Tidyverse

Rachel

2022-11-03

In this portfolio, I will use the 'ASHRAE - Great Energy Predictor III' dataset available on Kaggle [1] to illustrate how we can use the Tidyverse to read/write, manipulate, view and plot large datasets. Here I will only consider 2 of the 5 CSV files:

- `building_metadata.csv`
- `train.csv`

with the aim of visualising how different types of buildings consume energy throughout the year.

First we load the relevant Tidyverse packages into `R`:

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(ggplot2)
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

Then read in our data using `readr`:

[1] https://www.kaggle.com/c/ashrae-energy-prediction

```
train <- read_csv("train.csv", show_col_types = FALSE)
buildMeta <- read_csv("building_metadata.csv", show_col_types = FALSE)
```

pipes

A feature of `R` that the Tidyverse makes use of frequently is `pipes`. This allows the object on the left-hand side to be passed as the first argument of the function on the right hand side. This is usually most helpful when we have many nested functions, the code becomes more readable. Below is an example which in reality would be unnecessary, but illustrates how `pipes` work

```
train %>% head()
```

```
## # A tibble: 6 x 4
##   building_id meter timestamp           meter_reading
##         <dbl> <dbl> <dttm>                      <dbl>
## 1           0     0 2016-01-01 00:00:00             0
## 2           1     0 2016-01-01 00:00:00             0
## 3           2     0 2016-01-01 00:00:00             0
## 4           3     0 2016-01-01 00:00:00             0
## 5           4     0 2016-01-01 00:00:00             0
## 6           5     0 2016-01-01 00:00:00             0
```

```
buildMeta %>% head()
```

```
## # A tibble: 6 x 6
##   site_id building_id primary_use square_feet year_built floor_count
##     <dbl>       <dbl> <chr>             <dbl>      <dbl>       <dbl>
## 1       0           0 Education          7432       2008          NA
## 2       0           1 Education          2720       2004          NA
## 3       0           2 Education          5376       1991          NA
## 4       0           3 Education         23685       2002          NA
## 5       0           4 Education        116607       1975          NA
## 6       0           5 Education          8000       2000          NA
```

The following sections will make use of `pipes` and hopefully make it clearer why using pipes is beneficial.

dplyr and tidyr

Many functions are particular about the format of the data needed, so `dplyr` and `tidyr` allow us to manipulate our data to get it into an appropriate format.

In our context, we want to plot an average energy consumption for each type of building. The first step for this would be to attach the building metadata from `buildMeta` to each of the meter reading entries in `train`:

```
Meta <- buildMeta[train$building_id+1,]
train_w_meta <- cbind(train, primary_use = Meta$primary_use)
```

We now have the building type and readings in the same dataframe:

```
head(train_w_meta)
```

```
##   building_id meter  timestamp meter_reading primary_use
## 1           0     0 2016-01-01             0   Education
## 2           1     0 2016-01-01             0   Education
## 3           2     0 2016-01-01             0   Education
## 4           3     0 2016-01-01             0   Education
## 5           4     0 2016-01-01             0   Education
## 6           5     0 2016-01-01             0   Education
```

```
class(train_w_meta$primary_use)
```

```
## [1] "character"
```

We can see however that `primary_use` is listed as type `chr`, however for us to group the readings by the type, it is easier to have this as a `fctr` variable:

```
train_w_meta$primary_use <- as.factor(train_w_meta$primary_use)
```

Up until this point, we haven't actually used any Tidyverse functions to manipulate our data frame. The `base R` functions are adequate for basic data manipulation, but we now need to take average readings for each building type at every timestamp. Here is where we making use of the Tidyverse can make this easier, the `group_by()` and `summarise()` functions from `dplyr` allows us to this in just a few lines

```
consumption <- train_w_meta %>%
               group_by(primary_use, timestamp) %>%
               summarise(average_reading=median(meter_reading))
```

```
## `summarise()` has grouped output by 'primary_use'. You can override using the
## `.groups` argument.
```

To do the same computations in `base R` it would be difficult to even see how to start with this.

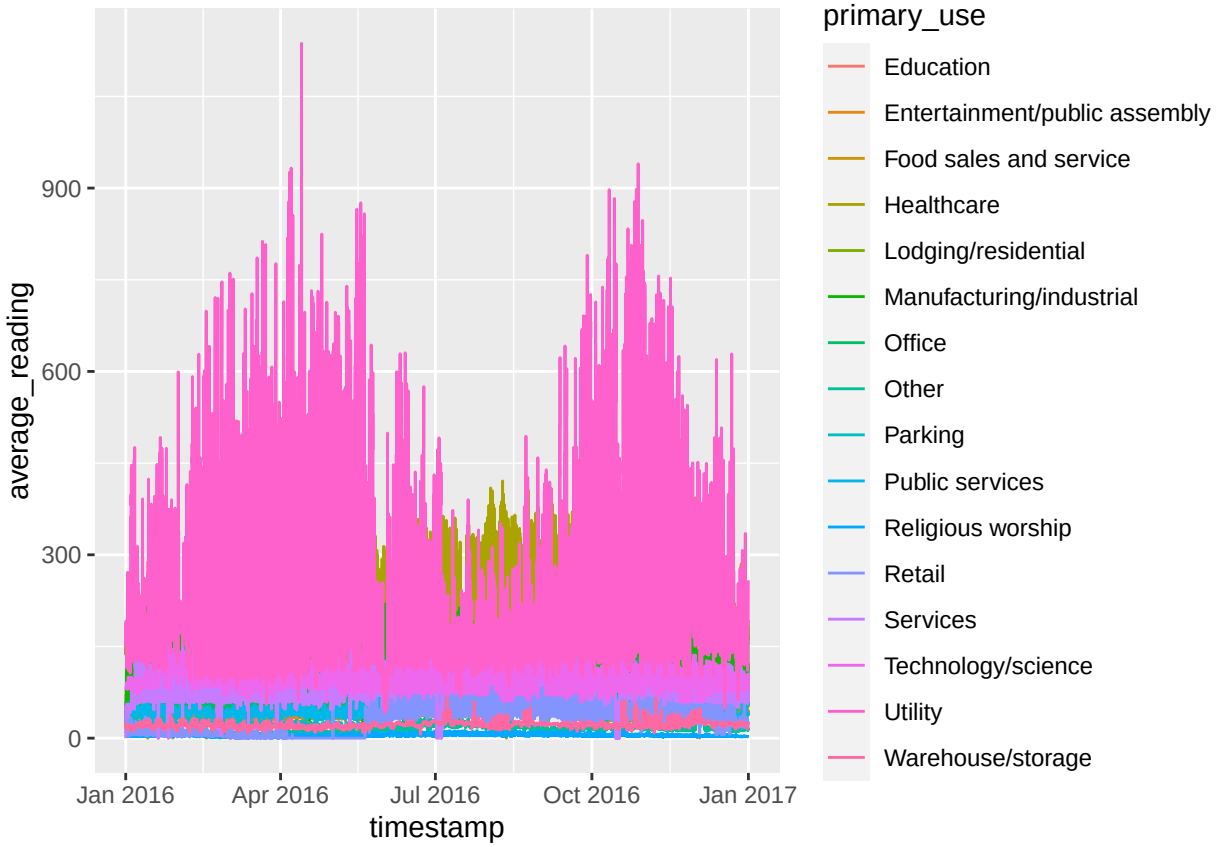
ggplot2

Now that our data is in an adequate form, we can pass the `consumption` data frame to `ggplot()`. Rather than the `base R` plotting function, we build `ggplot`s through layers. We start with just passing our data to the `ggplot()` function:

```
ggplot(consumption)
```

but as we can see there is no plot since we need to add a layer to indicate we want to plot lines, sorted by the building type. We do this by adding a `geom_line` layer to the plot, which contains a mapping argument `aes()`, which informs ggplot of which variables we want include and how:

```
ggplot(consumption) +
  geom_line(aes(x=timestamp, y = average_reading, colour = primary_use))
```



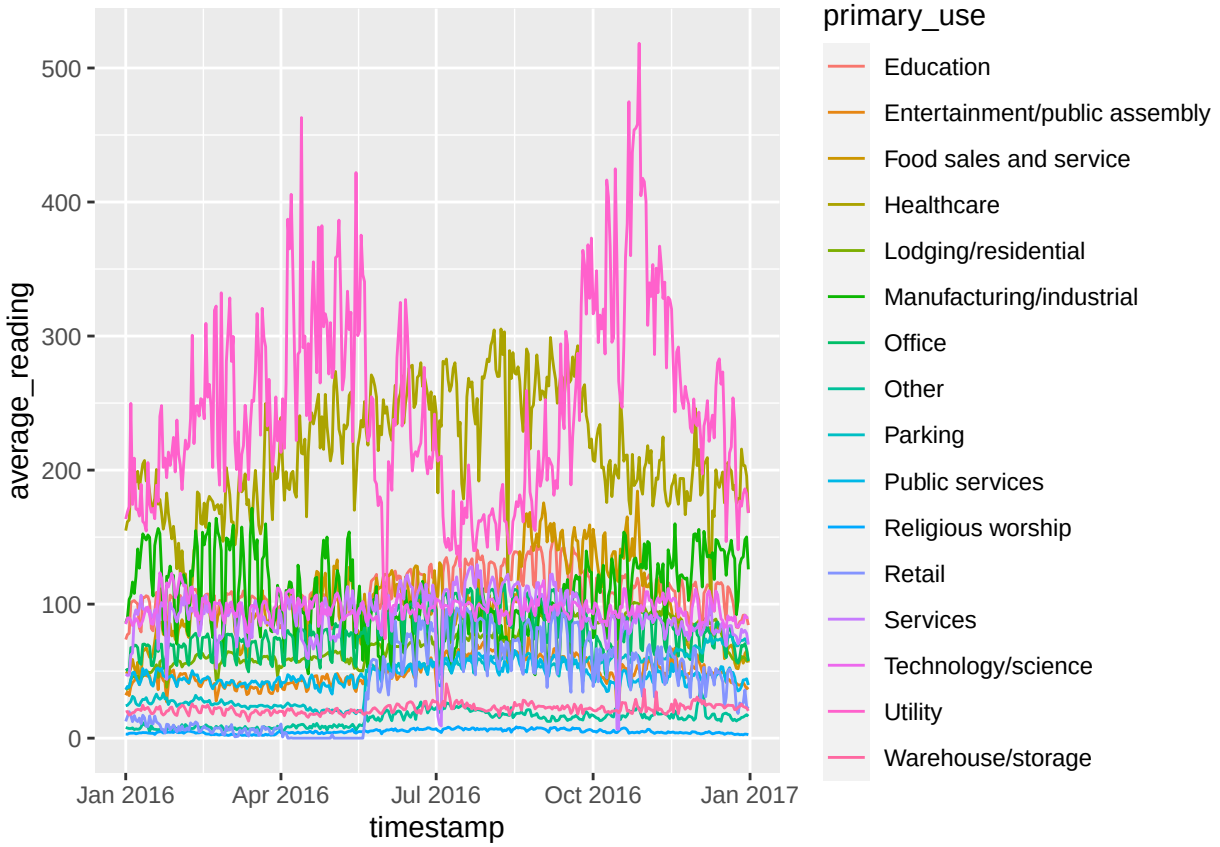
We can see that there is a lot of fluctuation, most likely due to a daily cycle, so we can take an average for each day:

```
consumption$timestamp <- as.Date(consumption$timestamp)
consumption_day <- consumption %>%
               group_by(timestamp,primary_use) %>%
               summarise(average_reading=mean(average_reading))
```

```
## `summarise()` has grouped output by 'timestamp'. You can override using the
## `.groups` argument.
```

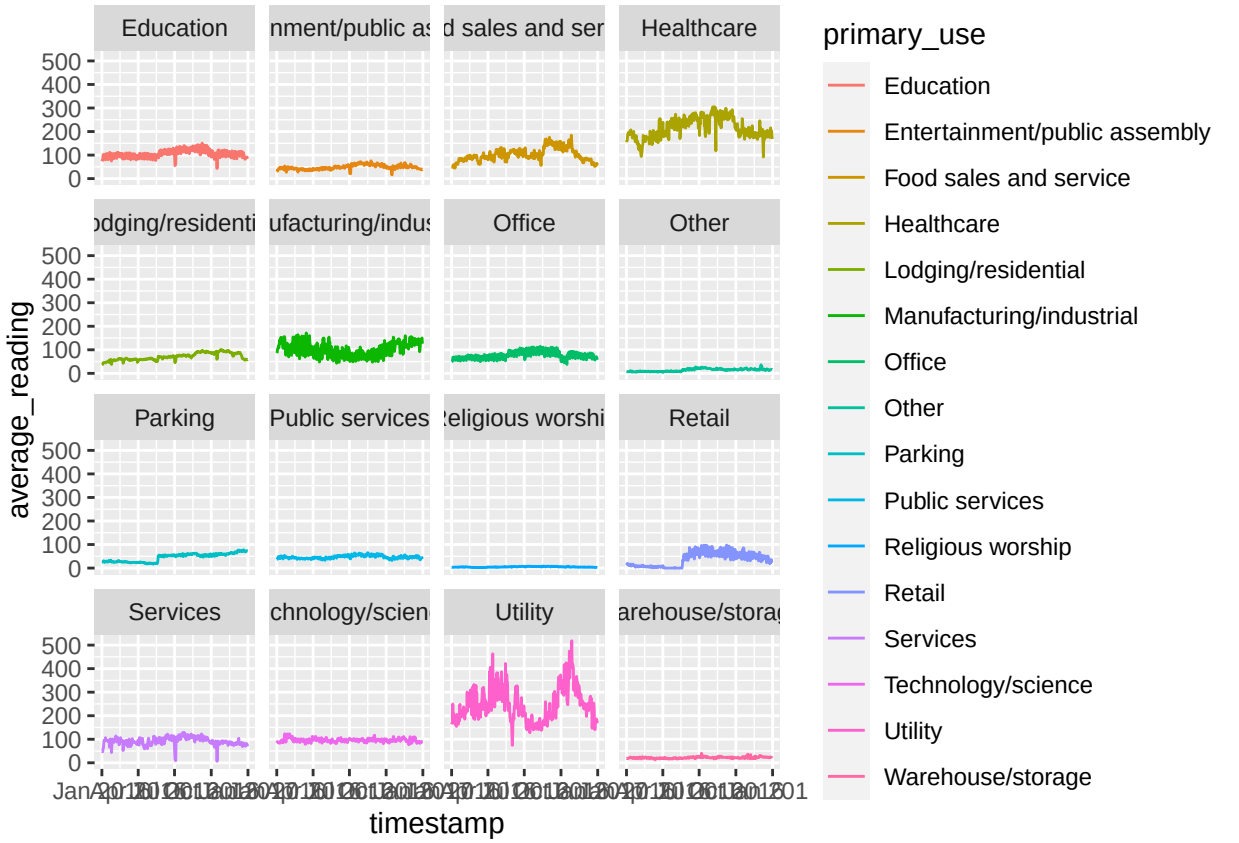
We then get smoother lines:

```
ggplot(consumption_day) +
  geom_line(aes(x=timestamp, y= average_reading, colour = primary_use))
```



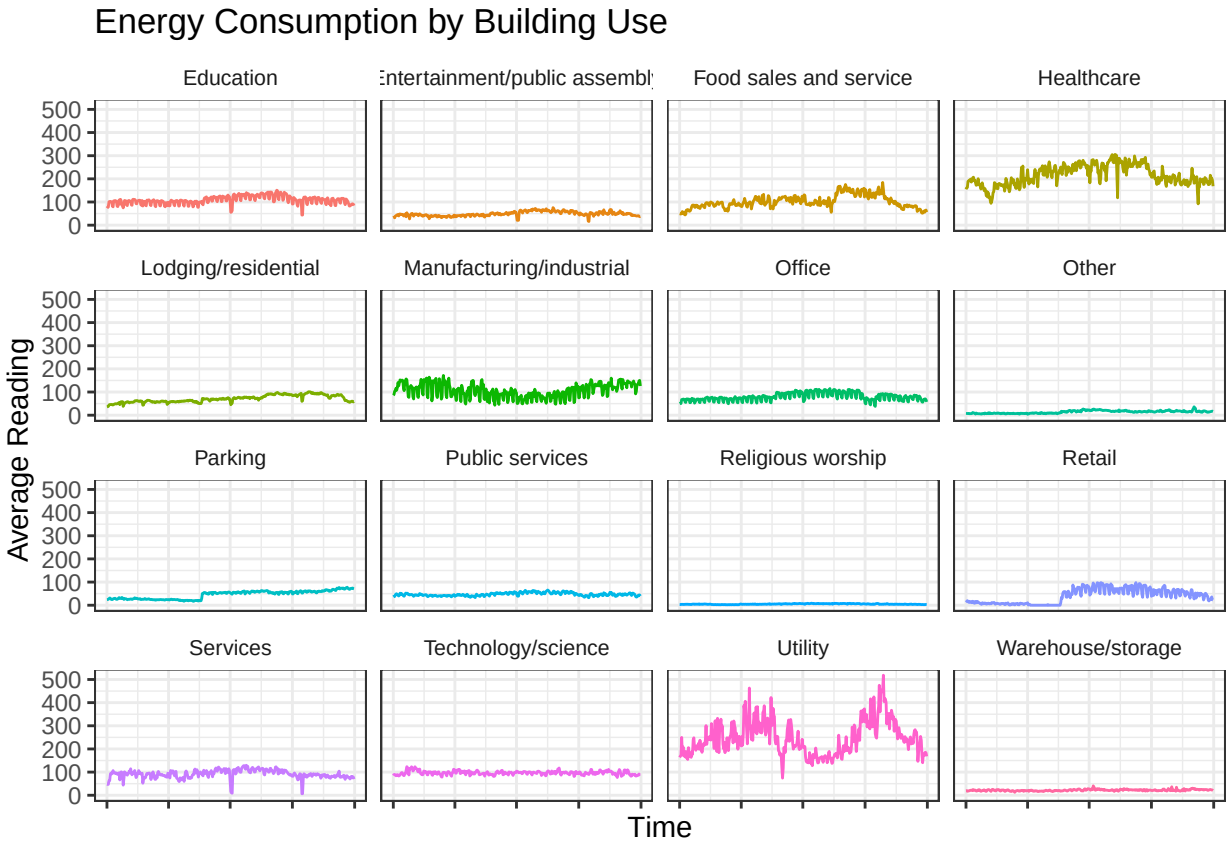
However, with so many lines, it is hard to see what is going on, so we can use `facet_wrap()` to create a grid of plots:

```
ggplot(consumption_day) +
  geom_line(aes(x=timestamp, y= average_reading, colour = primary_use)) +
  facet_wrap(~ primary_use)
```



However we can see this plot looks quite messy, so we can change some of the default theme and label settings by adding `theme()` and `labs()` layers to make a much nicer looking plot:

```
ggplot(consumption_day) +
  geom_line(aes(x=timestamp, y= average_reading, colour = primary_use)) +
  facet_wrap(~ primary_use) +
  theme_bw() +
  theme(legend.position = "none", axis.text.x = element_blank(), strip.text.x =
          element_text(size = 8), strip.background = element_rect(color="white", fill="white")) +
  labs(x = "Time",y = "Average Reading",title = "Energy Consumption by Building Use")
```


Energy Consumption by Building Use
Education
Entertainment/public assembl
Food sales and service
Healthcare
Lodging/residential
Manufacturing/industrial
Office
Other
Parking
Public services
Religious worship
Retail
Services
Technology/science
Utility
Warehouse/storage
Average Reading
0
100
200
300
400
500
Time