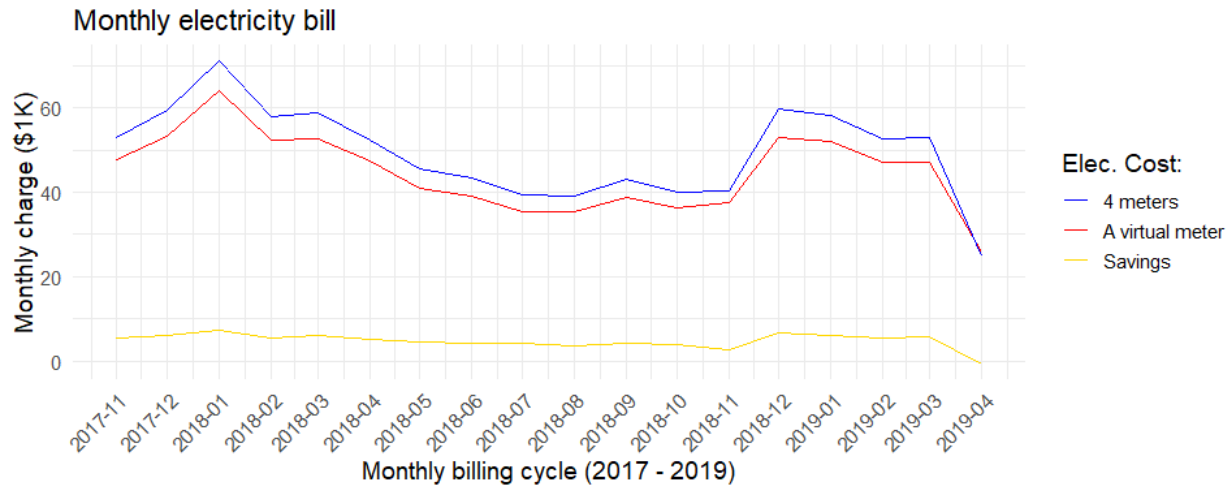
A virtual meter could save money by aggregating all the meters resulting in a payment once during a billing cycle as opposed to several billed payments, in this case four times. On the other hand, vitual meter may lead to higher rate per unit kW for the demand charge. For example, the utility has charged `Poker flat` $14.29 per kW for GS-2 service while GS-3 would involve $22.89 per kW. So it is necessary to perform a benefit-cost analysis to find if saving could happen by implementing a virtual meter.

On the other hand, GS-3 service has higher utility charge of $0.0294 per kWh while GS-2 has $0.06256 per kWh. The monthly power consumption for each meter and the total are shown in the figure below. Note that the energy consumption value is for each month in order since November 2017 to April 2019.

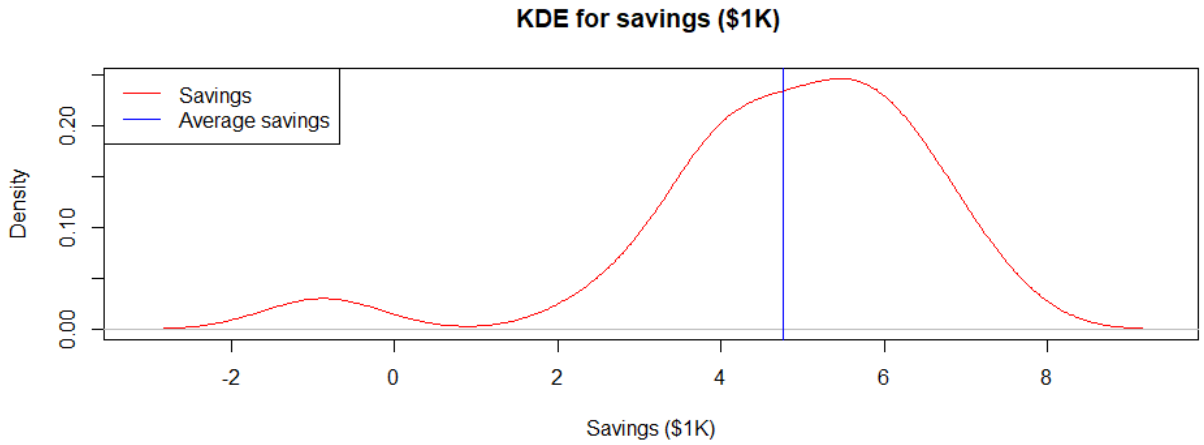


Adding customer charge and fuel & purchased power charge, which are fixed during a month, the total estimated electricity costs for the both cases (1) payments for the individual 4 meters, and (2) a payment for the virtual meter, were caculated.

It turns out aggregating all the meters by a virtual meter ends up with paying less with less peak power during a billing cycle as opposed to the aggregation of the peak power of the individual four meters. Note that a specific month has a negative saving meaning that the virtual meter option loses money. It is interesting to see that it would be more beneficial to have a virtual meter when higher energy consumption (kWh) is expected leading to more profitable option. The highest saving would be $7169.3 on 2018-01 and the lowest, $-877.89 on 2019-04, where the negative saving is due to the lower energy consumption on the month resulting from data missing.



The saving distribution is as below showing the most savings would occur between $4,000 to $7,000 a month.



The estimated average montly saving would be $5356.17 due to its skewness to left. So the conclusion is having a vurtaul meter is viable and saving money by reducing the billing.

- Note that the last month in the data, which is April of 2019, has only 10 days data available so the values are relatively lower than other months.
- There were missing months such as November and December of 2018, and missing days specially in September, 2018 so data filling technique was performed with the fact that this doesn't invole bias result.
- Nontheless, for the purpose of comparison of the both cases (a virtual meter, and 4 meters), missing or filling data won't be an issue as it applies the same to the both cases and still make the cases comparable.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search