Portfolio 5

Sam Bowyer

The Tidyverse

On tidyverse.org, the tidyverse is described as "an opinionated collection of R packages designed for data science" which "share an underlying deign philosophy, grammar and data structures". Not only does this provide us with very useful functionality, but provides it in a way that helps keep cope readable and therefore more easily maintainable.

The collection of packages is quite large (see tidyverse.org/packages), however, the main eight packages are:

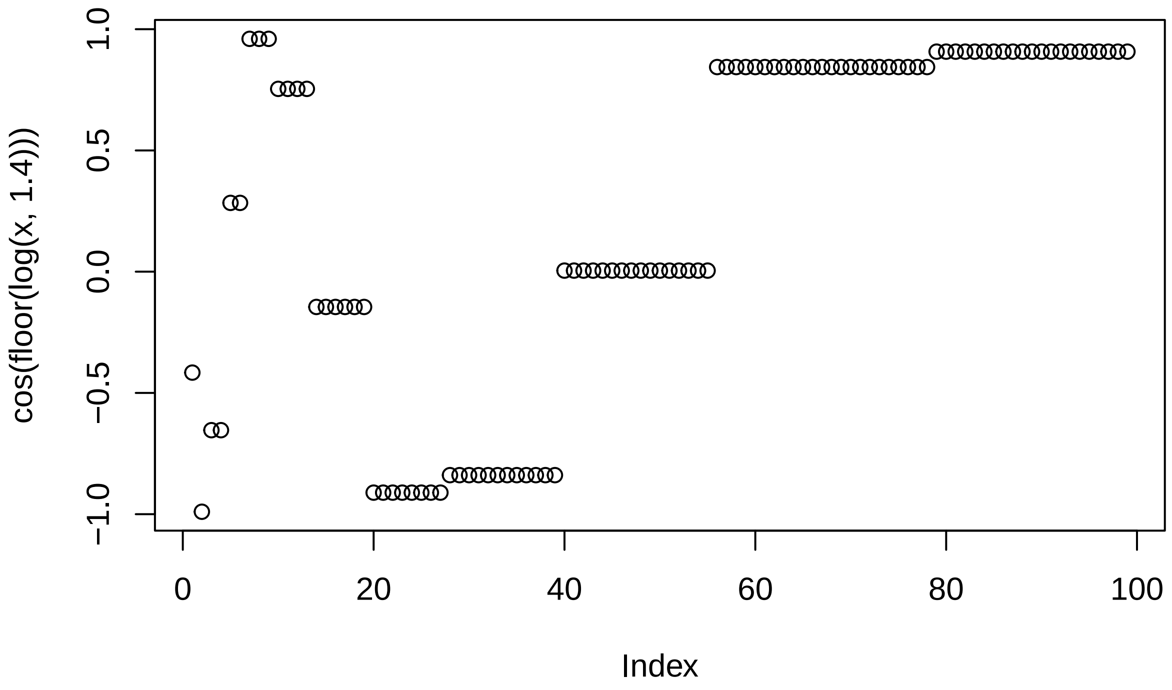
- `ggplot2` - for creating graphics
- `dplyr` - for data manipulation
- `tidyr` - for tidying data
- `readr` - for reading data files (such as csv, tsv etc.)
- `purrr` - for extending `R`'s functional programming capabilities
- `tibble` - a useful implementation of dataframes
- `stringr` - for working with strings
- `forcats` - for working with factors (categorical data)

We'll go through an example of working with some of these packages that will give an idea of how the tidyverse is used for data analysis, however, first it will be useful to work with another package: `magrittr`.

Pipes in `magrittr`

The package `magrittr` allows us to use the pipe operator `%>%` which can simplify and make code more readable, for example we can rewrite this code

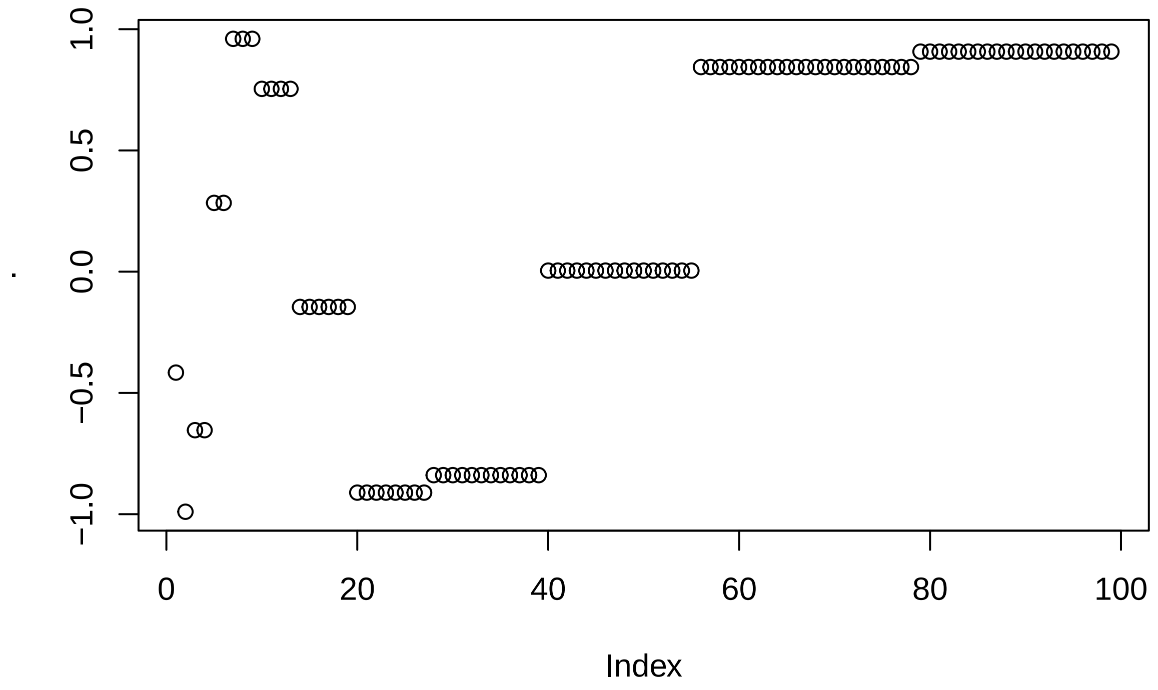
```r
x = 2:100
plot(cos(floor(log(x, 1.4))))
```



as the following

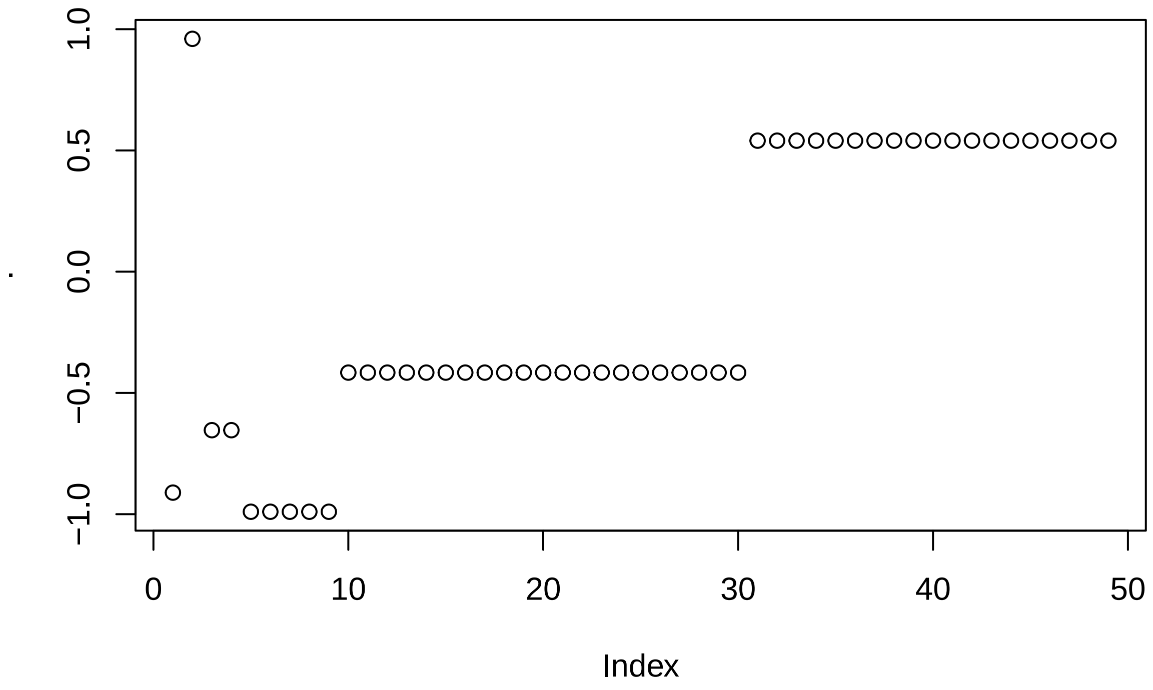
```
library(magrittr)

x %>% log(. , 1.4) %>% floor %>% cos %>% plot
```



So the output of whatever is to the left of the pipe gets sent as input to the expression on the right of the pipe, and as we saw with the `log(., 1.4)` part of the second example we can reference the left side's output directly with "`.`" if we want to specify other arguments to a function that we're using. If we want to 'pipe' the left-hand-side output to the second argument of a function we could write this as:

```
x = 2:50
x %>% log(1000, .) %>% floor %>% cos %>% plot
```



Data Analysis With The Tidyverse

Next we'll give a short example of using pipes and dataframes to analyse weather and meter reading data in different buildings at different locations in the UK during 2016, as per ASHRAE's Great Energy Predictor III competition on Kaggle.

First we'll load in the `tidyverse` packages and the package `gridExtra` which will help us to organise the layout of multiple `ggplot2` plots.

```
library(tidyverse)
library(gridExtra)
```

Then we'll load the meter reading and weather data (observe that this gives us some metadata about the datasets).

```
meters = read_csv("data/train.csv")
```

```
## Rows: 20216100 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (3): building_id, meter, meter_reading
## dttm (1): timestamp
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
weather = read_csv("data/weather_train.csv")
```

```
## Rows: 139773 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (8): site_id, air_temperature, cloud_coverage, dew_temperature, precip_...
## dttm (1): timestamp
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can now inspect both datasets to better figure out what we're working with.

```
meters
```

```
## # A tibble: 20,216,100 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
##  7           6     0 2016-01-01 00:00:00             0
##  8           7     0 2016-01-01 00:00:00             0
##  9           8     0 2016-01-01 00:00:00             0
## 10           9     0 2016-01-01 00:00:00             0
## # ... with 20,216,090 more rows
```

```
weather
```

```
## # A tibble: 139,773 x 9
##    site_id timestamp           air_tem~1 cloud~2 dew_t~3 preci~4 sea_l~5 wind_~6
##      <dbl> <dttm>                  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1       0 2016-01-01 00:00:00      25         6    20        NA   1020.       0
##  2       0 2016-01-01 01:00:00      24.4      NA    21.1      -1   1020.      70
##  3       0 2016-01-01 02:00:00      22.8       2    21.1       0   1020.       0
##  4       0 2016-01-01 03:00:00      21.1       2    20.6       0   1020.       0
##  5       0 2016-01-01 04:00:00      20         2    20        -1   1020      250
##  6       0 2016-01-01 05:00:00      19.4      NA    19.4       0     NA        0
##  7       0 2016-01-01 06:00:00      21.1       6    21.1      -1   1019.       0
##  8       0 2016-01-01 07:00:00      21.1      NA    21.1       0   1019.     210
##  9       0 2016-01-01 08:00:00      20.6      NA    20         0   1018.       0
## 10       0 2016-01-01 09:00:00      21.1      NA    20.6       0   1019      290
## # ... with 139,763 more rows, 1 more variable: wind_speed <dbl>, and
## #   abbreviated variable names 1: air_temperature, 2: cloud_coverage,
## #   3: dew_temperature, 4: precip_depth_1_hr, 5: sea_level_pressure,
## #   6: wind_direction
```

We would like to create a plot that tracks mean meter readings (from each type of meter) along with the weather from January 2016 to January 2017. We can check the values that the `meter` column takes by using the `unique` function as follows:

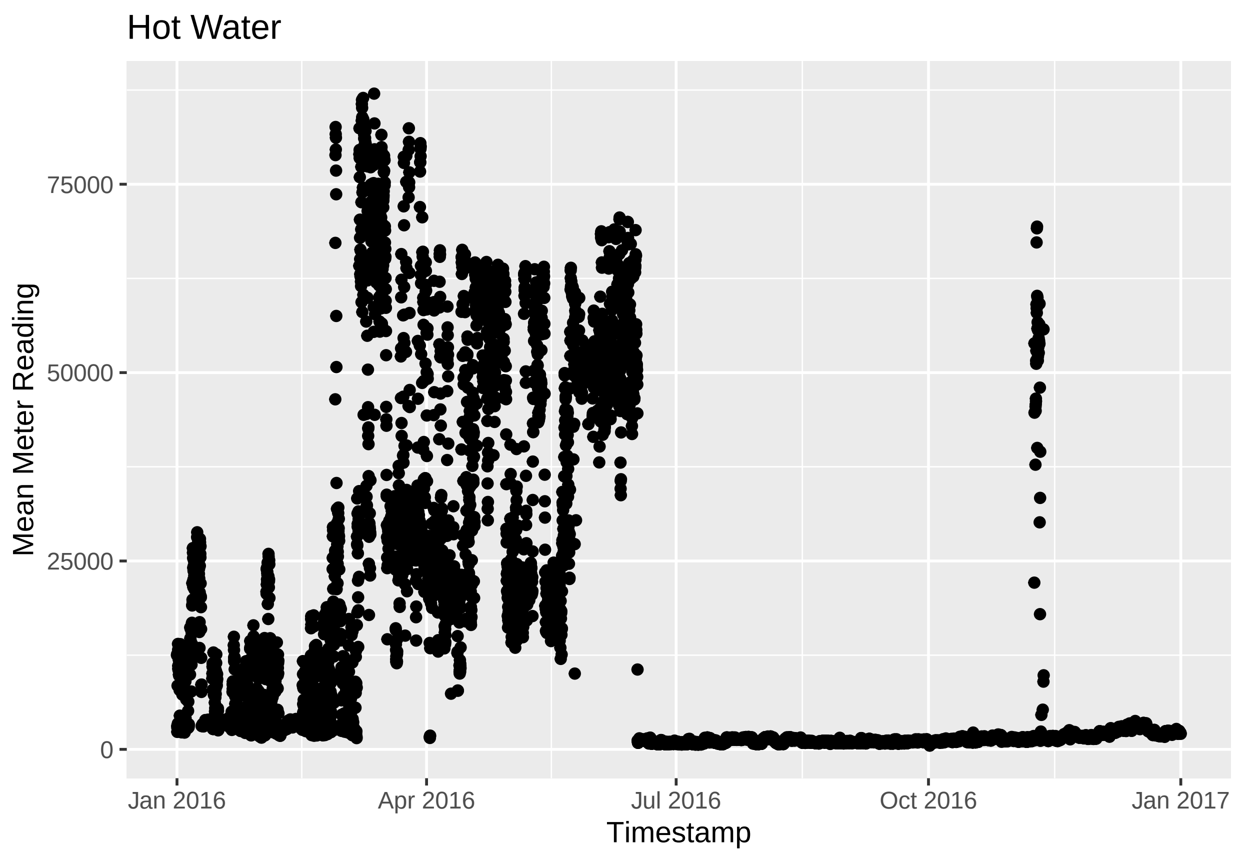
```
meters["meter"] %>% unique
```

```
## # A tibble: 4 x 1
##   meter
##   <dbl>
## 1     0
## 2     3
## 3      1
## 4      2
```

Since we now that each meter takes values 0 to 3 we can write the following function which returns a `ggplot` of the mean meter reading over time for a certain meter type. By following the pipes we can see that the function achieves this by filtering out the meter readings not of our specified type, then averaging over locations by grouping all rows with the same timestamp (with `group_by`) and taking the mean of these groups (through `summarise`). (We define the vector `meterTypes` to help us create titles for each of these plots based on the type of meter we're taking data from.)

```r
meterTypes = c("Chilled Water", "Electric", "Hot Water", "Steam")

meanMeterReadingPlot <- function(meterNo){
  meters %>%
  filter(meter==meterNo) %>%
  group_by(timestamp) %>%
  summarise(mean=mean(meter_reading)) %>%
  ggplot() + geom_point(aes(timestamp, mean)) + ggtitle(meterTypes[meterNo+1]) +
    xlab("Timestamp") + ylab("Mean Meter Reading")
}
meanMeterReadingPlot(2)
```



(Note that the `ggplot2` package inside `tidyverse` allows us to flexibly add elements to the plot with the `+` operator.)

From this plot we see that there is a serious outlier around November which it would be good to remove, perhaps representing readings from a faulty meter. To find the offending building we can use the following series of pipes, which filter the dataframe down to October and November hot water meter readings, then find the `building_id` with the greatest maximum reading:

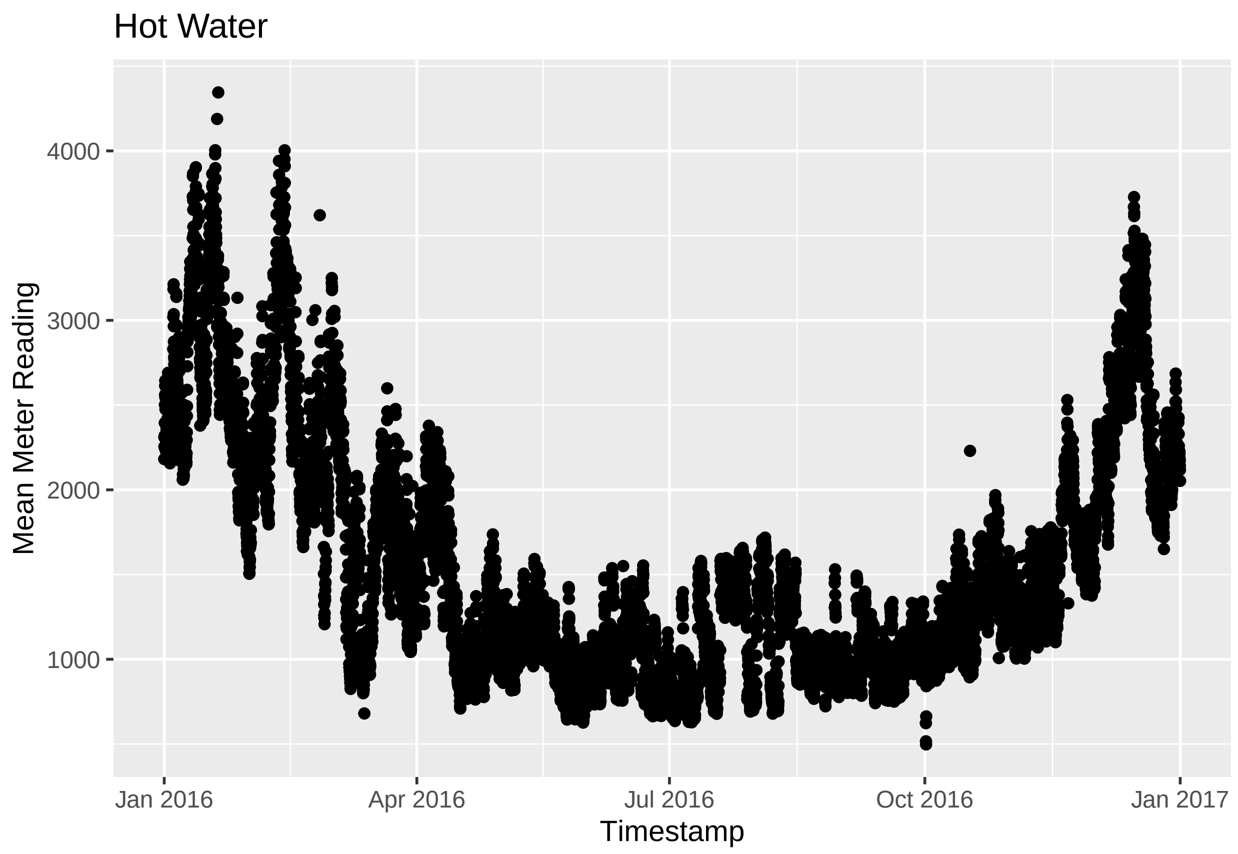
```
outlier_building_id = meters %>%
  filter(timestamp > as.POSIXct("2016-10-1"),
         timestamp < as.POSIXct("2016-12-1"), meter==2) %>%
  group_by(building_id) %>% summarise(max_reading = max(meter_reading)) %>%
  arrange(desc(max_reading)) %>%
  head(1) %>%
  pull(building_id)

outlier_building_id
```

```
## [1] 1099
```

Thus we can rewrite out `meanMeterReadingPlot` function to exclude this building by adding in the line `filter(building_id!=outlier_building_id)`, so the function becomes:

```
meanMeterReadingPlot <- function(meterNo){
  meters %>%
  filter(meter==meterNo) %>%
  filter(building_id!=outlier_building_id) %>%
  group_by(timestamp) %>%
  summarise(mean=mean(meter_reading)) %>%
  ggplot() + geom_point(aes(timestamp, mean)) + ggtitle(meterTypes[meterNo+1]) +
    xlab("Timestamp") + ylab("Mean Meter Reading")
}
meanMeterReadingPlot(2)
```

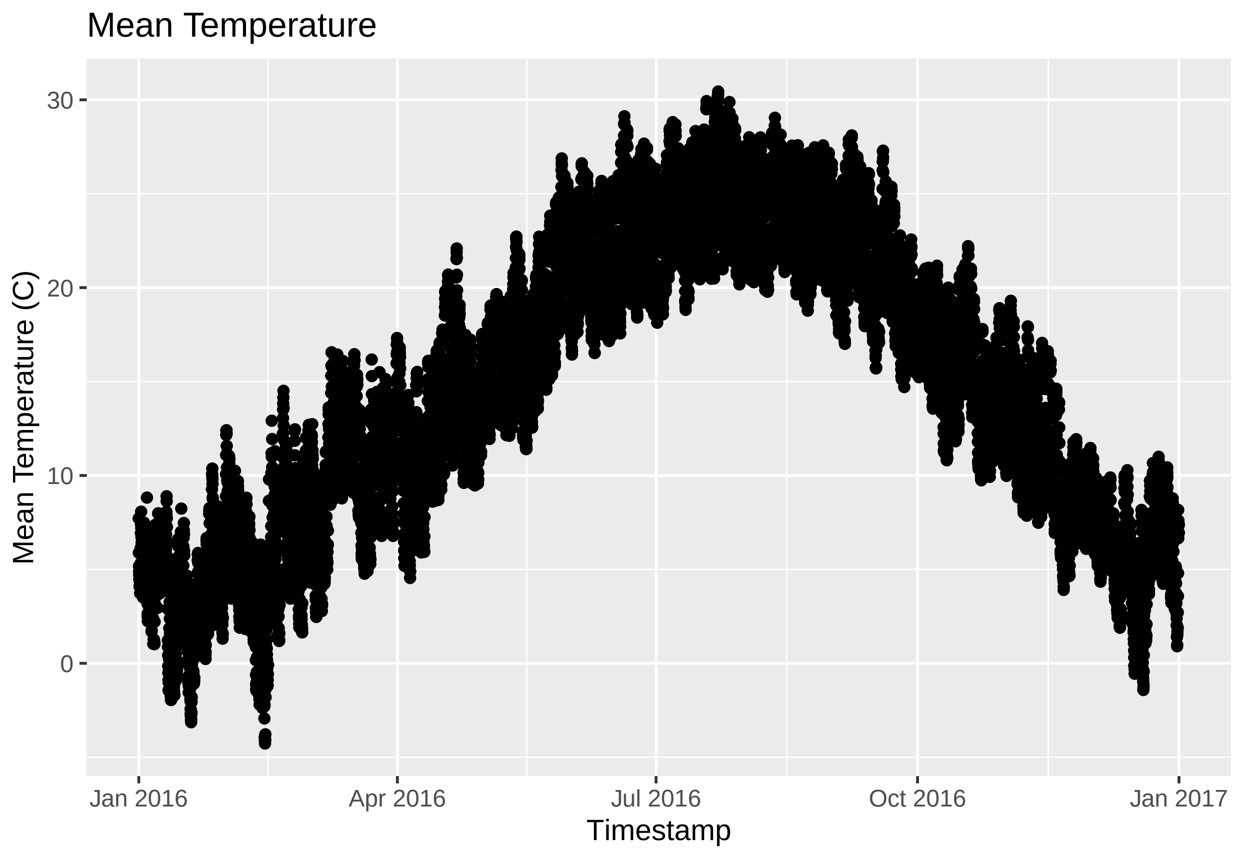


Which leads to a much better plot with the outlier now removed.

We can then make a similar plot using the weather data (using the `air_temperature` column of the `weather` dataframe):

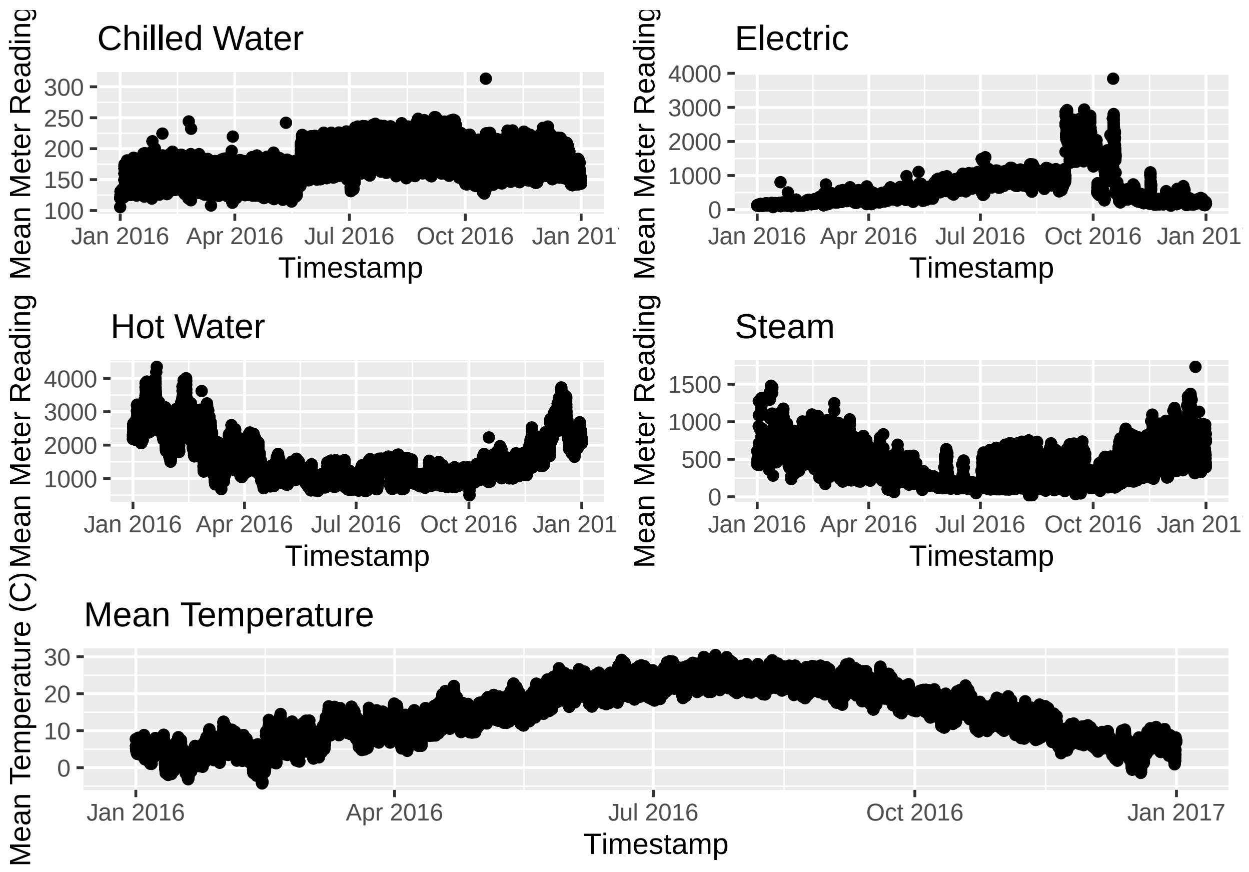
```r
weatherPlot = weather %>%
  group_by(timestamp) %>%
  summarise(temp = mean(air_temperature)) %>%
  ggplot() + geom_point(aes(timestamp, temp)) + ggtitle("Mean Temperature") +
    xlab("Timestamp") + ylab("Mean Temperature (C)")

weatherPlot
```



And finally we can put these plots together using the function `grid.arrange` from the `gridExtra` package we loaded earlier.

```r
grid.arrange(meanMeterReadingPlot(0), meanMeterReadingPlot(1),
             meanMeterReadingPlot(2), meanMeterReadingPlot(3), weatherPlot,
             layout_matrix = rbind(c(1, 2),
                                   c(3, 4),
                                   c(5,5))
             )
```



Pivoting

One further example we'll quickly consider is using `tidyr` to reshape some data. Let's use the `billboard` dataset which holds the Billboard Top 100 songs from each week in 2000.

```
data("billboard")
billboard
```

```
## # A tibble: 317 x 79
##    artist track date.ent~1   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8   wk9
##    <chr>  <chr> <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 2 Pac  Baby~ 2000-02-26    87    82    72    77    87    94    99    NA    NA
##  2 2Ge+h~ The ~ 2000-09-02    91    87    92    NA    NA    NA    NA    NA    NA
##  3 3 Doo~ Kryp~ 2000-04-08    81    70    68    67    66    57    54    53    51
##  4 3 Doo~ Loser 2000-10-21    76    76    72    69    67    65    55    59    62
##  5 504 B~ Wobb~ 2000-04-15    57    34    25    17    17    31    36    49    53
##  6 98^0   Give~ 2000-08-19    51    39    34    26    26    19     2     2     3
##  7 A*Tee~ Danc~ 2000-07-08    97    97    96    95   100    NA    NA    NA    NA
##  8 Aaliy~ I Do~ 2000-01-29    84    62    51    41    38    35    35    38    38
##  9 Aaliy~ Try ~ 2000-03-18    59    53    38    28    21    18    16    14    12
## 10 Adams~ Open~ 2000-08-26    76    76    74    69    68    67    61    58    57
## # ... with 307 more rows, 67 more variables: wk10 <dbl>, wk11 <dbl>,
## #   wk12 <dbl>, wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>,
## #   wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>,
## #   wk24 <dbl>, wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>,
## #   wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>,
## #   wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>,
## #   wk42 <dbl>, wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, ...
```

The `pivot_longer` function increases the number of rows and decrease the number of columns in a dataframe/tibble. To see this, we can use it to create individual rows for each song-week combination. To do this we provide `pivot_longer` with the first argument `-c(artist, track, date.entered)` to indicate that we want to keep these columns and place the values of the other columns (`week1`, `week2`, etc.) into a column with a name specified by the argument `values_to`. The columns we're removing (`week1`, `week2`, etc.) then move into a new column with a name specified by the argument `names_to`.

```
longBillboard = billboard %>% pivot_longer(-c(artist, track, date.entered),
                           names_to="week", values_to="place")
longBillboard
```

```
## # A tibble: 24,092 x 5
##    artist track                   date.entered week  place
##    <chr>  <chr>                   <date>       <chr> <dbl>
##  1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
##  2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
##  3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
##  4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
##  5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
##  6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
##  7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7      99
##  8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8      NA
##  9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9      NA
## 10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10     NA
## # ... with 24,082 more rows
```

The function `pivot_wider` works in the opposite way by increases the number of columns and decreasing the number of rows. To get back to the original dataset from `longBillboard` using `pivot_wider` we would run the command given below:

```
longBillboard %>% pivot_wider(names_from = "week", values_from = "place")
```

```
## # A tibble: 317 x 79
##    artist track date.ent~1   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8   wk9
##    <chr>  <chr> <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 2 Pac  Baby~ 2000-02-26    87    82    72    77    87    94    99    NA    NA
##  2 2Ge+h~ The ~ 2000-09-02    91    87    92    NA    NA    NA    NA    NA    NA
##  3 3 Doo~ Kryp~ 2000-04-08    81    70    68    67    66    57    54    53    51
##  4 3 Doo~ Loser 2000-10-21    76    76    72    69    67    65    55    59    62
##  5 504 B~ Wobb~ 2000-04-15    57    34    25    17    17    31    36    49    53
##  6 98^0   Give~ 2000-08-19    51    39    34    26    26    19     2     2     3
##  7 A*Tee~ Danc~ 2000-07-08    97    97    96    95   100    NA    NA    NA    NA
##  8 Aaliy~ I Do~ 2000-01-29    84    62    51    41    38    35    35    38    38
##  9 Aaliy~ Try ~ 2000-03-18    59    53    38    28    21    18    16    14    12
## 10 Adams~ Open~ 2000-08-26    76    76    74    69    68    67    61    58    57
## # ... with 307 more rows, 67 more variables: wk10 <dbl>, wk11 <dbl>,
## #   wk12 <dbl>, wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>,
## #   wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>,
## #   wk24 <dbl>, wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>,
## #   wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>,
## #   wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>,
## #   wk42 <dbl>, wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, ...
```

This turns the entries in the column specified by `names_from` into individual columns that take values from the corresponding entry in the column specified by `values_from`.