Plotting Change Through Time

Author: Cole Brookson **Date:** 20 July 2022

It is sometimes the case that the easiest way to achieve our goal is, rather than forcing all our data into one large, unwieldy dataframe, we can simply make multiple dataframes and use those multiple dataframes in multiple `geom_`s to make our plot look exactly like we want it. There are myriad reasons why this may be the case, but an easy example is when you have data that span an equivalent time series or range, but have different temporal resolution or coverage. To illustrate this, we'll use an example on temperature data in Alaska.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

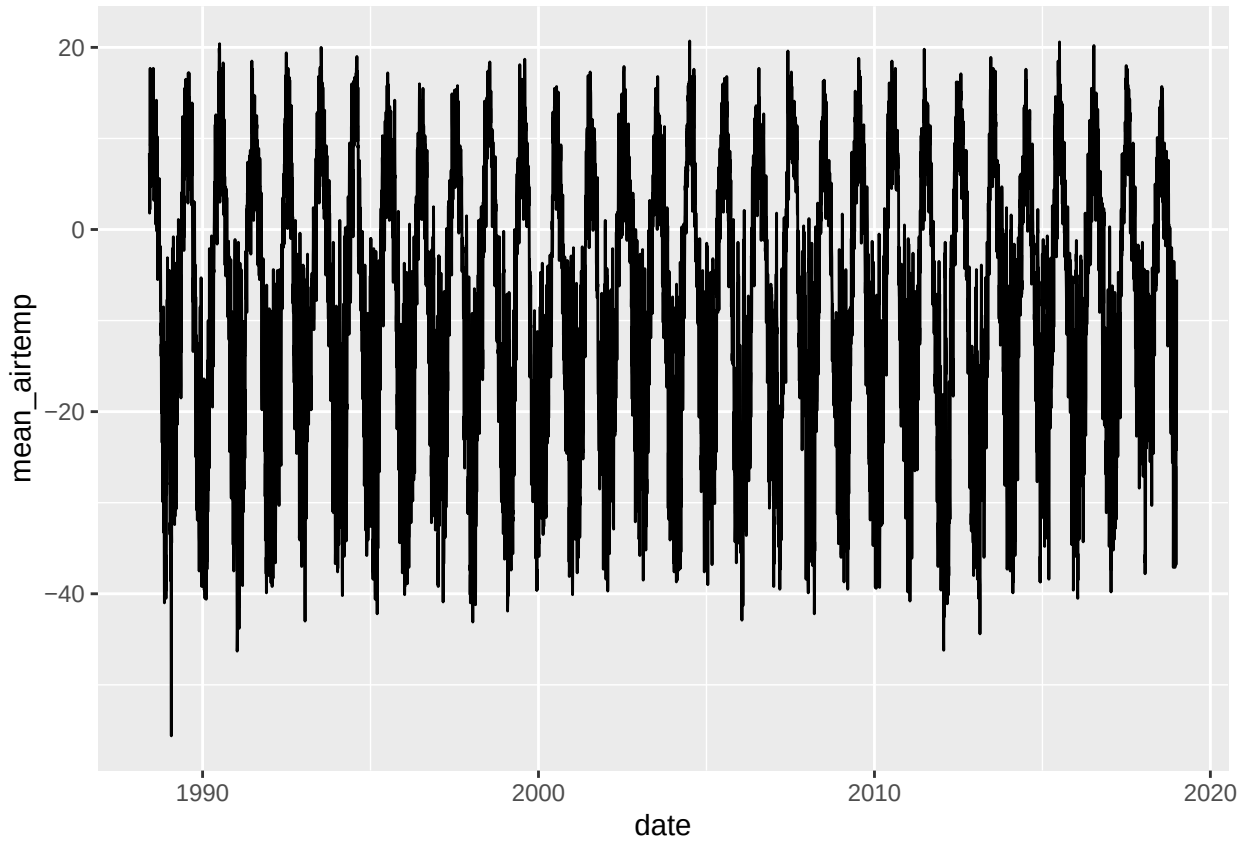
```
library(lterdatasampler)

df = lterdatasampler::arc_weather
```

Timeseries Plots (Line Plots)

Line plots, also referred to as timeseries plots colloquially, are plots that have the focal variable of interest on the y-axis, and some time variable on the x-axis. To follow our (LINK TO ITERATIVE PLOTTING) iterative plotting method, we'll start one of these plots and build up.

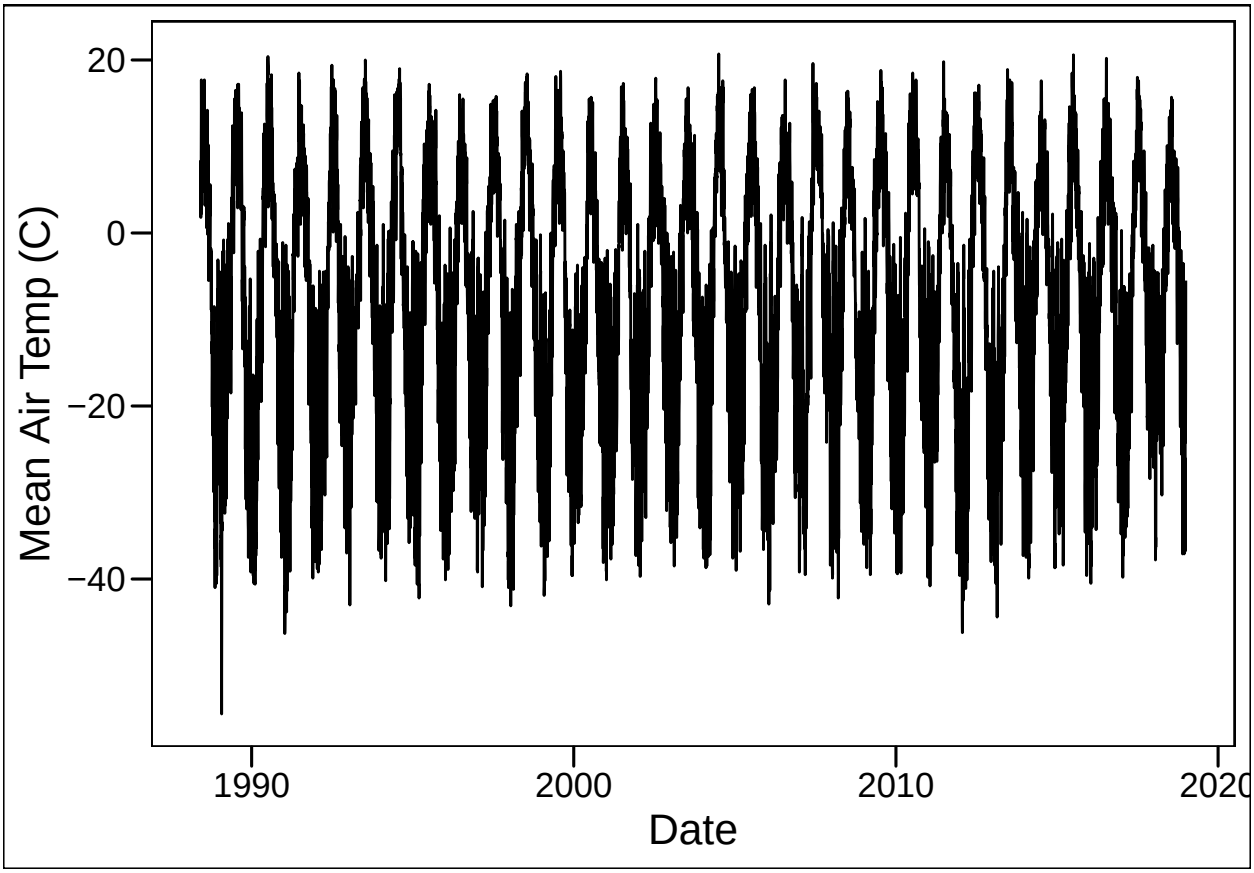
To start with just the most basic elements (data call, aesthetic mappings, and geom function), we can plot mean air temperature through time:

```
ggplot() +
  geom_line(data = df,
            mapping = aes(x = date, y = mean_airtemp))
```



Okay! Very ugly. Perhaps we can extract some more useful information out of this. But first, let's take care of the easy part and add in the mandatory components (labels, a theme) and then work on getting it more useful.

```
# load the library that has theme_base() in it
library(ggthemes)

ggplot() +
  geom_line(data = df,
            mapping = aes(x = date, y = mean_airtemp)) +
  # add in the theme
  ggthemes::theme_base() +
  # add labels +
  labs(x = "Date", y = "Mean Air Temp (C)")
```



So we can see there's a consistent oscilating pattern (which definitely makes sense), but it's still hard to pull out any interesting patterns. What if we plotted instead of just these raw values, the monthly mean for each month in our dataframe. This brings up some handy data manipulation tasks as well.

Prepping the Data

So to do this, there are a number of approaches we could take. I will choose to perform this task by making a new, separate dataframe with the monthly mean information as this will make life much easier.

The first step is to extract the month and year as separate columns (LINK TO DATES PIECE):

```
df = df %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    month = lubridate::month(date),
    year = lubridate::year(date)
  )
df
```

```
## # A tibble: 11,171 x 7
## # Rowwise:
##    date       station       mean_airtemp daily_precip mean_windspeed month  year
##    <date>     <chr>                <dbl>        <dbl>          <dbl> <dbl> <dbl>
##  1 1988-06-01 Toolik Field~          8.4          0             NA       6  1988
##  2 1988-06-02 Toolik Field~          6            0             NA       6  1988
##  3 1988-06-03 Toolik Field~          5.8          0             NA       6  1988
##  4 1988-06-04 Toolik Field~          1.8          0             NA       6  1988
##  5 1988-06-05 Toolik Field~          6.8          2.5           NA       6  1988
```

```
##  6 1988-06-06 Toolik Field~          5.2          0             NA       6  1988
##  7 1988-06-07 Toolik Field~          2.2          7.6           NA       6  1988
##  8 1988-06-08 Toolik Field~          9.4          0             NA       6  1988
##  9 1988-06-09 Toolik Field~         13.1          0             NA       6  1988
## 10 1988-06-10 Toolik Field~         17.7          0              3.9     6  1988
## # ... with 11,161 more rows
```

Now we need to create a new dataframe that has only the `mean_airtemp`, the `month`, and `year`:

```
df_monthly = df %>%
  dplyr::select(year, month, mean_airtemp)
```

Now, we'll use our `group_by()` and `summarize()` pair to get mean average monthly air temperatures:

```
df_monthly = df_monthly %>%
  dplyr::group_by(year, month) %>%
  dplyr::summarize(
    month_airtemp = mean(mean_airtemp, na.rm = TRUE)
  )
```

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

```
df_monthly
```

```
## # A tibble: 367 x 3
## # Groups:   year [31]
##     year month month_airtemp
##    <dbl> <dbl>         <dbl>
##  1  1988     6          9.61
##  2  1988     7         12.0
##  3  1988     8          6.96
##  4  1988     9         -0.26
##  5  1988    10        -16.4
##  6  1988    11        -29.2
##  7  1988    12        -17.1
##  8  1989     1        -29.9
##  9  1989     2         -8.06
## 10  1989     3        -19.4
## # ... with 357 more rows
```

Great, this is exactly what we wanted. Now, to make our plotting easier, it would be best if we were asking `ggplot` to put the same data type on the x-axis, that is, two `date` values. So, what we'll do here, is inpute a day of the month, and convert our month and year columns into a date.

Since we are working with monthly means, let's say for simplicity sake we actually just want each faux "day" to be the 15th since that's around the middle of each month:

```
df_monthly = df_monthly %>%
  dplyr::mutate(
    day = 15
  )
df_monthly
```

```
## # A tibble: 367 x 4
## # Groups:   year [31]
##     year month month_airtemp   day
##    <dbl> <dbl>         <dbl> <dbl>
##  1  1988     6          9.61    15
```

```
##  2  1988     7         12.0     15
##  3  1988     8          6.96    15
##  4  1988     9         -0.26    15
##  5  1988    10        -16.4     15
##  6  1988    11        -29.2     15
##  7  1988    12        -17.1     15
##  8  1989     1        -29.9     15
##  9  1989     2         -8.06    15
## 10  1989     3        -19.4     15
## # ... with 357 more rows
```

Great! And now we convert into a single date column:

```
df_monthly = df_monthly %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    date = lubridate::make_date(year, month, day)
  )
df_monthly
```

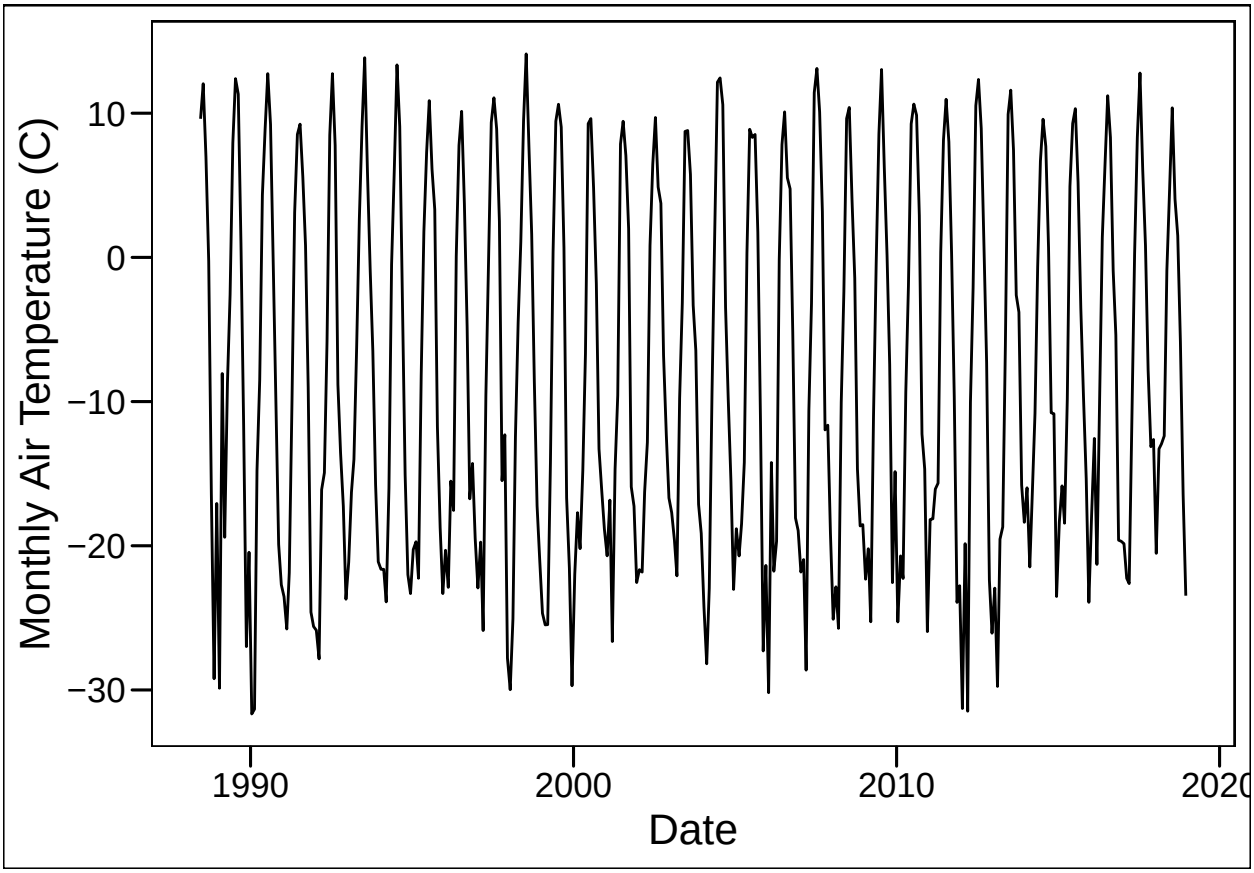
```
## # A tibble: 367 x 5
## # Rowwise:  year
##     year month month_airtemp   day date
##    <dbl> <dbl>         <dbl> <dbl> <date>
##  1  1988     6          9.61    15 1988-06-15
##  2  1988     7         12.0     15 1988-07-15
##  3  1988     8          6.96    15 1988-08-15
##  4  1988     9         -0.26    15 1988-09-15
##  5  1988    10        -16.4     15 1988-10-15
##  6  1988    11        -29.2     15 1988-11-15
##  7  1988    12        -17.1     15 1988-12-15
##  8  1989     1        -29.9     15 1989-01-15
##  9  1989     2         -8.06    15 1989-02-15
## 10  1989     3        -19.4     15 1989-03-15
## # ... with 357 more rows
```

Perfect. Now back to plotting.

Plotting with Two Dataframes

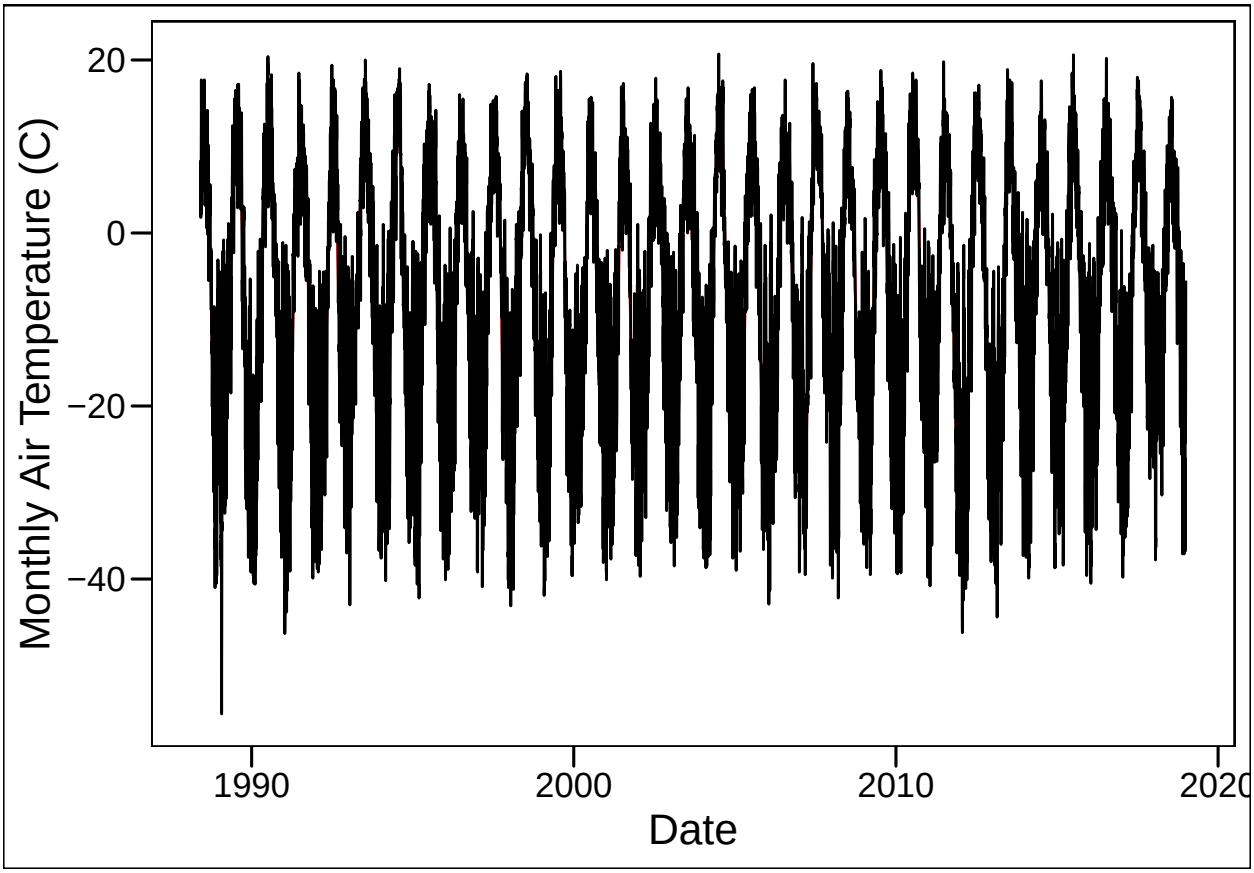
The easiest way to go about this, since we're plotting a variable called `date` from two dataframes is to use two separate calls to geom_line, one for each dataframe, and that will allow us to show both the ones we're interested in (the monthly and daily values):

```
ggplot() +
  # first geom_line() for the monthly values
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp)) +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)")
```



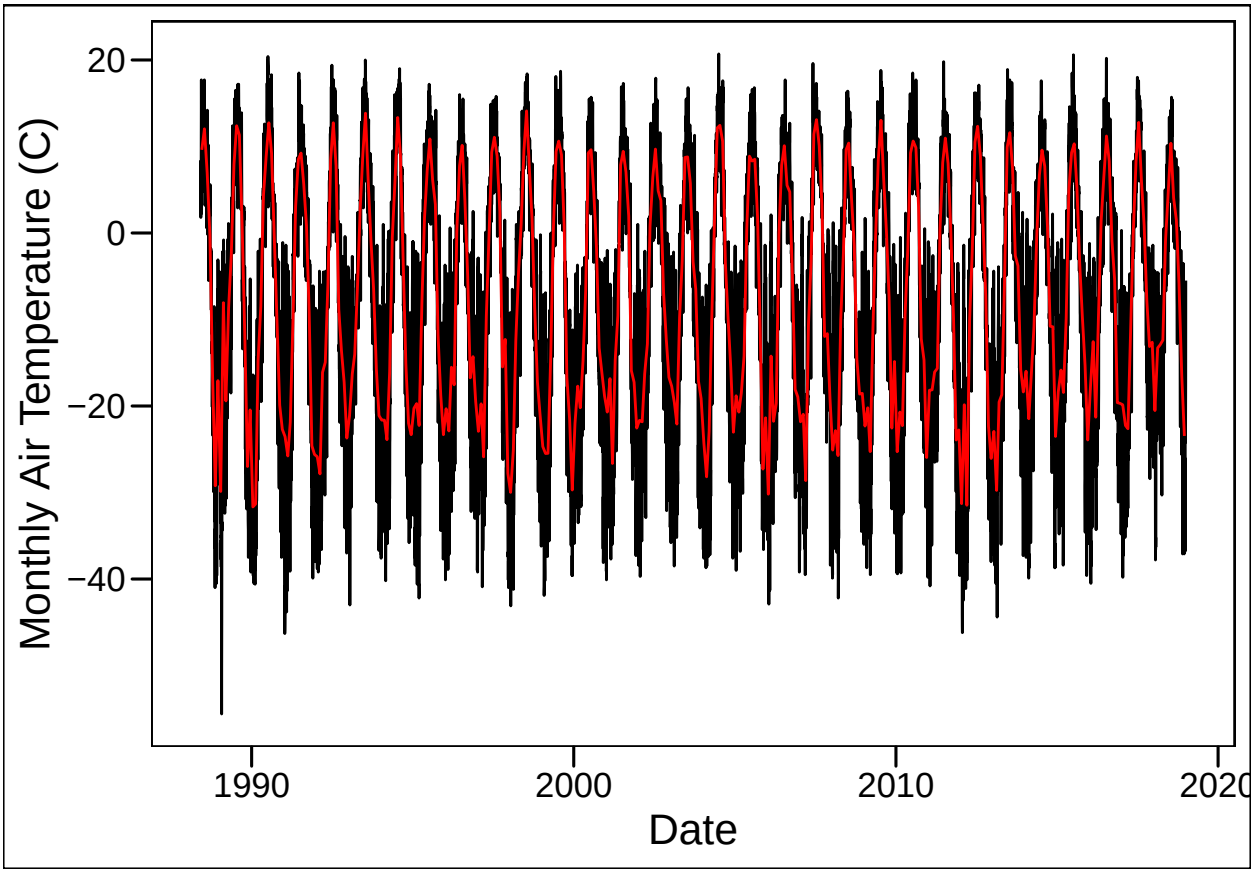
Okay, let's add our second `geom_line()` call. We'll make our monthly values red and our daily values can be black:

```
ggplot() +
  # first geom_line() for the monthly values
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp),
            colour = "red") +
  # second geom_line() for the daily values
  geom_line(data = df, mapping = aes(x = date, y = mean_airtemp)) +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)")
```



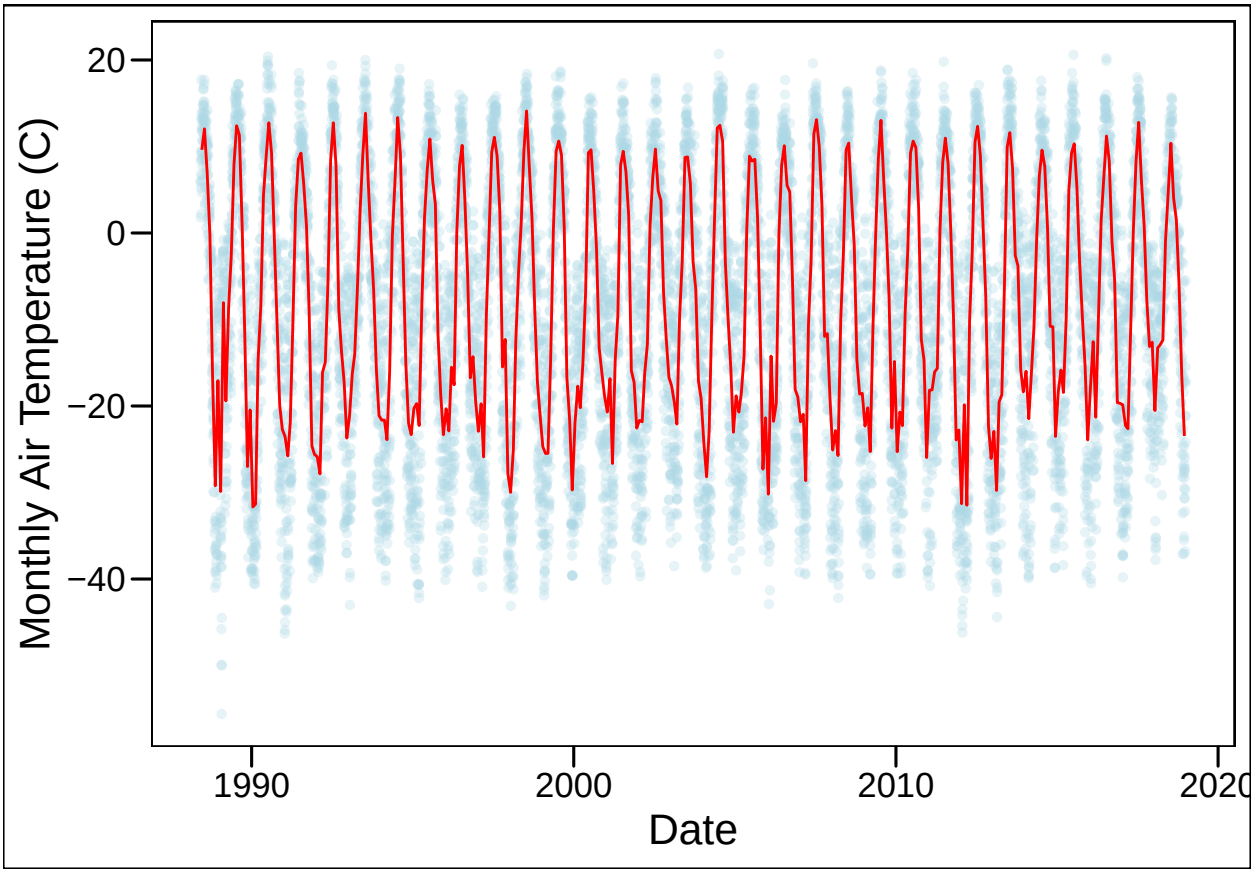
Well that's weird, we can't see anything! How come? Well, likely because the black lines of the daily temperatures are significantly more data-rich, and in our plotting, we placed our monthly `geom_line()` first, which means it will be plotted first, and then the thick black line will go on top. Let's reverse them and see if that helps:

```
ggplot() +
  # switched order - daily first
  geom_line(data = df, mapping = aes(x = date, y = mean_airtemp)) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp),
            colour = "red") +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)")
```



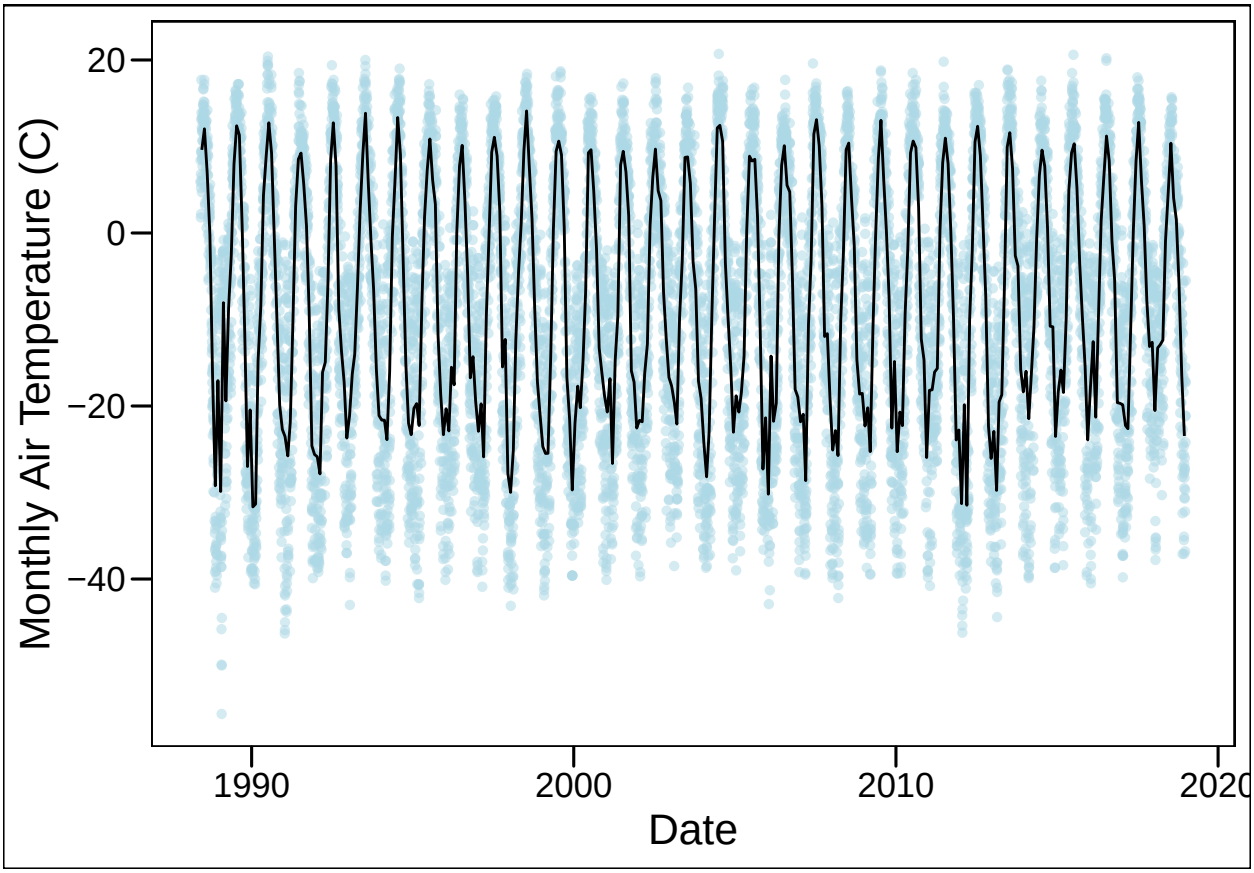
That's better! Still not super cute though. **NOTE: It is often hard to make two sets of timeseries following the same pattern look good. My recommendation here would be turn one (likely the daily measurements) into points, and keep the trend line. So let's try that, we'll make the points somewhat transparent, and keep them relatively small:

```
ggplot() +
  # switched order - daily first
  geom_point(data = df, mapping = aes(x = date, y = mean_airtemp),
             shape = 16, colour = "lightblue", alpha = 0.3) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp),
            colour = "red") +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)")
```



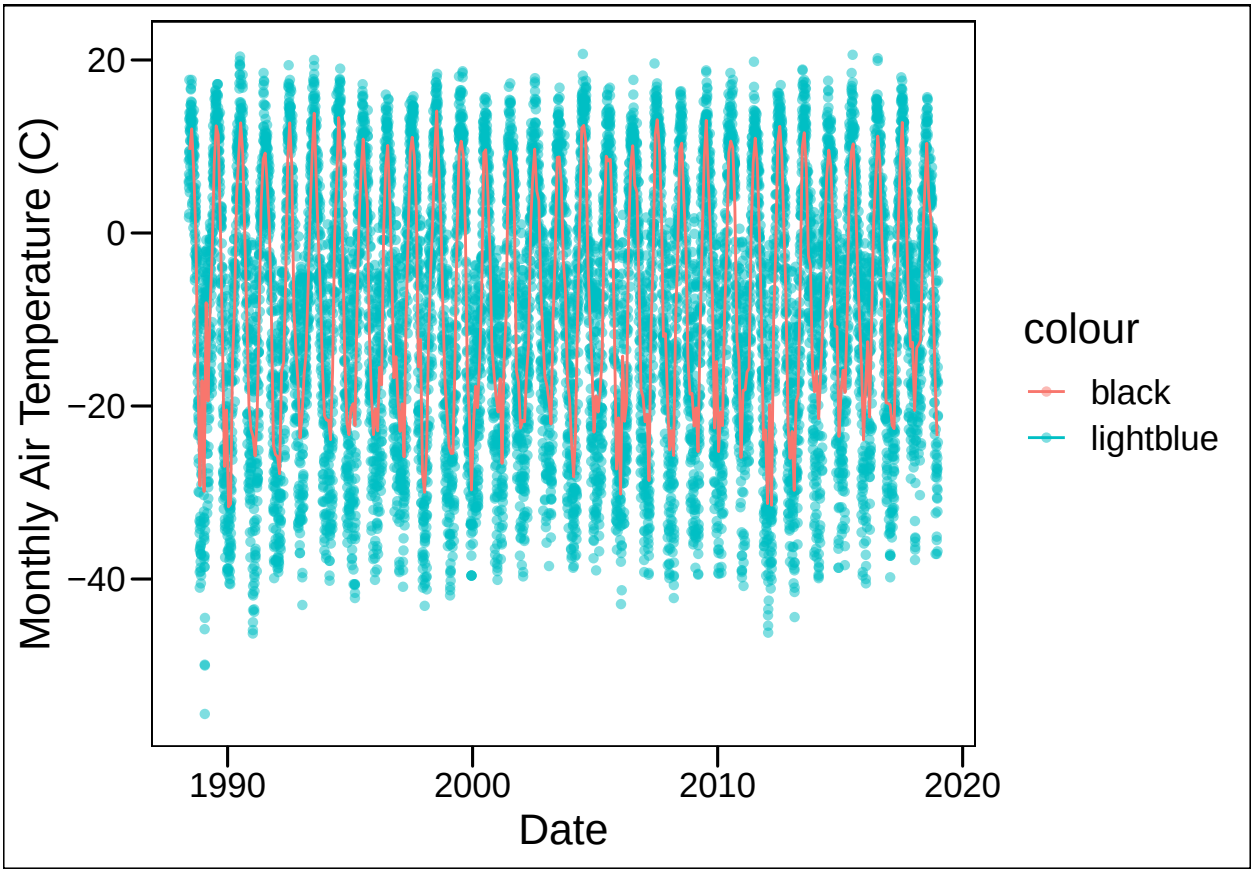
That's somewhat better. The transparency might be a bit too much, so we can change that, also let's change the main line colour to black:

```
ggplot() +
  # switched order - daily first
  geom_point(data = df, mapping = aes(x = date, y = mean_airtemp),
             shape = 16, colour = "lightblue", alpha = 0.5) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp),
            colour = "black") +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)")
```



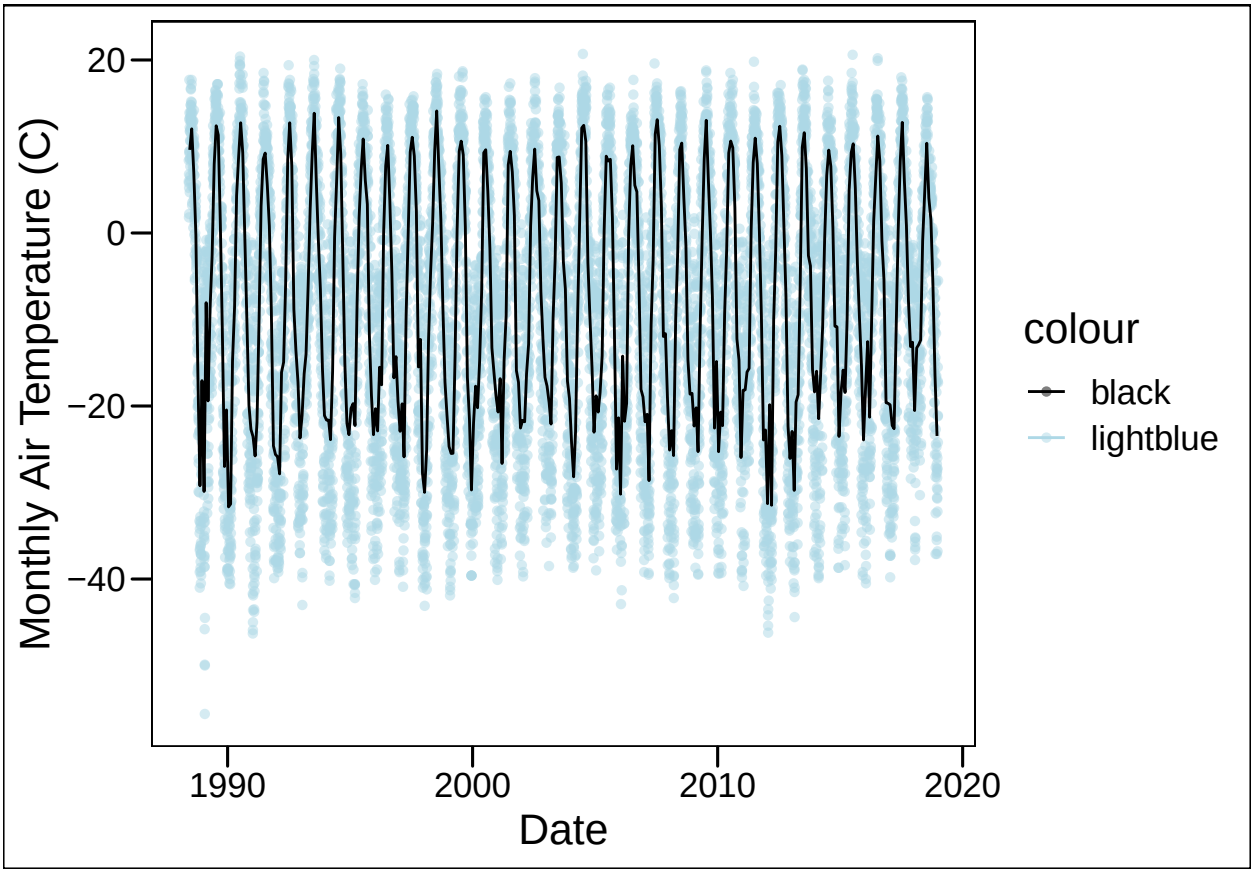
I think that's much more legible. And there we have it! A complete timeseries plot. Something is missing though ... a legend. Why wasn't one generated for us? Well, we used two separate dataframes here, and at no point did we make any differentiation between size/colour/shape etc, within the `aes()` calls. Only if we do that (e.g. we plotted two different lines from the same df and coloured them via some grouping variable) would R know we wanted a legend. Now we could just explain what the difference between the line and the points are either verbally during presentation or in text in a figure caption, but it is always true that having a legend is the easiest way to show people reading what they are looking at. Luckily, editing or adding legends is quite easy. We are using two `geoms` here so we want two legends. *We can force a legend by calling our `colour` argument inside the `aes()` section:*

```
ggplot() +
  # switched order - daily first
  geom_point(data = df, mapping = aes(x = date, y = mean_airtemp,
                                      colour = "lightblue"), shape = 16,
                                      alpha = 0.5) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp,
                                             colour = "black")) +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)") +
  guides(
    colour = guide_legend()
  )
```



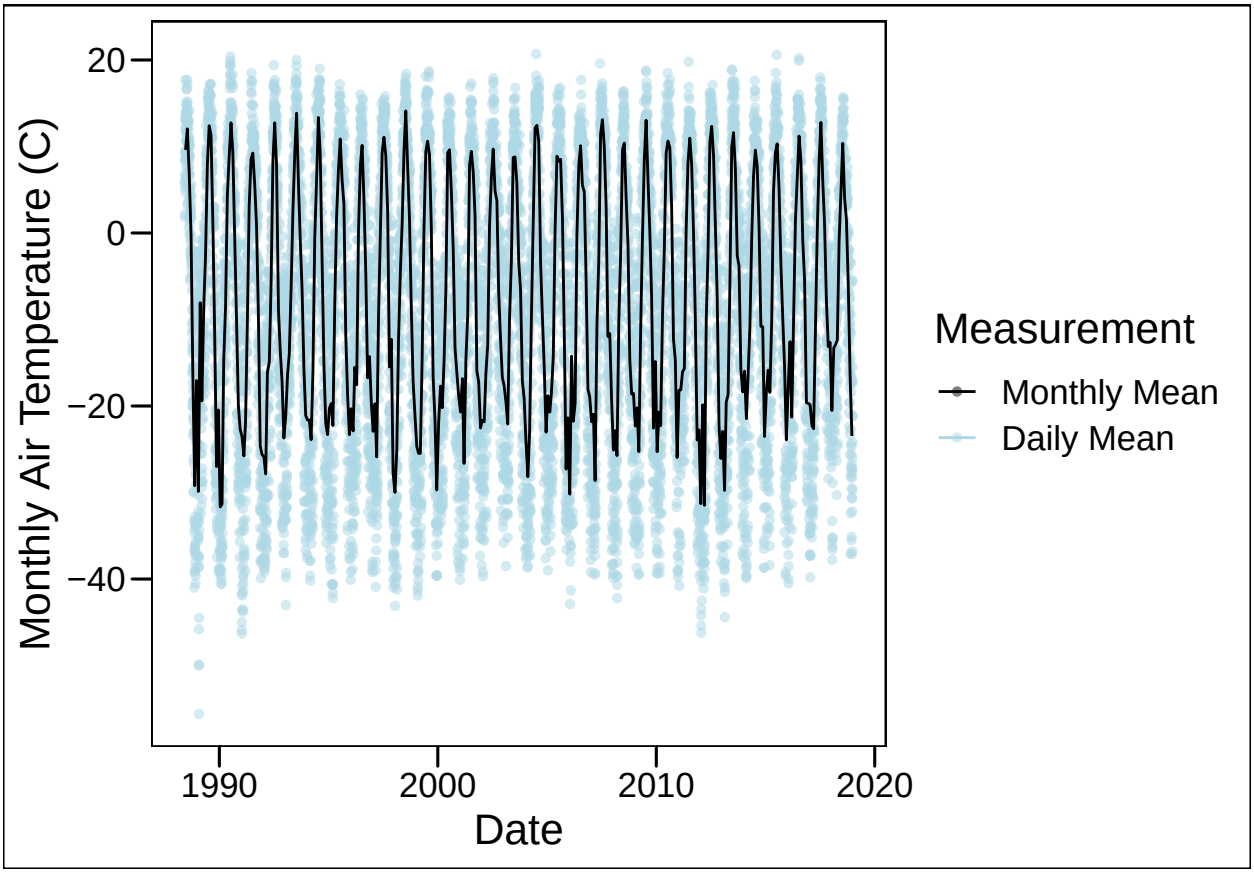
Ok, so we have a legend but it's not at all what we want colour-wise, and the labels are all wrong. Fear not!!! So first, we will have to turn to our old friend `scale_colour_*` and for this use case, we want `scale_colour_identity()` as it will force ggplot to recognize the colour names when they are inside `aes()`:

```
ggplot() +
  # switched order - daily first
  geom_point(data = df, mapping = aes(x = date, y = mean_airtemp,
                                      colour = "lightblue"), shape = 16,
                                      alpha = 0.5) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp,
                                             colour = "black")) +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)") +
  guides(
    colour = guide_legend()
  ) +
  # add in colour scale
  scale_colour_identity(guide = "legend")
```



Okay, that bit is accomplished. We can now add in components to make the legend look as we want. Remember that all those legend arguments will go in the `scale_*` that we're working with *to make the legend:*

```
ggplot() +
  # switched order - daily first
  geom_point(data = df, mapping = aes(x = date, y = mean_airtemp,
                                      colour = "lightblue"), shape = 16,
                                      alpha = 0.5) +
  # monthly second
  geom_line(data = df_monthly, mapping = aes(x = date, y = month_airtemp,
                                             colour = "black")) +
  ggthemes::theme_base() +
  labs(x = "Date", y = "Monthly Air Temperature (C)") +
  guides(
    colour = guide_legend()
  ) +
  # add in colour scale
  scale_colour_identity(guide = "legend",
                        name = "Measurement", # the title of the legend
                        breaks = c("black", "lightblue"), # the colours
                        labels = c("Monthly Mean", "Daily Mean") # legend labels
                        )
```



And now our plot is done and ready to display in a project/presentation.