Tutorial: Cleaning Smart Meter Data in R

*Chris Park**

18 August 2017

Contents

1 Prerequisites

This tutorial is in *Notebook* format, so that c code is interleaved with explanations (which hopefully makes it easier to understand code). Reasonable familiarity with R is assumed. If you're new to R, you might find sections 3.1 and 3.2 of the following tutorial useful. Note that comments are prepended by `##` and code output by `#`. This tutorial was created using R version 3.3.3, on a Windows machine. Here's some information about my current R session:

```
sessionInfo()
# R version 3.3.3 (2017-03-06)
# Platform: x86_64-w64-mingw32/x64 (64-bit)
# Running under: Windows >= 8 x64 (build 9200)
#
# locale:
# [1] LC_COLLATE=English_United States.1252  LC_CTYPE=English_United States.1252
# [3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
# [5] LC_TIME=English_United States.1252
#
# attached base packages:
# [1] stats     graphics  grDevices utils     datasets  methods   base
#
# other attached packages:
# [1] magrittr_1.5      ggplot2_2.2.1     data.table_1.10.4
#
# loaded via a namespace (and not attached):
#  [1] colorspace_1.3-2 scales_0.4.1     lazyeval_0.2.0   plyr_1.8.4       tools_3.3.3
#  [6] gtable_0.2.0     tibble_1.3.3     Rcpp_0.12.11     grid_3.3.3       knitr_1.16
# [11] rlang_0.1.1      munsell_0.4.3
```

*chris.park@protonmail.com

2 Background

This notebook is intended to serve as a template for assessing the quality of publicly available UK smart meter datasets in R. In this notebook, we use data from the Low Carbon London (LCL) project[1], which consists of electricity consumption readings from SMETS[2]-compliant Landis+Gyr E470 ZigBee Smart Meters[3] for a sample of 5,566 London households between 23 November 2011 and 28 Feburary 2014 (~ 827 days). All households were customers of EDF Energy.

The project employed a stratified sampling procedure (using ACORN groups) targeting specific demographic groups to attain a sample broadly representative of Greater London. Household recruitment was conducted on a voluntary, opt-in basis, and was spread out over as wide an area of London as possible. However, since the sample only contained customers from a single supplier (EDF Energy) and recruitment was voluntary, the sample is likely to be subject to self- selection bias, e.g. biased towards those who are more open to experimenting with new technology, EDF Energy's customer base may not be representative of Greater London, etc. Moreover, the recruitment process failed to account for household size; it is unlikely that a near-random spread of household size would have been achieved by stratifiying by ACORN group alone. This issues must be taken into account when drawing inferences from the data.

There were two groups of customers in the dataset:

- 1,123 customers subjected to Dynamic Time of use energy prices, where tariffs were given a day ahead via the Smart Meter In-Home Display or text message to mobile phone. Customers were issued high (67.29p/kWh), normal (11.76p/kWh), or low (3.99p/kWh) depending on the time of day.
- 4,443 customers were on a flat rate tariff of 14.23p/kWh.

The full data set can be downloaded as a `.csv` file from the London Data Store[4]. It is approximately 11GB in size, and contains 167,932,474 observations on the following 6 variables:

The full data set can be downloaded as a `.csv` file from the London Data Store[5]. It is approximately 11GB in size, and contains 167,932,474 observations on the following 6 variables:

- `LCLid`: Unique household identifier (integer)
- `stdorToU`: Household tariff scheme, either standard or dynamic time of use.
- `DateTime`: Date and time, in `DD/MM/YYYY HH:MM:SS` format.
- `KWH/hh (per half hour)`: half-hourly electricity consumption, in kWh per half hour.
- `Acorn`: CACI Acorn Group[6], a geo-demographic customer classification scheme that segments the UK population using demographic data and social factors.
- `Acorn_grouped`: Social class definitions derived from `Acorn`.

Note: ACORN stands for A Classification of Residential Neighbourhoods More information about the data can be found on Low Carbon London Report C5[7].

3 Data Import

Before we load the full data set, it is a good idea to have a look at the first few rows to get a sense of what we're dealing with, e.g. variable names and types. We use the sample data representing one household available from the London Data Store and read in the first few rows. The first 6 elements represent the variables and the next 6 represent values attached to the variables.

[1] http://innovation.ukpowernetworks.co.uk/innovation/en/Projects/tier-2-projects/Low-Carbon-London-(LCL)/

[2] https://www.gov.uk/government/consultations/smart-metering-equipment-technical-specifications-second-version

[3] http://www.landisgyr.co.uk/product/landisgyr-e470-zigbee-smart-electricity-meter/

[4] https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households

[5] https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households

[6] http://acorn.caci.co.uk/downloads/Acorn-User-guide.pdf

[7] https://innovation.ukpowernetworks.co.uk/innovation/en/Projects/tier-2-projects/Low-Carbon-London-(LCL)/Project-Documents/LCL%20Learning%20Report%20-%20C5%20-%20Accessibility%20and%20validity%20of%20smart%20meter%20data.pdf

```r
url = paste0("https://files.datapress.com/london/dataset/",
             "smartmeter-energy-use-data-in-london-households/",
             "UKPN-LCL-smartmeter-sample.csv")
scan(url, what = "", sep = ",", n = 12)
# Read 12 items
#  [1] "LCLid"                    "stdorToU"                 "DateTime"
#  [4] "KWH/hh (per half hour) " "Acorn"                    "Acorn_grouped"
#  [7] "MAC003718"                "Std"                      "17/10/2012 13:00:00"
# [10] "0.09"                     "ACORN-A"                  "Affluent"
```

In this notebook, we will use the `fread()` function from the `data.table` package to read in the full `.csv` data set. This is because it currently leads industry benchmarks and seems to be much faster than other functions for reading in `.csv` files, like `read.csv` from base R and `read_csv()` from the `readr` package[8]. We will also use other `data.table` functions for data manipulation and aggregation as they provide better performance than base R functions.

In general, it is preferable to specify the types of columns as an argument to a function importing the `.csv` file, as it means R has to do less work to figure out what the type is itself. For example, the `read.csv()` function in base R reads in all values within variables as character values then converts them as appropriate – which would amount to a lot of work for a large data set like ours. Now that we know the lay of the land (see output above), we can specify appropriate types for our 6 variables.

```r
## Vector of column/variable names. Observe that the names in the raw file
## are rather long and contain a mixture of camel case and underscores.
## We provide new names for brevity and consistency.
col_names = c("id", "tou", "datetime", "kwh_hh", "acorn", "acorn_grp")
col_classes = c(rep("character", 2), "POSIXct", rep("character", 3))

## Read in the full data set, saved as "lcl.csv" locally. Note that we skip
## the first row (which contains column labels) as we are specifying the
## column labels explicitly. It took me ~85 seconds to read in the full
## ~11GB file.
system.time({
  lcl = fread("lcl.csv", sep = ",", skip = 1,
              colClasses = col_classes,
              col.names = col_names)
})
# Read 167932474 rows and 6 (of 6) columns from 10.477 GB file in 00:01:24
#    user  system elapsed
#   82.01    3.07   85.31

## Note that I have a lot of RAM on my machine - roughly 32GB. On Windows,
## the function  "memory.limit()" returns the total RAM in MBs.
memory.limit() / 1024
# [1] 31.85742
```

Observe that the `kwh_hh` column was specified as a `character` (instead of `numeric`). This is because it contains the string `Null`, presumably as a placeholder for missing values. We can replace these strings by missing values (`NA`) once we read it in. This is a common issue in smart meter data, and each data set will use different placeholders, e.g. `""`, `NA`, `NULL`, `null`, `NULL`, `99`, etc. If you're from the SQL world, this is equivalent to first creating a table specifying `kwh_hh` as a character value, populating it with data, then updating the table after reading it in.

[8]https://csgillespie.github.io/efficientR/5-3-importing-data.html#fast-data-reading

4 Data Cleaning

```
## Let's have a look at the structure of the data.
str(lcl)
# Classes 'data.table' and 'data.frame':    167932474 obs. of  6 variables:
#  $ id       : chr  "MAC000002" "MAC000002" "MAC000002" "MAC000002" ...
#  $ tou      : chr  "Std" "Std" "Std" "Std" ...
#  $ datetime : chr  "2012-10-12 00:30:00.0000000" "2012-10-12 01:00:00.0000000"
#                    "2012-10-12 01:30:00.0000000" "2012-10-12 02:00:00.0000000" ...
#  $ kwh_hh   : chr  "0" "0" "0" "0" ...
#  $ acorn    : chr  "ACORN-A" "ACORN-A" "ACORN-A" "ACORN-A" ...
#  $ acorn_grp: chr  "Affluent" "Affluent" "Affluent" "Affluent" ...
#  - attr(*, ".internal.selfref")=<externalptr>
#  - attr(*, "index")= atomic
#   ..- attr(*, "__kwh_hh")= int  1 2 3 4 5 6 7 8 9 10 ...

## At the moment, both "datetime" and "kwh_hh" are character values, which is a
## problem. First, we change the "datetime" into a UTC date-time.
lcl[, datetime := as.POSIXct(datetime, tz = "UTC",
                             format = "%Y-%m-%d %H:%M:%S")]

## Let's check that things have worked out.
str(lcl$datetime)
# POSIXct[1:167932474], format: "2012-10-12 00:30:00" "2012-10-12 01:00:00"
#  "2012-10-12 01:30:00" ...

lcl[, .(datetime)]
#                      datetime
#         1: 2012-10-12 00:30:00
#         2: 2012-10-12 01:00:00
#         3: 2012-10-12 01:30:00
#         4: 2012-10-12 02:00:00
#         5: 2012-10-12 02:30:00
#        ---
# 167932470: 2012-06-21 05:30:00
# 167932471: 2012-06-21 06:00:00
# 167932472: 2012-06-21 06:30:00
# 167932473: 2012-06-21 07:00:00
# 167932474: 2012-12-19 12:32:41

## We can now do things like work out the start and end dates of the trial.
trial_range = range(lcl[, datetime])
trial_range
# [1] "2011-11-23 09:00:00 UTC" "2014-02-28 00:00:00 UTC"

## Or how long it ran for.
difftime(trial_range[2], trial_range[1])
# Time difference of 827.625 days
```

```
## Next, we look at the "kwh_hh" column. There are 5,560 observations with
## "Null" (string) values.
lcl[kwh_hh == "Null"]
#               id tou            datetime kwh_hh   acorn   acorn_grp
```

```r
#    1: MAC000002 Std 2012-12-19 12:37:27   Null ACORN-A    Affluent
#    2: MAC000003 Std 2012-12-19 12:37:26   Null ACORN-P   Adversity
#    3: MAC000004 Std 2012-12-19 12:32:40   Null ACORN-E    Affluent
#    4: MAC000006 Std 2012-12-19 12:37:26   Null ACORN-Q   Adversity
#    5: MAC000007 Std 2012-12-19 12:37:27   Null ACORN-H Comfortable
#   ---
# 5556: MAC005550 ToU 2012-12-18 15:16:35   Null ACORN-C    Affluent
# 5557: MAC005551 ToU 2012-12-18 15:16:35   Null ACORN-F Comfortable
# 5558: MAC005557 ToU 2012-12-19 12:32:41   Null ACORN-Q   Adversity
# 5559: MAC005564 ToU 2012-12-19 12:32:41   Null ACORN-Q   Adversity
# 5560: MAC005565 ToU 2012-12-19 12:32:41   Null ACORN-C    Affluent

## It looks like the datetime column has non-round times for rows that
## contain "Null" values. We expect readings to have taken place every
## 30 minutes, so want minute/second values that are either 00:00 or
## 30:00.
##
## Let's see if any of the "minute" values for the "Null" readings
## were 0 or 30.
lcl[kwh_hh == "Null", .(min = minute(datetime))][min %in% c(0, 30)]
# Empty data.table (0 rows) of 1 col: min

## Our suspicions seem to have been confirmed. Let's see how many
## households were affected by this.
lcl[kwh_hh == "Null", .N, id]
#              id N
#    1: MAC000002 1
#    2: MAC000003 1
#    3: MAC000004 1
#    4: MAC000006 1
#    5: MAC000007 1
#   ---
# 5556: MAC005550 1
# 5557: MAC005551 1
# 5558: MAC005557 1
# 5559: MAC005564 1
# 5560: MAC005565 1

## It seems that it affected 5,560 out of 5,566 households, and each
## household only had 1 "Null" reading. This could be indicative of some
## kind of region-wide interruption, e.g. power surge/fault.
lcl[kwh_hh == "Null", .N, id][N != 1]
# Empty data.table (0 rows) of 2 cols: id,N

## Let's drop the "Null" rows for now, since they only represent
## 5560/167932474 (or 0.0033%) of all observations, and coerce
## the "kwh_hh" values into numeric values. It might also be an
## interesting exercise to see if these values appeared at a
## particular time of day, but we won't go into that today.
lcl = lcl[kwh_hh != "Null"][, kwh_hh := as.numeric(kwh_hh)]
str(lcl$kwh_hh)
# num [1:167926914] 0 0 0 0 0 0 0 0 0 0 ...
```

Let's now move on to checking the validity of the ACORN codes. Valid household ACORN codes range from

ACORN-A (Lavish Lifestyles) to ACORN-Q (Difficult Circumstances), with the following high-level groups:

- Affluent: ACORN-A to ACORN-E
- Comfortable: ACORN-F to ACORN-J
- Adversity: ACORN-K to ACORN-Q

```
## Let's check if there are any invalid/unexpected ACORN codes or groups.
valid_code = sprintf("ACORN-%s", LETTERS[1:17])
valid_grp = c("Affluent", "Comfortable", "Adversity")

## It looks like there are 1,450,032 readings with in valid ACORN codes or
## groups.
lcl[!(acorn %in% valid_code)]
#                 id tou            datetime kwh_hh   acorn acorn_grp
#       1: MAC000023 Std 2011-12-07 10:30:00  0.540 ACORN-U   ACORN-U
#       2: MAC000023 Std 2011-12-07 11:00:00  0.863 ACORN-U   ACORN-U
#       3: MAC000023 Std 2011-12-07 11:30:00  0.475 ACORN-U   ACORN-U
#       4: MAC000023 Std 2011-12-07 12:00:00  0.495 ACORN-U   ACORN-U
#       5: MAC000023 Std 2011-12-07 12:30:00  0.341 ACORN-U   ACORN-U
#      ---
# 1450028: MAC005492 ToU 2014-02-27 22:30:00  0.122  ACORN-    ACORN-
# 1450029: MAC005492 ToU 2014-02-27 23:00:00  0.140  ACORN-    ACORN-
# 1450030: MAC005492 ToU 2014-02-27 23:30:00  0.192  ACORN-    ACORN-
# 1450031: MAC005492 ToU 2014-02-28 00:00:00  0.088  ACORN-    ACORN-
# 1450032: MAC005492 ToU 2014-02-28 00:00:00  0.088  ACORN-    ACORN-

lcl[!(acorn_grp %in% valid_grp)]
#                 id tou            datetime kwh_hh   acorn acorn_grp
#       1: MAC000023 Std 2011-12-07 10:30:00  0.540 ACORN-U   ACORN-U
#       2: MAC000023 Std 2011-12-07 11:00:00  0.863 ACORN-U   ACORN-U
#       3: MAC000023 Std 2011-12-07 11:30:00  0.475 ACORN-U   ACORN-U
#       4: MAC000023 Std 2011-12-07 12:00:00  0.495 ACORN-U   ACORN-U
#       5: MAC000023 Std 2011-12-07 12:30:00  0.341 ACORN-U   ACORN-U
#      ---
# 1450028: MAC005492 ToU 2014-02-27 22:30:00  0.122  ACORN-    ACORN-
# 1450029: MAC005492 ToU 2014-02-27 23:00:00  0.140  ACORN-    ACORN-
# 1450030: MAC005492 ToU 2014-02-27 23:30:00  0.192  ACORN-    ACORN-
# 1450031: MAC005492 ToU 2014-02-28 00:00:00  0.088  ACORN-    ACORN-
# 1450032: MAC005492 ToU 2014-02-28 00:00:00  0.088  ACORN-    ACORN-

## Are the invalid values unique to the columns, or are they identical?
invalid_code = unique(lcl[!(acorn %in% valid_code), acorn])
invalid_grp = unique(lcl[!(acorn_grp %in% valid_grp), acorn_grp])
identical(invalid_code, invalid_grp)
# [1] TRUE

## Now we know that we just have to deal with one of the two columns
## (i.e. "acorn" annd acorn_grp"). Let's see which values are
## invalid.
invalid_code
# [1] "ACORN-U" "ACORN-"

## How many households had these invalid ACORN codes?
lcl[acorn %in% invalid_code, .N, id]
#            id     N
```

```
#  1: MAC000023 39068
#  2: MAC000099 38831
#  3: MAC001256 31338
#  4: MAC001348 31193
#  5: MAC001600 30665
#  6: MAC001795 30236
#  7: MAC001997 29950
#  8: MAC002056 24688
#  9: MAC002087 29853
# 10: MAC002152 29661
# 11: MAC002485 28848
# 12: MAC002647 31960
# 13: MAC002720 31638
# 14: MAC003078 31360
# 15: MAC003163 27729
# 16: MAC003218  9920
# 17: MAC003317 24997
# 18: MAC003328 24971
# 19: MAC003407 24189
# 20: MAC003619 24186
# 21: MAC003780 23843
# 22: MAC003860 23568
# 23: MAC003884 23553
# 24: MAC003992 28884
# 25: MAC004010 28893
# 26: MAC004067 31186
# 27: MAC004069 11476
# 28: MAC004142 31059
# 29: MAC004215 31007
# 30: MAC004515 38450
# 31: MAC004570 38307
# 32: MAC004587 28651
# 33: MAC004649 15687
# 34: MAC004672 28548
# 35: MAC004788 33268
# 36: MAC004828 33183
# 37: MAC005036 36428
# 38: MAC005363 35372
# 39: MAC005424 32837
# 40: MAC001074 10786
# 41: MAC001147 25051
# 42: MAC001704 30617
# 43: MAC001706 30478
# 44: MAC001851 30094
# 45: MAC002774 30520
# 46: MAC003652 24179
# 47: MAC003916 23486
# 48: MAC003977 28797
# 49: MAC004323 21451
# 50: MAC004467 38596
# 51: MAC005492 26496
#             id     N
```

```
## It looks like most of the 51 households have over 20,000 readings with
## the invalid ACORN code. Let's see the reading counts for valid households.
lcl[acorn %in% valid_code, .N, id][, .(mean = mean(N), median = median(N))]
#        mean median
# 1: 30213.59  31135

## Since houses have around 30,000 readings each, it would seem that the
## majority of readings from those 51 households are invalid.  Let's drop
## these households (they are a tiny minority anyway, only representing
## 51/5561 (or ~0.92%) of all households).
invalid_id = lcl[acorn %in% invalid_code, .N, id][, id]
lcl = lcl[!(id %in% invalid_id)]

## Verify that there are no more invalid ACORN codes left.
lcl[acorn %in% invalid_code]
# Empty data.table (0 rows) of 6 cols: id,tou,datetime,kwh_hh,acorn,acorn_grp
```

Next, let's check the validity of `id` and `tou`.

```
## Are all tariff categories valid?
lcl[!(tou %in% c("Std", "ToU"))]
# Empty data.table (0 rows) of 6 cols: id,tou,datetime,kwh_hh,acorn,acorn_grp

## Are all id's in the expected form?
lcl[!(id %in% sprintf("MAC%06d", 1:5567))]
# Empty data.table (0 rows) of 6 cols: id,tou,datetime,kwh_hh,acorn,acorn_grp
```

What shall we do next? I guess we could investigate zero readings. A lot of readings seem to equal zero - is this something that warrants further investigation?

```
## Before we go any further: are there any negative values?
lcl[kwh_hh < 0]
# Empty data.table (0 rows) of 6 cols: id,tou,datetime,kwh_hh,acorn,acorn_grp

## How many readings equal 0?
lcl[kwh_hh == 0]
#                  id tou            datetime kwh_hh   acorn   acorn_grp
#       1: MAC000002 Std 2012-10-12 00:30:00      0 ACORN-A    Affluent
#       2: MAC000002 Std 2012-10-12 01:00:00      0 ACORN-A    Affluent
#       3: MAC000002 Std 2012-10-12 01:30:00      0 ACORN-A    Affluent
#       4: MAC000002 Std 2012-10-12 02:00:00      0 ACORN-A    Affluent
#       5: MAC000002 Std 2012-10-12 02:30:00      0 ACORN-A    Affluent
#      ---
# 1915457: MAC005537 ToU 2013-12-28 14:00:00      0 ACORN-F Comfortable
# 1915458: MAC005537 ToU 2013-12-28 14:30:00      0 ACORN-F Comfortable
# 1915459: MAC005551 ToU 2013-04-18 18:30:00      0 ACORN-F Comfortable
# 1915460: MAC005551 ToU 2013-04-18 19:00:00      0 ACORN-F Comfortable
# 1915461: MAC005564 ToU 2013-08-06 08:00:00      0 ACORN-Q   Adversity

## Approx. 1.15% of observations seem to equal 0.
nrow(lcl[kwh_hh == 0]) / nrow(lcl) * 100
# [1] 1.150587

## How many households were affected?
lcl[kwh_hh == 0, .N, id][order(-N)]
```

```r
#              id     N
#    1: MAC000197 39350
#    2: MAC000037 38771
#    3: MAC004067 31186
#    4: MAC001309 30509
#    5: MAC002594 28660
#   ---
# 1643: MAC005449     1
# 1644: MAC005477     1
# 1645: MAC005482     1
# 1646: MAC005516     1
# 1647: MAC005564     1

## Only 1.2% of the observations were affected, but nearly 30% of households
## seem to have zero readings.
summary(lcl[kwh_hh == 0, .N, id][order(-N)])
#      id                  N
# Length:1628        Min.   :    1.0
# Class :character   1st Qu.:    2.0
# Mode  :character   Median :   32.0
#                    Mean   : 1176.6
#                    3rd Qu.:  795.8
#                    Max.   :39350.0

## It would be useful to be able to look at the distribution of these values
## by time of day, e.g. 48 half-hour periods in the day. We write a simple
## function that returns which of the 48 half-hour blocks a POSIX time
## corresponds to.
match_hh = function(x) {
  stopifnot(inherits(x, "POSIXt"),
            hour(x) %in% 0:23,
            minute(x) %in% c(0, 30))
  x_char = paste(sprintf("%02d", hour(x)),
                 sprintf("%02d", minute(x)),
                 "00", sep = ":")
  half_hr = paste(rep(sprintf("%02d", 0:23), each = 2),
                  c("00", "30"), "00", sep = ":")
  data.table::chmatch(x_char, c(half_hr[-1], half_hr[1]))
}

## Example: 00:00 corresponds to hh-period 48, 00:30 to 1, etc.
match_hh(as.POSIXct("2012-08-12 00:00:00 UTC"))
match_hh(as.POSIXct("2012-08-12 00:30:00 UTC"))
match_hh(as.POSIXct("2012-08-12 23:30:00 UTC"))

## Let's look at the distribution of zero values by time of day.
## It looks like they are distributed in a fairly uniform manner
## throughout the day, (instead of just at night, for example).
## This is something an energy reseracher could look into in a bit
## more detail. Is it because people are using alternative sources
## of energy, perhaps?
lcl[kwh_hh == 0, .N, .(half_hr = match_hh(datetime))][order(half_hr)]
#     half_hr     N
```

```
#  1:       1 41587
#  2:       2 44043
#  3:       3 45785
#  4:       4 47433
#  5:       5 47685
#  6:       6 48898
#  7:       7 49065
#  8:       8 49514
#  9:       9 49338
# 10:      10 49232
# 11:      11 47962
# 12:      12 46951
# 13:      13 44979
# 14:      14 43143
# 15:      15 41732
# 16:      16 40623
# 17:      17 40241
# 18:      18 41362
# 19:      19 41809
# 20:      20 43254
# 21:      21 43825
# 22:      22 44209
# 23:      23 44739
# 24:      24 44246
# 25:      25 43354
# 26:      26 43248
# 27:      27 42991
# 28:      28 42395
# 29:      29 41754
# 30:      30 40868
# 31:      31 39404
# 32:      32 37520
# 33:      33 35595
# 34:      34 33690
# 35:      35 31841
# 36:      36 30452
# 37:      37 29446
# 38:      38 28830
# 39:      39 28051
# 40:      40 27786
# 41:      41 27690
# 42:      42 27893
# 43:      43 28237
# 44:      44 29681
# 45:      45 31283
# 46:      46 34037
# 47:      47 37016
# 48:      48 40744
#     half_hr     N

## We could the same across all non-zero readings:
lcl[kwh_hh != 0, .N, .(half_hr = match_hh(datetime))][order(half_hr)]
#     half_hr       N
```

```
#  1:       1 3421323
#  2:       2 3418928
#  3:       3 3417270
#  4:       4 3415554
#  5:       5 3415524
#  6:       6 3414317
#  7:       7 3414224
#  8:       8 3413492
#  9:       9 3413838
# 10:      10 3413755
# 11:      11 3415197
# 12:      12 3416218
# 13:      13 3418215
# 14:      14 3420016
# 15:      15 3421376
# 16:      16 3422573
# 17:      17 3423429
# 18:      18 3422573
# 19:      19 3422444
# 20:      20 3421615
# 21:      21 3421509
# 22:      22 3421479
# 23:      23 3421491
# 24:      24 3422500
# 25:      25 3423385
# 26:      26 3424153
# 27:      27 3424576
# 28:      28 3425266
# 29:      29 3426074
# 30:      30 3427015
# 31:      31 3428900
# 32:      32 3430293
# 33:      33 3432573
# 34:      34 3434691
# 35:      35 3436249
# 36:      36 3437977
# 37:      37 3438866
# 38:      38 3438985
# 39:      39 3440159
# 40:      40 3440460
# 41:      41 3440296
# 42:      42 3440229
# 43:      43 3439730
# 44:      44 3438413
# 45:      45 3436825
# 46:      46 3434104
# 47:      47 3431358
# 48:      48 3541984
#     half_hr       N

## Or across ALL readings. It looks like we have roughly 3.4 million
## observations for each half-hour period in a day.
lcl[, .N, .(half_hr = match_hh(datetime))][order(half_hr)]
```

```
#     half_hr       N
#  1:       1 3462910
#  2:       2 3462971
#  3:       3 3463055
#  4:       4 3462987
#  5:       5 3463209
#  6:       6 3463215
#  7:       7 3463289
#  8:       8 3463006
#  9:       9 3463176
# 10:      10 3462987
# 11:      11 3463159
# 12:      12 3463169
# 13:      13 3463194
# 14:      14 3463159
# 15:      15 3463108
# 16:      16 3463196
# 17:      17 3463670
# 18:      18 3463935
# 19:      19 3464253
# 20:      20 3464869
# 21:      21 3465334
# 22:      22 3465688
# 23:      23 3466230
# 24:      24 3466746
# 25:      25 3466739
# 26:      26 3467401
# 27:      27 3467567
# 28:      28 3467661
# 29:      29 3467828
# 30:      30 3467883
# 31:      31 3468304
# 32:      32 3467813
# 33:      33 3468168
# 34:      34 3468381
# 35:      35 3468090
# 36:      36 3468429
# 37:      37 3468312
# 38:      38 3467815
# 39:      39 3468210
# 40:      40 3468246
# 41:      41 3467986
# 42:      42 3468122
# 43:      43 3467967
# 44:      44 3468094
# 45:      45 3468108
# 46:      46 3468141
# 47:      47 3468374
# 48:      48 3582728
#     half_hr       N

## We can also do things like look at differences between groups.
lcl[, .(median = mean(kwh_hh)), by = .(acorn, match_hh(datetime))][order(acorn)]
```

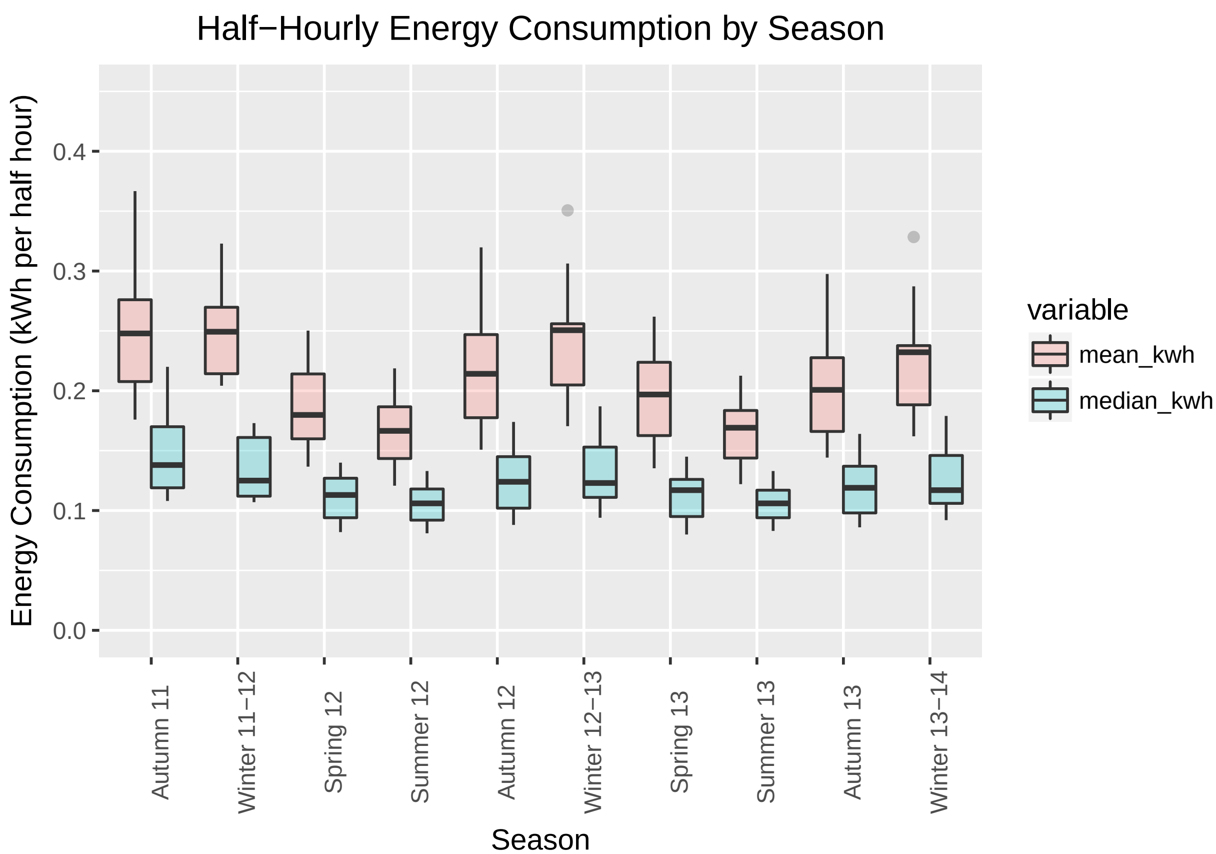
```
#        acorn match_hh median
#   1: ACORN-A        1  0.160
#   2: ACORN-A        2  0.147
#   3: ACORN-A        3  0.138
#   4: ACORN-A        4  0.133
#   5: ACORN-A        5  0.130
#  ---
# 812: ACORN-Q       18  0.096
# 813: ACORN-Q       19  0.097
# 814: ACORN-Q       20  0.097
# 815: ACORN-Q       21  0.097
# 816: ACORN-Q       22  0.098

## Perhaps we can compress this down a little bit. Instead of looking at every
## half-hour period, we could split it into:
##
##       Awake: 6am - 9am
##         Day: 9am - 9pm
##       Sleep: 9pm - 6am
time_of_day = function(x) {
  stopifnot(inherits(x, "POSIXt"),
            hour(x) %in% 0:23,
            minute(x) %in% c(0, 30))
  hr = hour(x)
  ifelse(hr >= 21 | hr < 6, "Sleep",
        ifelse(hr >= 6 & hr <= 9, "Awake", "Day"))
}
## Example
samp = sample(lcl$datetime, 10)
samp
#  [1] "2012-08-21 04:30:00 UTC" "2013-07-13 22:00:00 UTC" "2012-05-18 18:30:00 UTC"
#  [4] "2013-07-03 15:00:00 UTC" "2012-10-19 09:30:00 UTC" "2013-12-04 12:00:00 UTC"
#  [7] "2013-03-30 02:00:00 UTC" "2012-08-15 15:30:00 UTC" "2013-12-15 16:30:00 UTC"
# [10] "2012-12-26 20:30:00 UTC"
time_of_day(samp)
#  [1] "night"   "night"   "day"     "day"     "morning" "day"     "night"   "day"
#  [9] "day"     "night"

## It would probably be useful to have season information as well. The season
## cutoff dates for the 2011-2014 were:
##
## 2011: 2011-03-20 (Spring), 2011-06-21 (Summer)
##       2011-09-23 (Autumn), 2011-12-22 (Winter)
##
## 2012: 2012-03-20 (Spring), 2012-06-20 (Summer)
##       2012-09-22 (Autumn), 2012-12-21 (Winter)
##
## 2013: 2013-03-20 (Spring), 2013-06-21 (Summer)
##       2013-09-22 (Autumn), 2013-12-21 (Winter)
##
## 2014: 2014-03-20 (Spring), 2013-06-21 (Summer)
##       2014-09-23 (Autumn), 2014-12-21 (Winter)
seas = c("Spring", "Summer", "Autumn", "Winter")
```

```r
labs = c(paste(seas[3],  11), paste(seas[4], "11-12"),
         paste(seas[-4], 12), paste(seas[4], "12-13"),
         paste(seas[-4], 13), paste(seas[4], "13-14"))
brks = as.Date(c("2011-09-23", "2011-12-22",
                 "2012-03-20", "2012-06-20", "2012-09-22", "2012-12-21",
                 "2013-03-20", "2013-06-21", "2013-09-22", "2013-12-21",
                 "2014-03-20"))

## Let's add on variables for describing the time of day, day of week,
## season, and half-hour period. We also convert all categorical variables
## into factors.
ids = unique(lcl$id)
tods = c("Awake", "Day", "Sleep")
tous = c("Std", "ToU")
lcl[, `:=`(tod = factor(time_of_day(datetime), levels = tods),
           seas = cut(as.Date(datetime), breaks = brks, labels = labs),
           hh_period = factor(match_hh(datetime), levels = 1:48),
           acorn_grp = factor(acorn_grp, levels = valid_grp),
           acorn = factor(acorn, levels = valid_code),
           id = factor(id, levels = ids),
           tou = factor(tou, levels = tous),
           wday = factor(wday(datetime), 1:7))]

## Look at average kwh_hh values by hh period, time of day, season, and ACORN group.
sub = lcl[order(seas),
          .(mean_kwh = mean(kwh_hh),
            median_kwh = median(kwh_hh),
            iqr_kwh = IQR(kwh_hh)),
          .(tod, seas, acorn_grp)]

head(sub, 10)
#       tod        seas   acorn_grp  mean_kwh median_kwh iqr_kwh
#  1:   Day   Autumn 11   Adversity 0.2991763      0.189 0.28200
#  2: Sleep   Autumn 11   Adversity 0.2076762      0.108 0.18500
#  3: Awake   Autumn 11   Adversity 0.2110109      0.119 0.19125
#  4: Awake   Autumn 11    Affluent 0.2479052      0.138 0.22100
#  5:   Day   Autumn 11    Affluent 0.3667230      0.220 0.38000
#  6: Sleep   Autumn 11    Affluent 0.2760183      0.153 0.25600
#  7: Awake   Autumn 11 Comfortable 0.2025312      0.125 0.16100
#  8:   Day   Autumn 11 Comfortable 0.2607550      0.170 0.23700
#  9: Sleep   Autumn 11 Comfortable 0.1759550      0.108 0.16000
# 10:   Day Winter 11-12   Adversity 0.2697229      0.161 0.24800

## It would be quite useful to plot this. Let's convert the table into long
## form where each average kwh_hh value (mean, median, iqr) is a separate obs.
## We create a series of subtables (for convenience and to avoid recomputing)
## for plotting.
sub = melt(sub, measure = patterns("_kwh$"))
sub
#        tod        seas   acorn_grp variable     value
#   1:   Day   Autumn 11   Adversity mean_kwh 0.2991763
#   2: Sleep   Autumn 11   Adversity mean_kwh 0.2076762
#   3: Awake   Autumn 11   Adversity mean_kwh 0.2110109
```

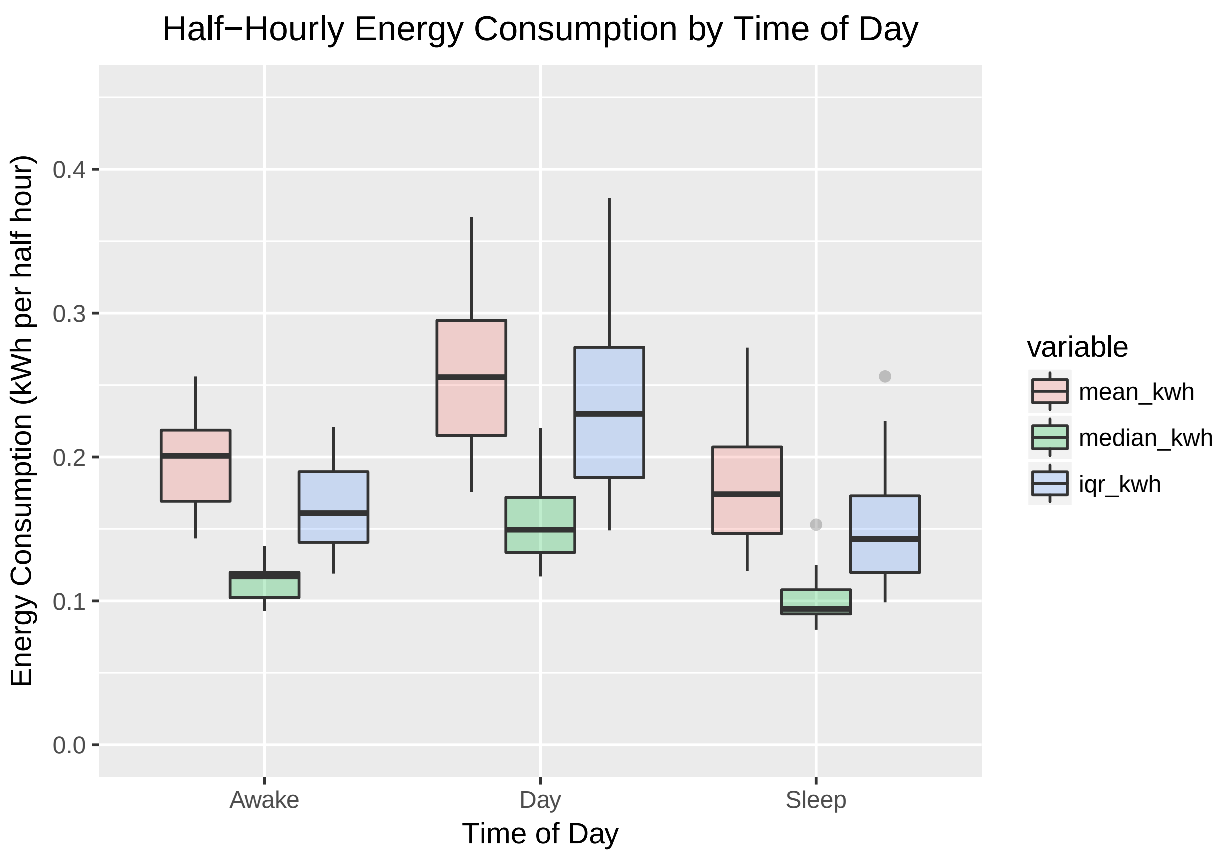
```
#   4: Awake    Autumn 11    Affluent mean_kwh 0.2479052
#   5:   Day    Autumn 11    Affluent mean_kwh 0.3667230
#  ---
# 266: Awake Winter 13-14   Adversity  iqr_kwh 0.1370000
# 267:   Day Winter 13-14   Adversity  iqr_kwh 0.2150000
# 268: Sleep Winter 13-14 Comfortable  iqr_kwh 0.1650000
# 269: Awake Winter 13-14 Comfortable  iqr_kwh 0.1680000
# 270:   Day Winter 13-14 Comfortable  iqr_kwh 0.2650000

## Other useful tables.
##
## Group by 30min period, ACORN group, and time of day.
sub2 = lcl[, .(mean_kwh = mean(kwh_hh),
               med_kwh = median(kwh_hh)),
           .(hh_period, acorn_grp, tod)]
sub2 = sub2 %>% melt(measure.vars = patterns("_kwh$"))
sub2
#      hh_period   acorn_grp   tod variable     value
#   1:         1    Affluent Sleep mean_kwh 0.1914601
#   2:         2    Affluent Sleep mean_kwh 0.1723959
#   3:         3    Affluent Sleep mean_kwh 0.1576239
#   4:         4    Affluent Sleep mean_kwh 0.1469427
#   5:         5    Affluent Sleep mean_kwh 0.1407429
#  ---
# 284:        19 Comfortable Awake  med_kwh 0.1280000
# 285:        20 Comfortable   Day  med_kwh 0.1260000
# 286:        21 Comfortable   Day  med_kwh 0.1240000
# 287:        22 Comfortable   Day  med_kwh 0.1230000
# 288:        23 Comfortable   Day  med_kwh 0.1240000

## Group by 30min period, ACORN group, and time of use.
sub3 = lcl[, .(mean_kwh = mean(kwh_hh),
               med_kwh = median(kwh_hh)),
           .(hh_period, acorn_grp, tou)] %>%
  melt(measure.vars = patterns("_kwh$"))
#     hh_period   acorn_grp tou variable     value
#   1:         1    Affluent Std mean_kwh 0.2036458
#   2:         2    Affluent Std mean_kwh 0.1836262
#   3:         3    Affluent Std mean_kwh 0.1673017
#   4:         4    Affluent Std mean_kwh 0.1552036
#   5:         5    Affluent Std mean_kwh 0.1480318
#  ---
# 572:        18 Comfortable ToU  med_kwh 0.1240000
# 573:        19 Comfortable ToU  med_kwh 0.1210000
# 574:        20 Comfortable ToU  med_kwh 0.1190000
# 575:        21 Comfortable ToU  med_kwh 0.1170000
# 576:        22 Comfortable ToU  med_kwh 0.1160000
```

```
## Generate half hour labels for plotting.
half_hr = paste(rep(sprintf("%02d", 0:23), each = 2),
                c("00", "30"), sep = ":")
names(half_hr) = 1:48
ind = (0:48)[c(FALSE, FALSE, TRUE, FALSE)]
hh = c(half_hr[-1], half_hr[1])[ind]
```

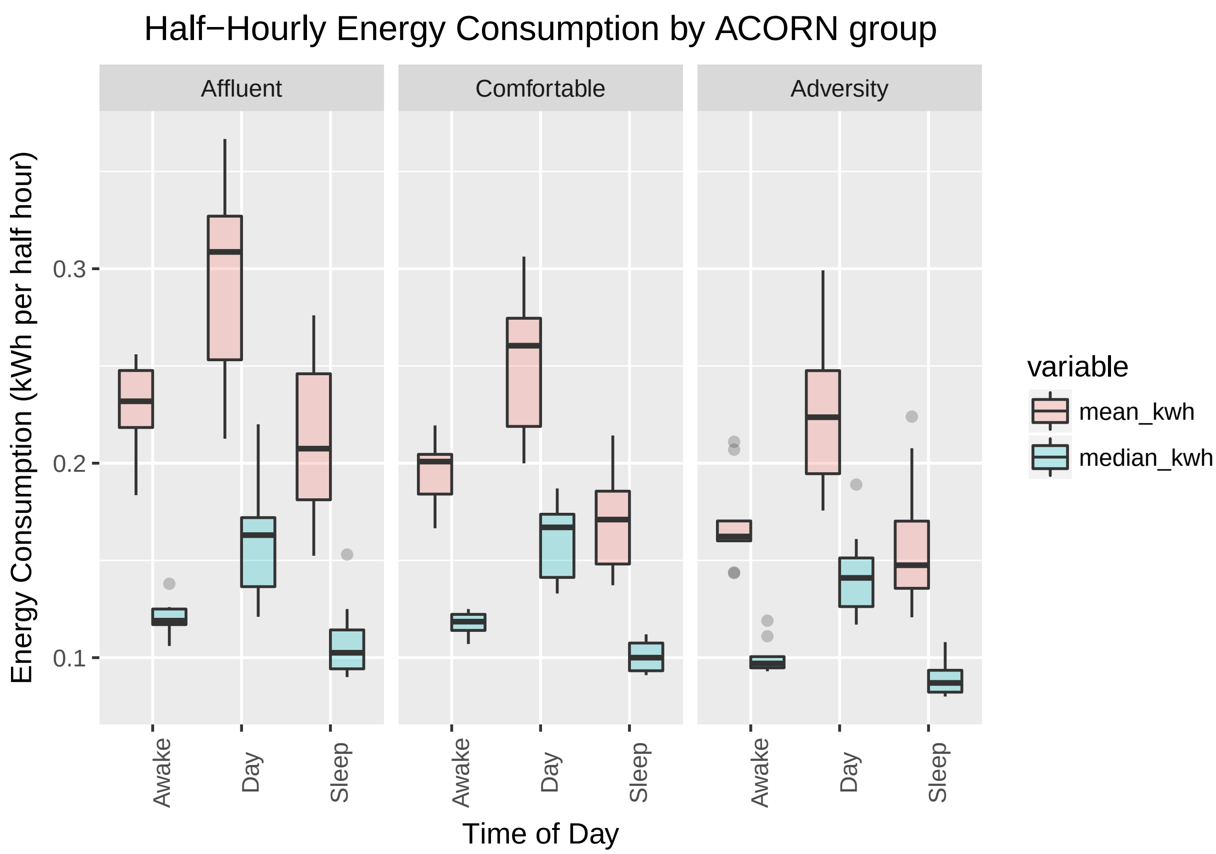
```r
cols = c(rgb(1, 0, 0, .3), rgb(0, 0, 1, .3), rgb(0, 1, 0, .3))

## Plot mean and median kwh_hh by season.
sub[variable != "iqr_kwh"] %>%
  ggplot(aes(seas, value)) +
  geom_boxplot(aes(fill = variable), alpha = 0.25) +
  scale_y_continuous(limit = c(0, .45)) +
  labs(title = "Half-Hourly Energy Consumption by Season",
       x = "Season", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 90))
```



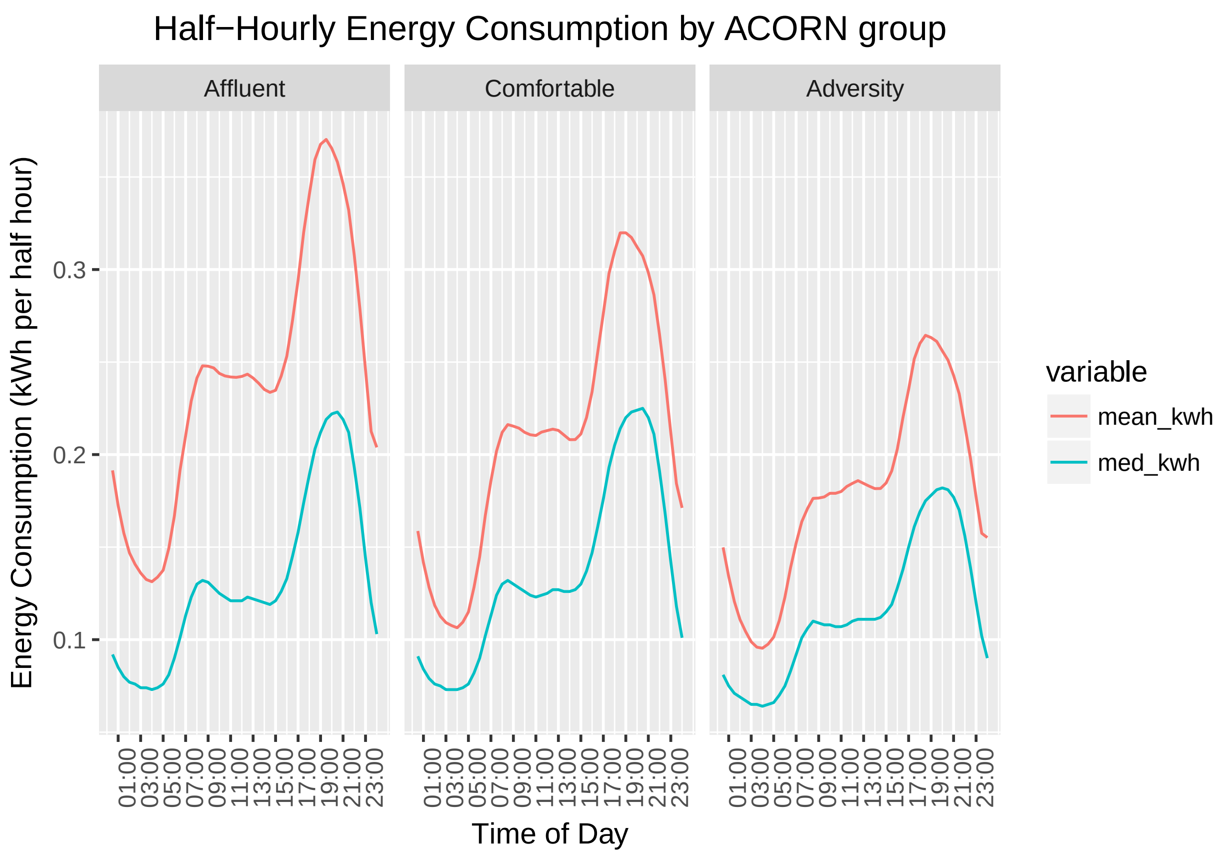

```r
## kwh_hh values by time of day.
sub %>%
  ggplot(aes(tod, value)) +
  geom_boxplot(aes(fill = variable), alpha = 0.25) +
  scale_y_continuous(limit = c(0, .45)) +
  labs(title = "Half-Hourly Energy Consumption by Time of Day",
       x = "Time of Day", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5))
```



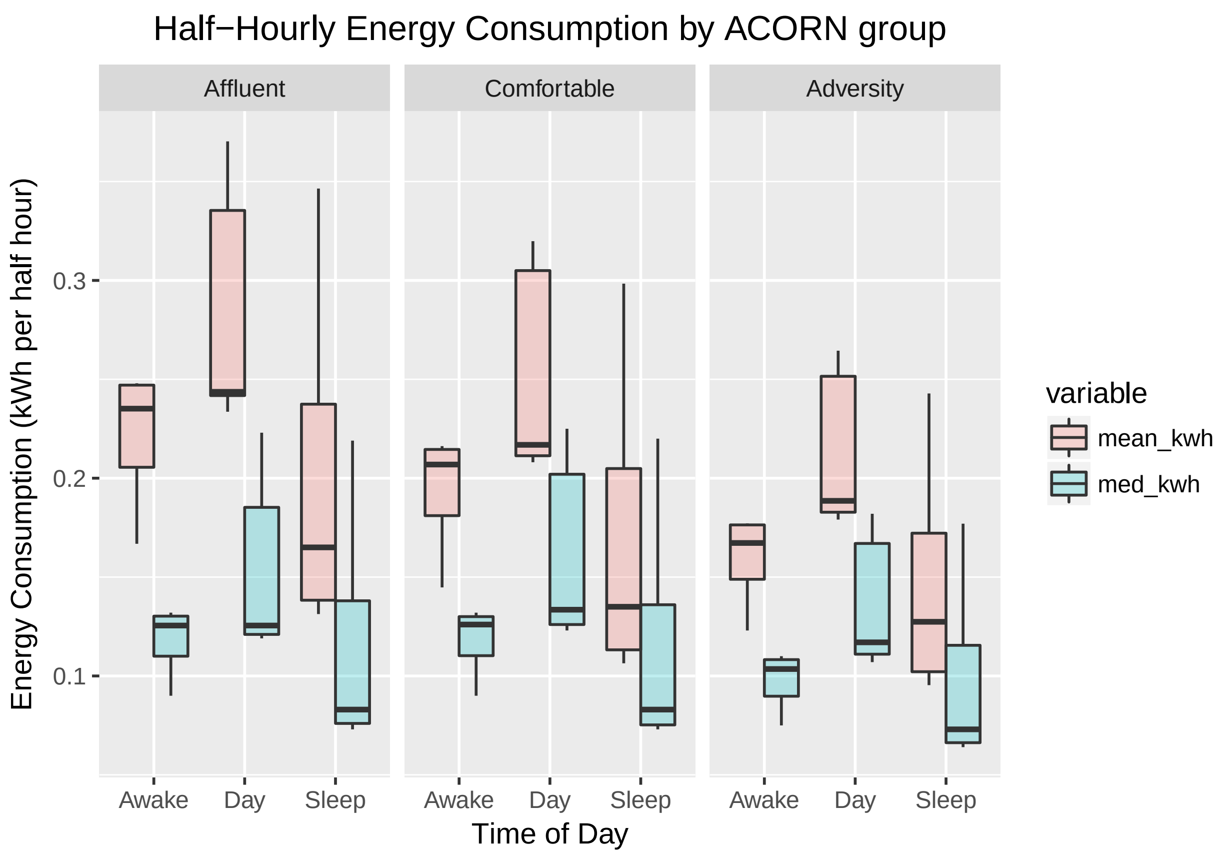

```
## kwh_hh by season and ACORN group.
sub[variable != "iqr_kwh"] %>%
  ggplot(aes(tod, value)) +
  geom_boxplot(aes(fill = variable), alpha = .25) +
  facet_grid(~ acorn_grp) +
  labs(title = "Half-Hourly Energy Consumption by ACORN group",
       x = "Time of Day", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 90))
```



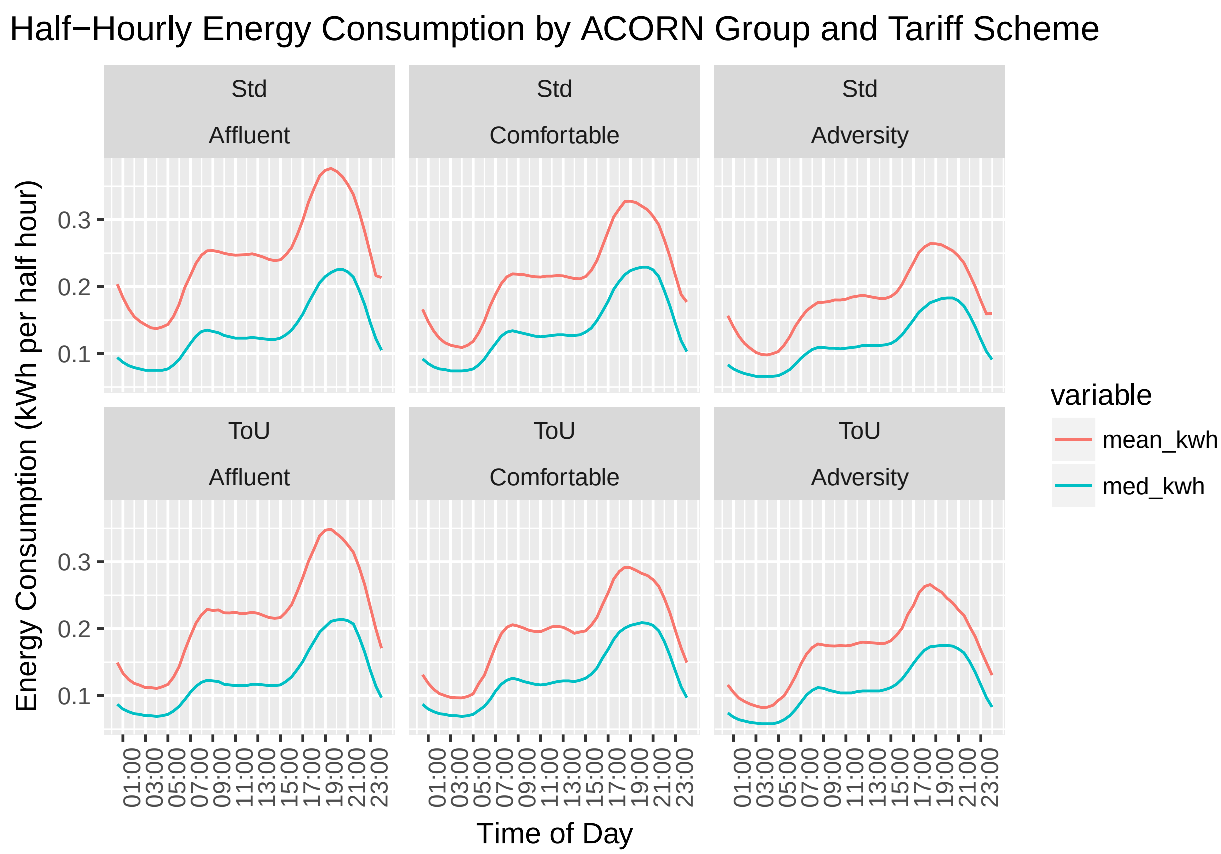

```
## kwh_hh by time of day and ACORN group.
sub2 %>%
  ggplot(aes(hh_period, value, col = variable)) +
  geom_line() +
  facet_wrap(~ acorn_grp) +
  scale_x_continuous(breaks = ind, lab = hh) +
  labs(title = "Half-Hourly Energy Consumption by ACORN group",
       x = "Time of Day", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 90))
```




```
## kwh_hh by time of day and ACORN group.
sub2 %>%
  ggplot(aes(tod, value)) +
  geom_boxplot(aes(fill = variable), alpha = .25) +
  facet_wrap(~ acorn_grp) +
  labs(title = "Half-Hourly Energy Consumption by ACORN group",
       x = "Time of Day", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
## kwh_hh by time of use and ACORN group.
sub3 %>%
  ggplot(aes(hh_period, value, col = variable)) +
  geom_line() +
  facet_wrap(~ tou + acorn_grp) +
  scale_x_continuous(breaks = ind, lab = hh) +
  labs(title = "Half-Hourly Energy Consumption by ACORN Group and Tariff Scheme",
       x = "Time of Day", y = "Energy Consumption (kWh per half hour)") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 90))
```



At the data visualization stage, it may be useful to create a dashboard to try out different types of graphs. (it's probably not very practical to do this for data cleaning and manipulation, as these steps tend to be rather programmatic). In R, you can do this using the `shiny` package. Some examples of Shiny apps can be found on Shiny Gallery and Shiny User Showcase (source code included).