Dodoma Time Series Analysis

Demonstration of tsibbles, fabletools and feasts

Alex Thomson - Stats4SD

04/03/2020

```
#libraries
library(fabletools)
library(tidyverse)
library(datasets)
library(zoo)
library(tsibble)
library(tseries)
library(lubridate)
library(fpp3)
library(tsibbledata)
library(seasonal)
library(feasts)
library(GGally)
library(grid)
library(gridExtra)
library(forecast)
```

Dodoma dataset

Introduction

This short report summarises some usages of the wide array of tools one can use for time series analysis of data in R. Particularly focusing on the introduction of ts/tsibble objects and functions from the fabletools and feasts packages. The data utilised for this report is the Dodoma data which includes observations of rainfall, maximum temperature, minimum temperature and sunshine hours since 1935 recorded daily. This analysis has kept the data in a daily format though it shall be demonstrated that it is straightforward to change this into weekly, monthly or annual data. This analysis shall be limited to data between 2011 and 2014 to simplify the modelling demonstration and allow for clearer presentation of graphs.

Data provided

Below is an example subset of the Dodoma dataset (the first 10 days of 2011)

Table 1: Dodoma dataset (1st - 10th January)

year	month	day	date	month_abbr	doy_366	rain	tmax	tmin	sunh
2011	1	1	2011-01-01	Jan	1	0.0	29.9	18.3	11.3
2011	1	2	2011-01-02	Jan	2	0.0	30.5	18.4	11.5

year	month	day	date	month_abbr	doy_366	rain	tmax	tmin	sunh
2011	1	3	2011-01-03	Jan	3	0.0	31.4	19.1	9.5
2011	1	4	2011-01-04	Jan	4	0.0	31.8	19.1	11.8
2011	1	5	2011-01-05	Jan	5	0.0	31.2	20.1	10.2
2011	1	6	2011-01-06	Jan	6	26.2	31.4	17.8	9.2
2011	1	7	2011-01-07	Jan	7	3.5	27.2	17.8	4.2
2011	1	8	2011-01-08	Jan	8	9.7	26.5	16.6	2.9
2011	1	9	2011-01-09	Jan	9	7.7	30.0	18.0	6.6
2011	1	10	2011-01-10	Jan	10	0.0	27.6	19.3	3.0

Converting data frame - tibble to a tsibble

The code below demonstrates how to turn the dodoma dataset into a tsibble object. First a new time based column is made called "date" by turning the character column "date" (from the original data) into a date format using the lubridate function `as_date`. Other functions are available depending on the unit of time you wish to use. (e.g. using `yearQuarter` for quarterly data or `yearMonth` for monthly data)

A key advantage of these functions are that they can be used much in the same way as the tidyverse family of packages (i.e dplyr, plyr, ggplot) in that they can easily follow on from one another using pipes `(%>%)` or additions `(+)` if you are creating graphs.

Following the addittion of this new variable, the other time based variables are removed since they are not necessary for a daily analysis as all date based information required is contained in the date variable.

The `as_tsibble` function is used to transform this tibble into a tsibble (a time series tidy table). "`index`" is used to tell the function what is the time variable to index the observations by. Not shown here is that the "`key`" argument can be used to distinguish different time series (i.e. by different levels of a factor variable most likely). While not used here in this climatic data, there are possibilities for this ability. For instance if you had data from many areas you could use this to differentiate between the different local time series, one for each place. Essentially this would group time series observations together by factor variables.

In the console output you can see that the data frame is now stored as a tsibble object with 1096 rows by 5 columns, the "[1D]" in the output states that the object has data for 1 day each i.e it is daily data. If this was quarterly data the output would read "[1Q]" instead.

```
#transform into tsibble
dodoma_daily<-dodoma_2011%>%
  mutate(date = as_date(date))%>%
  select(-year, -month, -day, -month_abbr, -doy_366)%>%
  as_tsibble(index = date)%>%
  arrange(date)

dodoma_daily
```

```
## # A tsibble: 1,096 x 5 [1D]
##    date        rain  tmax  tmin  sunh
##    <date>     <dbl> <dbl> <dbl> <dbl>
##  1 2011-01-01   0    29.9  18.3  11.3
##  2 2011-01-02   0    30.5  18.4  11.5
##  3 2011-01-03   0    31.4  19.1   9.5
##  4 2011-01-04   0    31.8  19.1  11.8
##  5 2011-01-05   0    31.2  20.1  10.2
##  6 2011-01-06  26.2  31.4  17.8   9.2
##  7 2011-01-07   3.5  27.2  17.8   4.2
```

```
##  8 2011-01-08   9.7  26.5  16.6   2.9
##  9 2011-01-09   7.7  30    18     6.6
## 10 2011-01-10   0    27.6  19.3   3
## # ... with 1,086 more rows
```

Using dplyr to change daily data into aggregate data

As previously mentioned, because the tsibble functions work much like dplyr they can be easily piped for easy data transformation. This includes converting daily data into monthly data quite simply.

In the example below a new variable called "yearmonth" was created by pasting together the year and month_abbr variables, the data was then grouped by this new variable.

To transform the data such that there was one row per month, some summary statistics were calculated for each climatic measure - the total monthly rainfall, the monthly maximum temperature, the monthly minimum temperature and the average daily number of sunshine hours.

The yearmonth variable was then formatted using the previously mentioned `yearmonth` function before creating a tsibble object as previously done before.

Table 2: Shows the monthly data for 1980 and 1981

```
dodoma_monthly<-dodoma%>%
  filter(year>=1980)%>%
  mutate(yearmonth = paste(year, month_abbr))%>%
  group_by(yearmonth)%>%
  summarise(rain = sum(rain, na.rm = TRUE), tmax = max(tmax),
            tmin = min(tmin), sunh=mean(sunh, na.rm = TRUE))%>%
  mutate(yearmonth = yearmonth(yearmonth))%>%
  as_tsibble(index = yearmonth)

knitr::kable(dodoma_monthly%>%slice(1:24),
             caption = "Monthly dodoma data (1980/1981)")
```

Table 2: Monthly dodoma data (1980/1981)

yearmonth	rain	tmax	tmin	sunh
1980 Jan	116.7	31.8	17.2	6.950000
1980 Feb	26.5	33.1	17.2	9.737931
1980 Mar	161.4	30.5	15.5	7.319355
1980 Apr	193.1	32.1	14.9	7.756667
1980 May	1.2	29.4	15.2	8.283871
1980 Jun	0.0	29.0	10.8	10.566667
1980 Jul	0.0	27.3	11.7	9.145454
1980 Aug	0.0	28.9	12.0	NaN
1980 Sep	0.0	32.5	14.2	9.717241
1980 Oct	1.0	33.2	14.3	9.119355
1980 Nov	7.8	33.6	16.2	9.106667
1980 Dec	87.0	34.0	16.2	6.635484
1981 Jan	26.4	32.6	15.6	8.977419
1981 Feb	108.7	32.9	17.2	8.079167
1981 Mar	144.3	32.9	16.2	7.964516
1981 Apr	43.5	31.8	16.0	6.703333
1981 May	4.3	29.4	14.2	8.580645

yearmonth	rain	tmax	tmin	sunh
1981 Jun	0.0	29.8	11.5	9.830000
1981 Jul	0.0	27.8	11.3	10.166667
1981 Aug	0.0	29.5	12.2	9.906452
1981 Sep	0.2	31.4	14.0	10.193333
1981 Oct	0.0	33.5	15.5	9.806452
1981 Nov	3.2	33.8	16.5	10.426667
1981 Dec	81.3	33.7	17.0	8.687097

Plotting time series

Time series data can easily be plotted like any other data. Each of the four climatic measures have been plotted using ggplot in the figure below. These plots can be observed for visual cues of trends, seasonality and possible cycles.

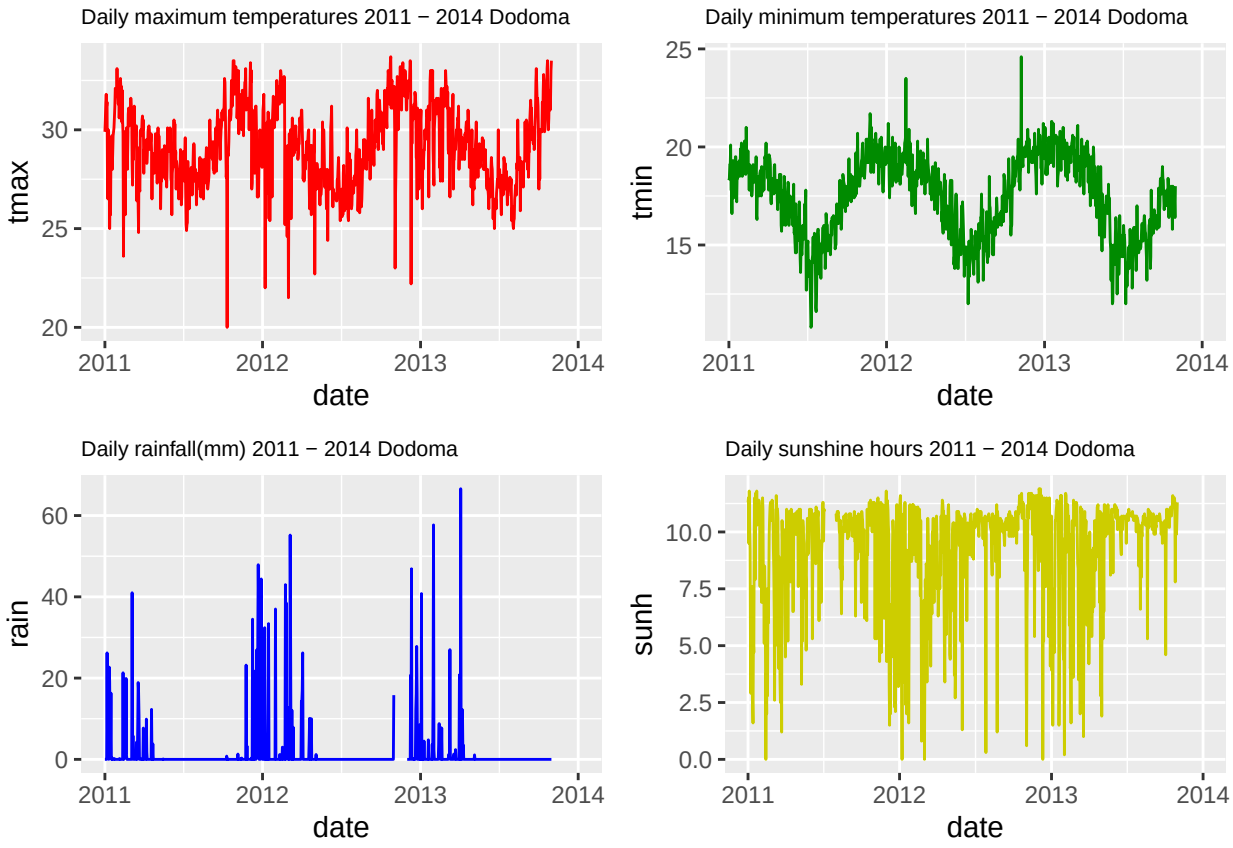
Unsurprisingly given this is climatic data, there is some clear patterns of seasonality in the data. There is a clear pattern in the minimum temperature which peaks in the months of November through to February, dropping to a trough in July before increasing steadily again. This pattern can be seen across the years. Maximum temperatures has a similar pattern though not quite as pronounced with outlying observations more common and not as well defined troughs. There is not a particularly large trend to either of these two time series.

There clear seasonality to the rainfall data as well, rainfall is practically non-existent through the months of May through November (the dry months), with rainfall occurring during December through April. There additionally seems to be an upward trend across the 3 year period.

Sunshine hours is harder to read thought there is some seasonality with values reaching very low in the months of December to March.

```
#plot maximum temperature
P1<-dodoma_daily%>%
  ggplot(aes(x = date, y = tmax))+
  geom_path(colour = "red")+
  ggtitle("Daily maximum temperatures 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot minimum temperature
P2<-dodoma_daily%>%
  ggplot(aes(x = date, y = tmin))+
  geom_path(colour = "green4")+
  ggtitle("Daily minimum temperatures 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot rainfall
P3<-dodoma_daily%>%
  ggplot(aes(x = date, y = rain))+
  geom_path(colour = "blue")+
  ggtitle("Daily rainfall(mm) 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot sunshine hours
P4<-dodoma_daily%>%
  ggplot(aes(x = date, y = sunh))+
  geom_path(colour = "yellow3")+
  ggtitle("Daily sunshine hours 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
```

```
#combine plots
grid.arrange(P1, P2, P3, P4, nrow=2)
```



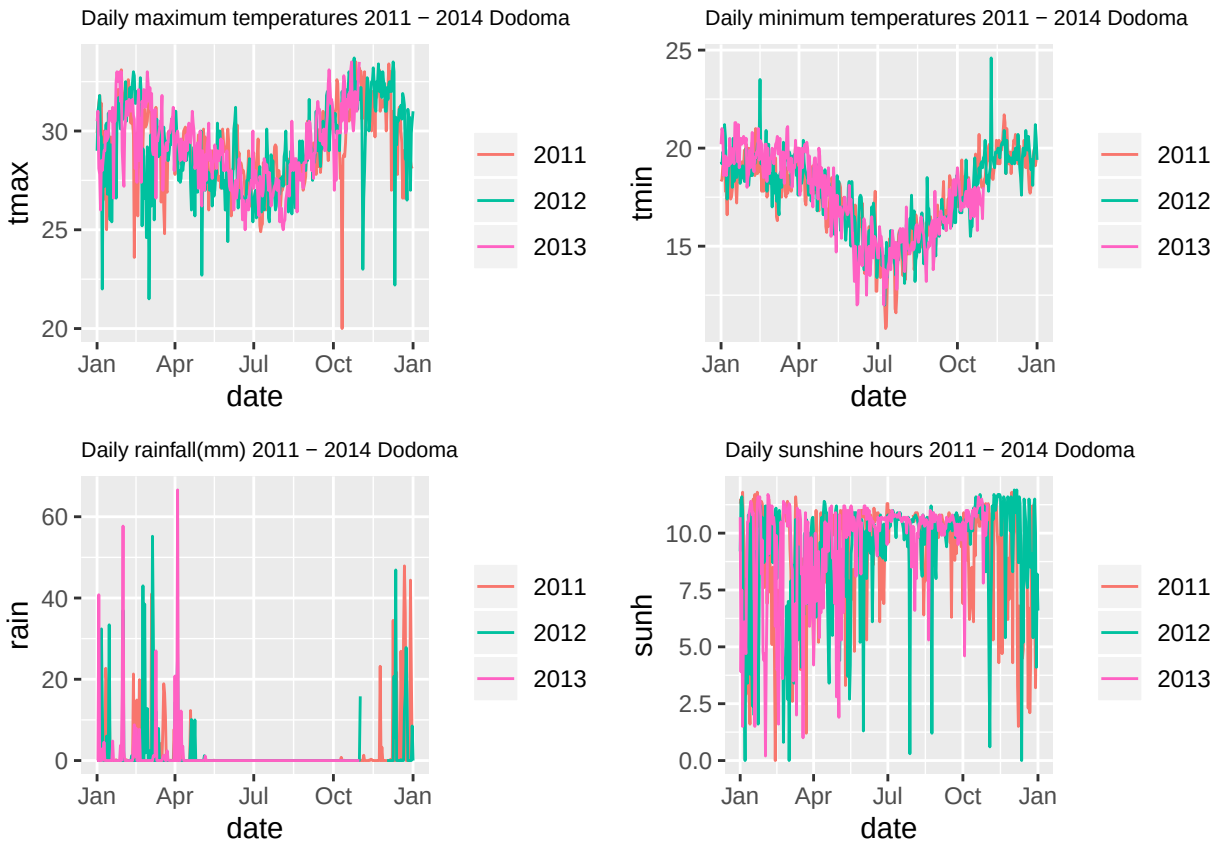
gg_season and gg_subseries

The `gg_season` and `gg_subseries` functions can be used to easily plot time-series data by different "seasons".

For the daily data, we can use `gg_season` to plot the different climatic across the 3 years in the sub setted data. Figure 2 is the seasonal sub-plots of each of the 4 climatic measures. This helps see the seasonal patterns of each measure more clearly as it picks separates out each season and plots them against each other. We can more clearly see how maximum and minimum temperatures peak in the months of December through March and troughs in July as well as how dry the months of June, July, August and September are across all 3 years.

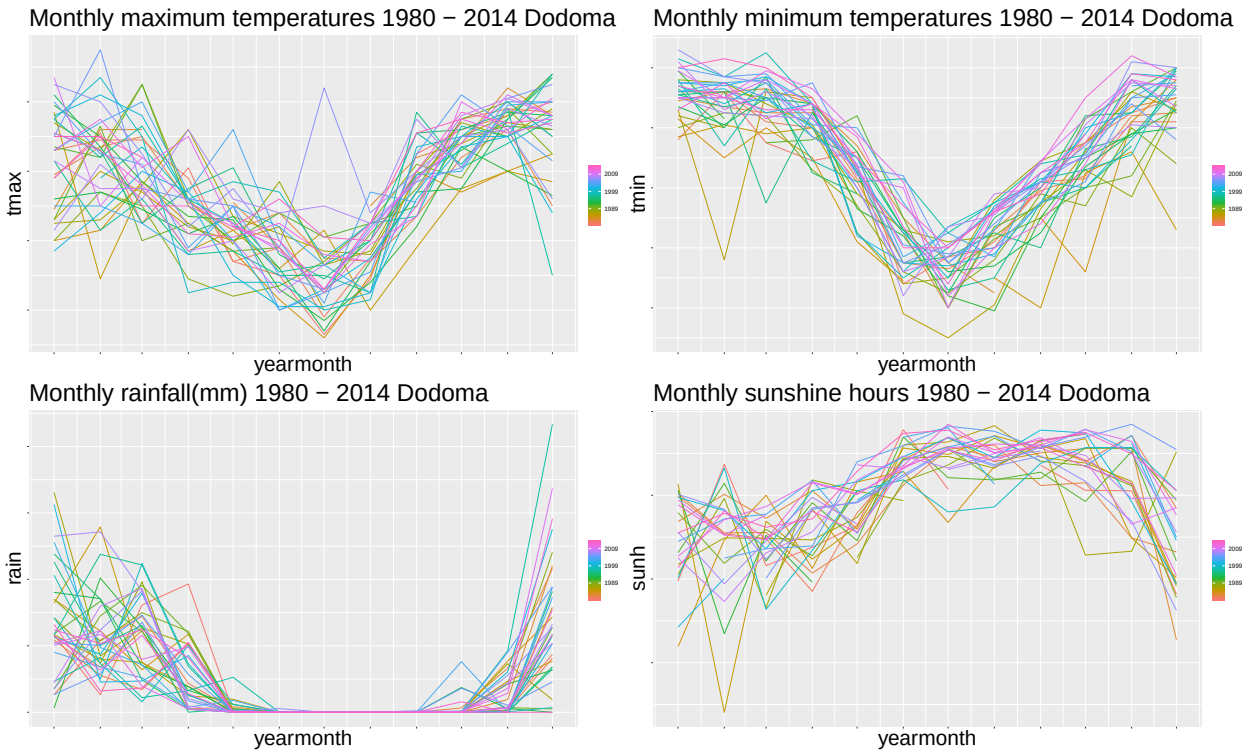
```
#plot maximum temperature
P1<-dodoma_daily%>%
  gg_season(tmax)+
  ggtitle("Daily maximum temperatures 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot minimum temperature
P2<-dodoma_daily%>%
  gg_season(tmin)+
  ggtitle("Daily minimum temperatures 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot rainfall
```

```
P3<-dodoma_daily%>%
  gg_season(rain)+
  ggtitle("Daily rainfall(mm) 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#plot sunshine hours
P4<-dodoma_daily%>%
  gg_season(sunh)+
  ggtitle("Daily sunshine hours 2011 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 8))
#combine plots
grid.arrange(P1, P2, P3, P4, nrow=2)
```



Furthermore, we can plot our monthly data in much the same way. The figures below plot each of the monthly statistics for each year since 1980. Again we can quite clearly see the seasonality of the measures, infarct a pattern to sunshine hours has become much clearer than what we could observe from the daily data. Average daily sunshine hours seems to trough in the late and early months of the year and peak during the middle of the year, during the dry seasons.

It is possible, depending on the data, to observe trends using this function if there is clear differences in the position of lines from different seasons/years. While there is not much in terms of this in these graphs, if the more recent lines were clearly grouped below or above the older lines then this would suggest a decreasing or increasing trend respectively, while seasonality remained stable.

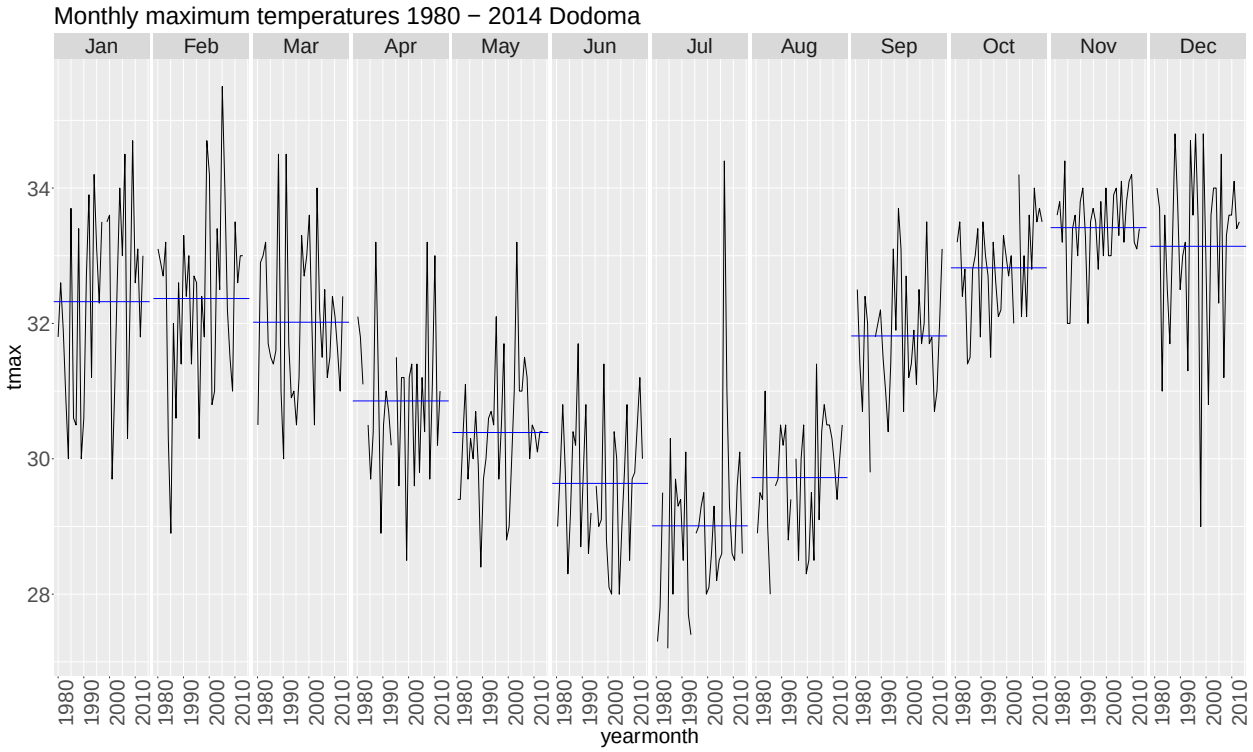


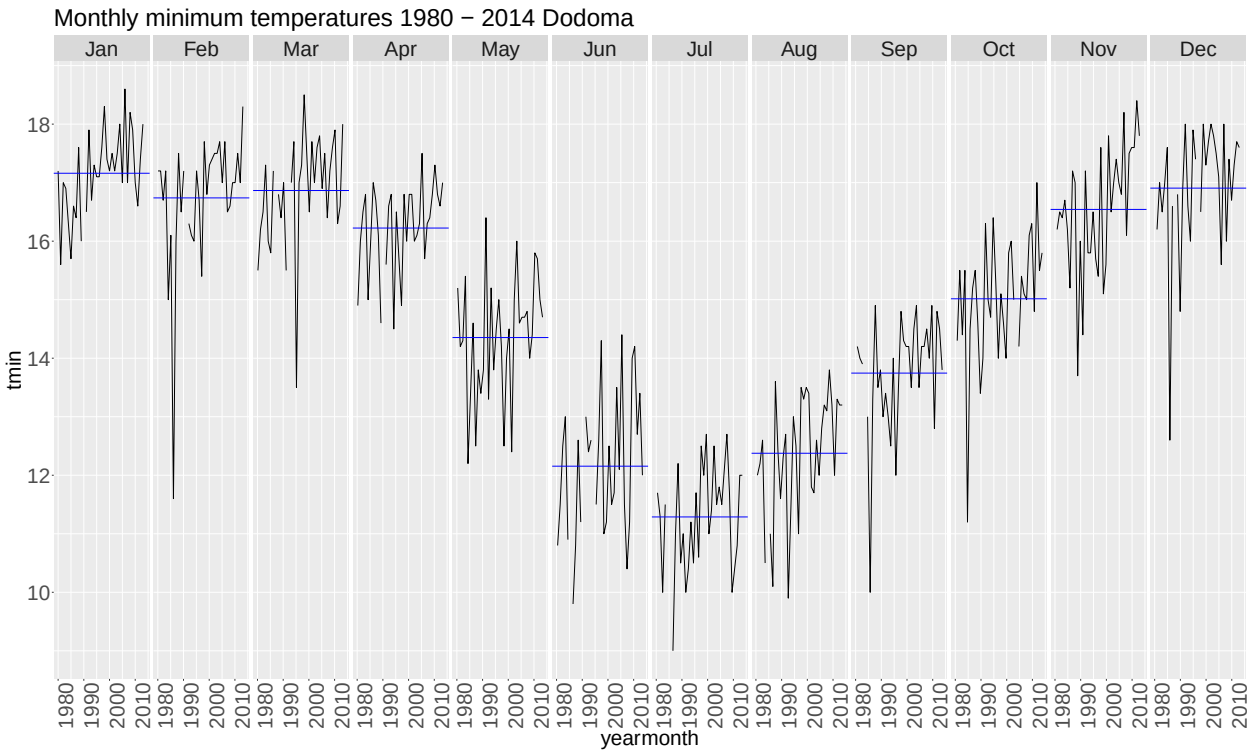
Perhaps a better way to observe this distinction between seasonality and trends would be to use `gg_subseries`, this creates separate graphs for each unit of a season. I.e with monthly data, 12 plots would be made, one for each month with the data from each year plotted, with daily data in a year - this would be 366 plots (slightly unmanageable to say the least).

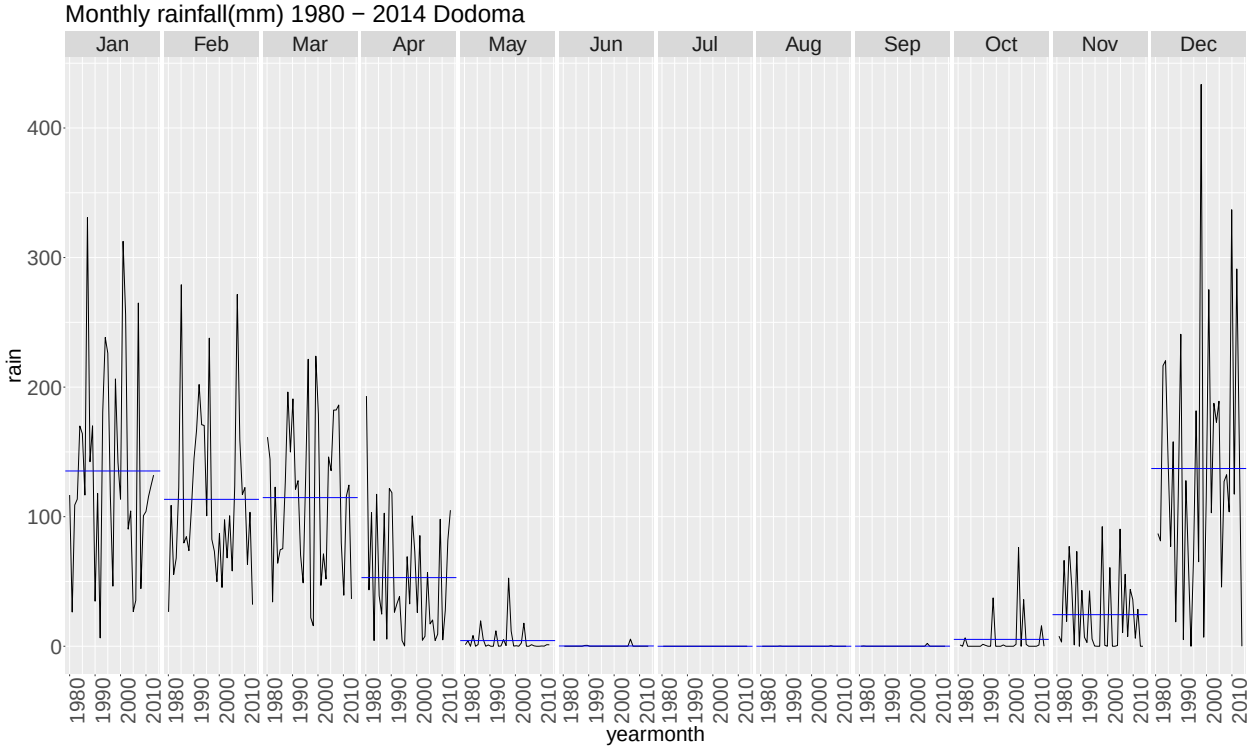
```
#plot maximum temperature
P1<-dodoma_monthly%>%
  gg_subseries(tmax)+
  ggtitle("Monthly maximum temperatures 1980 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 35),
        strip.text.x = element_text(size = 30),
        axis.text = element_text(size = 30),
        axis.title = element_text(size = 30))
#plot minimum temperature
P2<-dodoma_monthly%>%
  gg_subseries(tmin)+
  ggtitle("Monthly minimum temperatures 1980 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 35),
        strip.text.x = element_text(size = 30),
        axis.text = element_text(size = 30),
        axis.title = element_text(size = 30))
#plot rainfall
P3<-dodoma_monthly%>%
  gg_subseries(rain)+
  ggtitle("Monthly rainfall(mm) 1980 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 35),
        strip.text.x = element_text(size = 30),
        axis.text = element_text(size = 30),
        axis.title = element_text(size = 30))
#plot sunshine hours
```

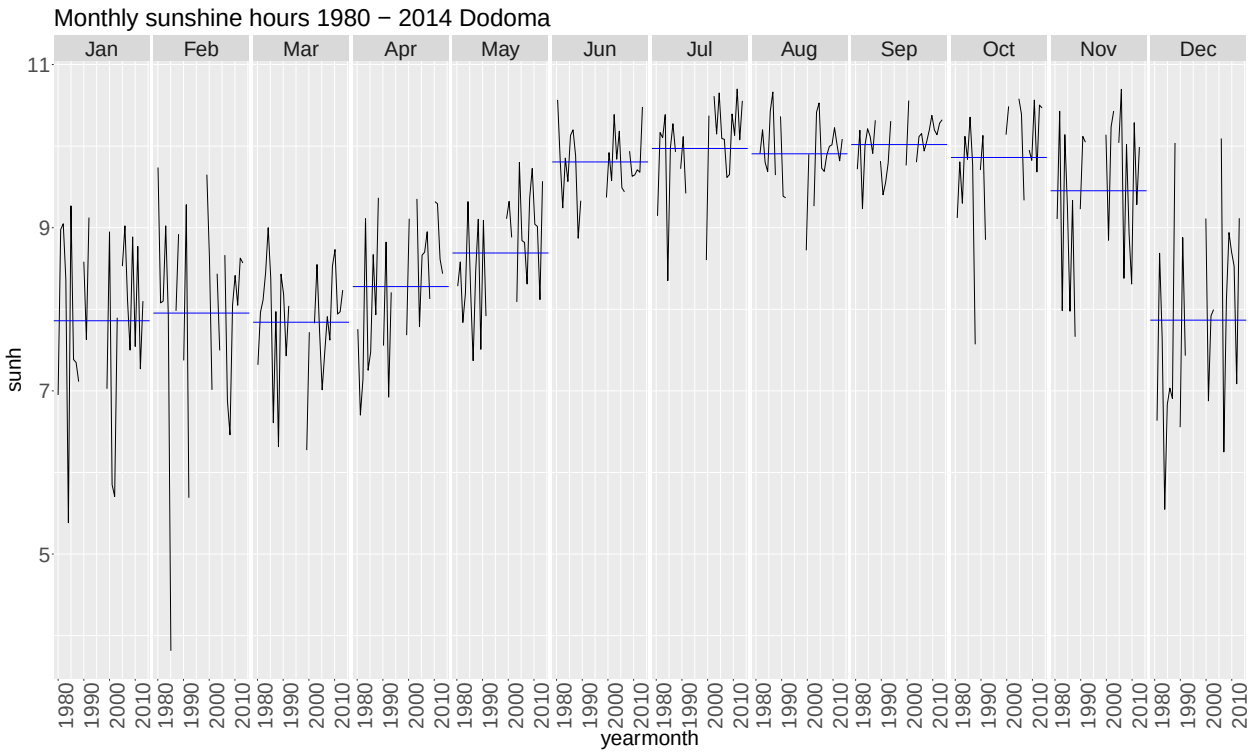
```
P4<-dodoma_monthly%>%
  gg_subseries(sunh)+
  ggtitle("Monthly sunshine hours 1980 - 2014 Dodoma")+
  theme(plot.title = element_text(size = 35),
        strip.text.x = element_text(size = 30),
        axis.text = element_text(size = 30),
        axis.title = element_text(size = 30))
```

The 4 figures below show each of the measures plotted using `gg_subseries()`. The blue line across each plot shows the mean line for each month across all years in the analysis (1980 - 2014). Therefore, we can see the seasonality quite clearly from the pattern of these mean lines. The individual plots show the values from each year for that particular month, allowing us to see a trend if one were to exist. There is little evidence of trends in each of these figures except perhaps a slight growth in the minimum temperature in recent years, especially in the middle of the year. There seems to be no overall trend to sunshine hours or rainfall.




Monthly minimum temperatures 1980 − 2014 Dodoma
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
tmin
18
16
14
12
10
1980
1990
2000
2010
yearmonth


Monthly rainfall(mm) 1980 − 2014 Dodoma
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
rain
400
300
200
100
0
1980
1990
2000
2010
yearmonth



Autocorrelation

Using these same packages, it is quite simple to calculate and plot the autocorrelation of a time series. Autocorrelation being a measure of the linear relationship between lagged values of a time series.

The `ACF` function from the `feasts` package can calculate these values as well as set a maximum number of calculations (`lag_max`).

In the first tsibble output below, you can see the autocorrelation coefficients lagged for 10 days.

The following two figures show the lagged coefficients for a whole year using daily data for daily minimum temperature and daily maximum temperature respectively. The 3rd and 4th figures show the same measures on a monthly scale over the course of 5 years of lagged months.

The blue dashed lines on each graph show the critical limit for significance in each analysis. In other words, all values of r_t that extend past these limits are considered statistically significant.

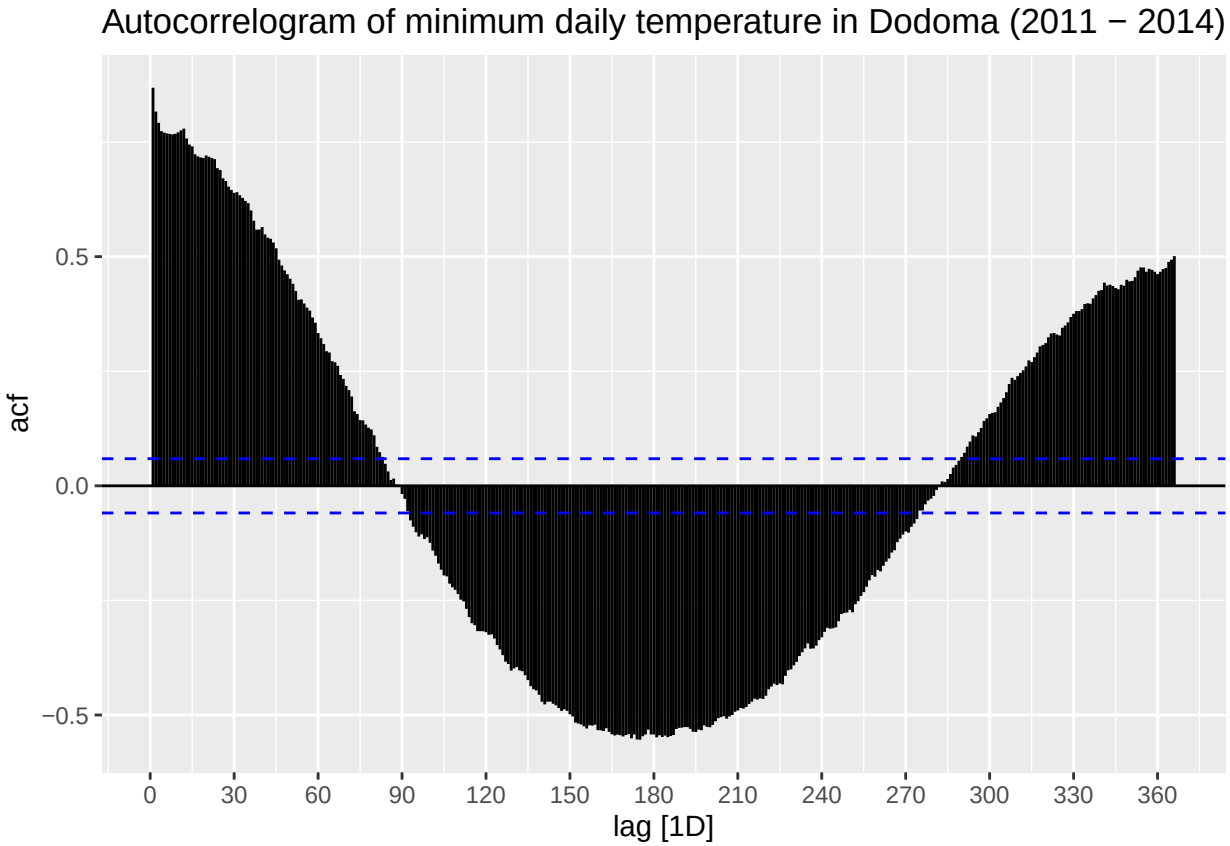
For minimum temperature, we can see a very smooth pattern in ACF values across the range of a year with positive correlation between a value on a given day and the 3 months prior as well as values between 9 and 12 months prior. While values which are 4 to 8 months away from a given day being significantly negatively correlated. Negative correlation between lagged values peaks at around a 6 months difference. This seasonality is additionally shown by the monthly plot. The monthly plot also demonstrates a possible increasing trend as the size of the coefficients decrease the further we move from time t. The highest coefficients are those which are 12 months apart, i.e the same month in each year. While the strongest negative relationships are for those which are in the middle of the 12 month period (6 months before/ 18 months before etc.)

Similar patterns are true of maximum temperature but is more susceptible to random dips and rises in ACF values, therefore suggesting that the patterns in minimum temperature are much more stable while maximum temperature is volatile and prone to outliers (this can be seen in some of the graphs).

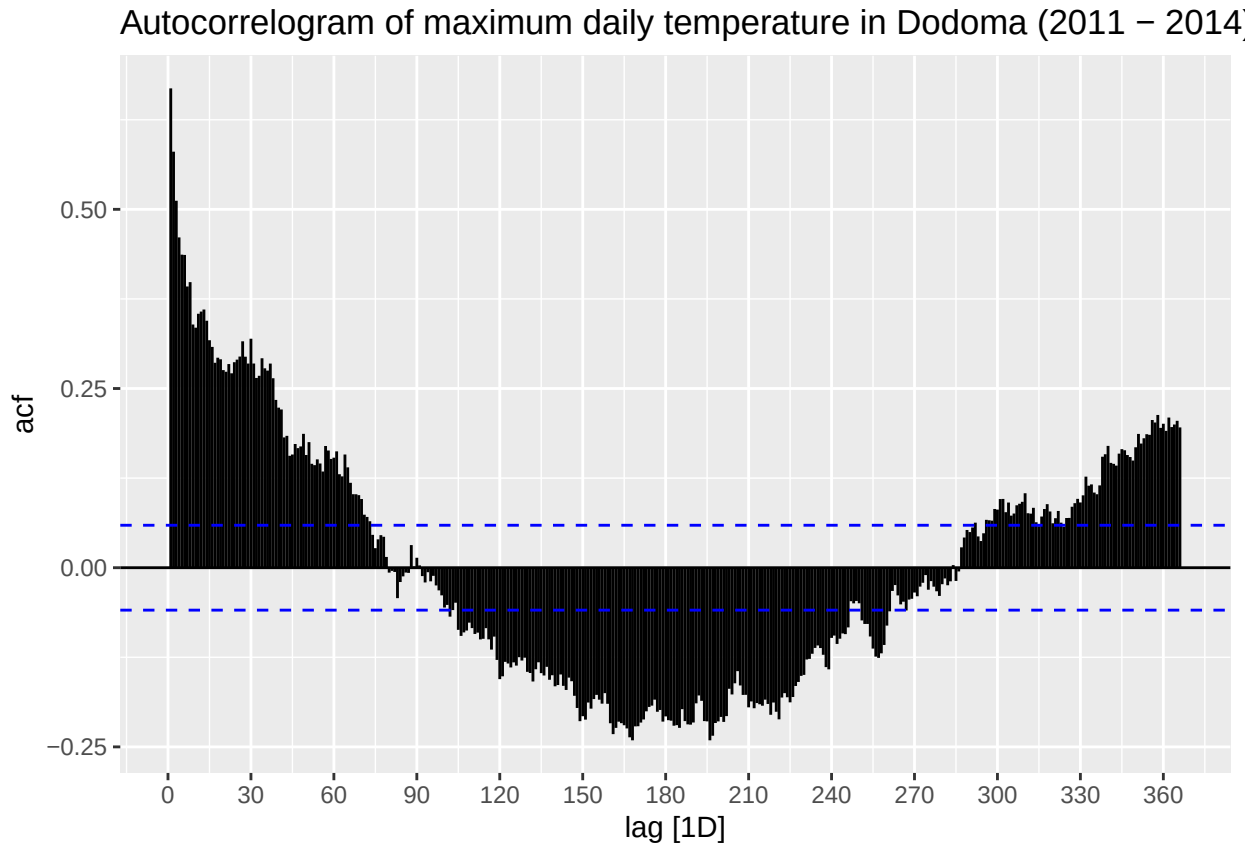
```
dodoma_daily %>%
  ACF(tmax, lag_max = 10)
```

```
## # A tsibble: 10 x 2 [1D]
##      lag   acf
##    <lag> <dbl>
##  1    1D 0.669
##  2    2D 0.580
##  3    3D 0.512
##  4    4D 0.461
##  5    5D 0.437
##  6    6D 0.436
##  7    7D 0.392
##  8    8D 0.398
##  9    9D 0.339
## 10   10D 0.335
```

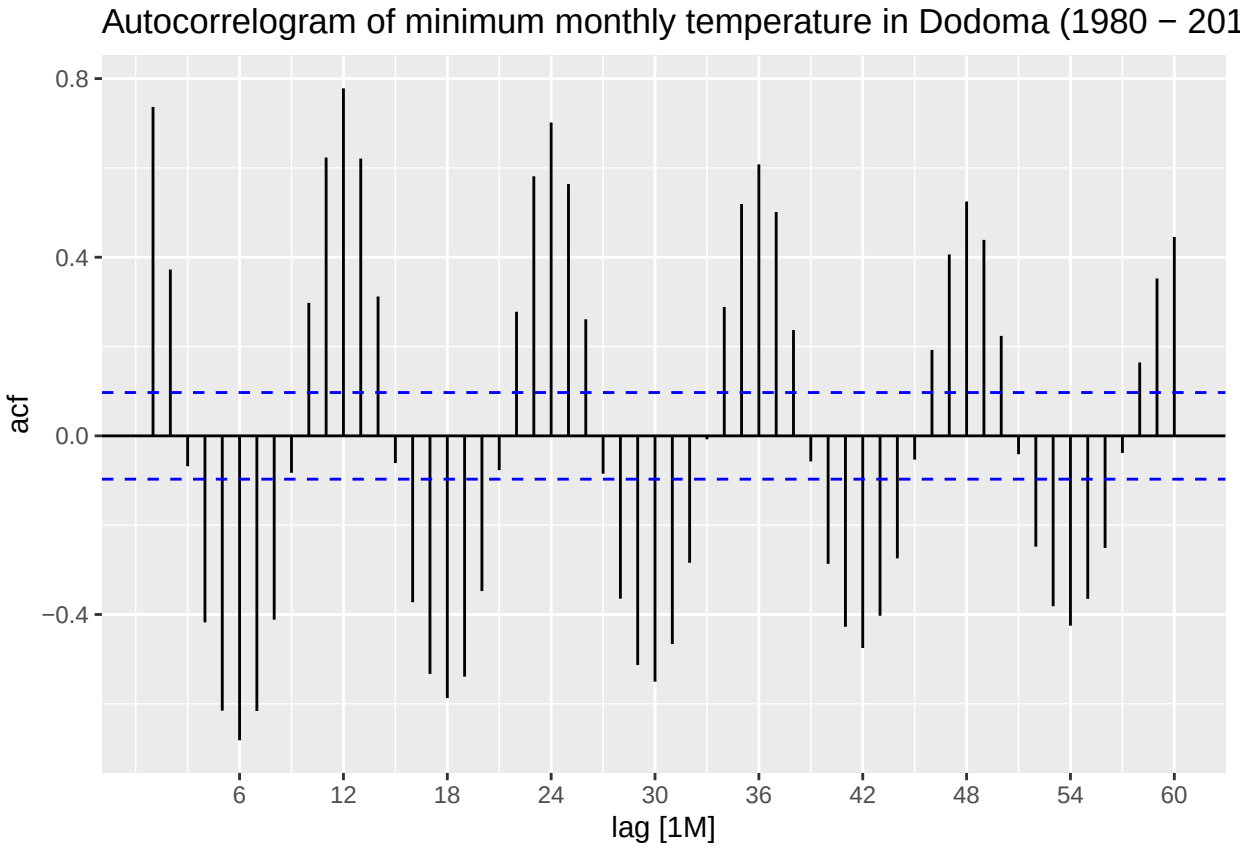
```
dodoma_daily %>%
  ACF(tmin, lag_max = 366)%>%
  autoplot()+
  scale_x_continuous(breaks = seq(0,360, 30))+
  ggtitle("Autocorrelogram of minimum daily temperature in Dodoma (2011 - 2014)")
```



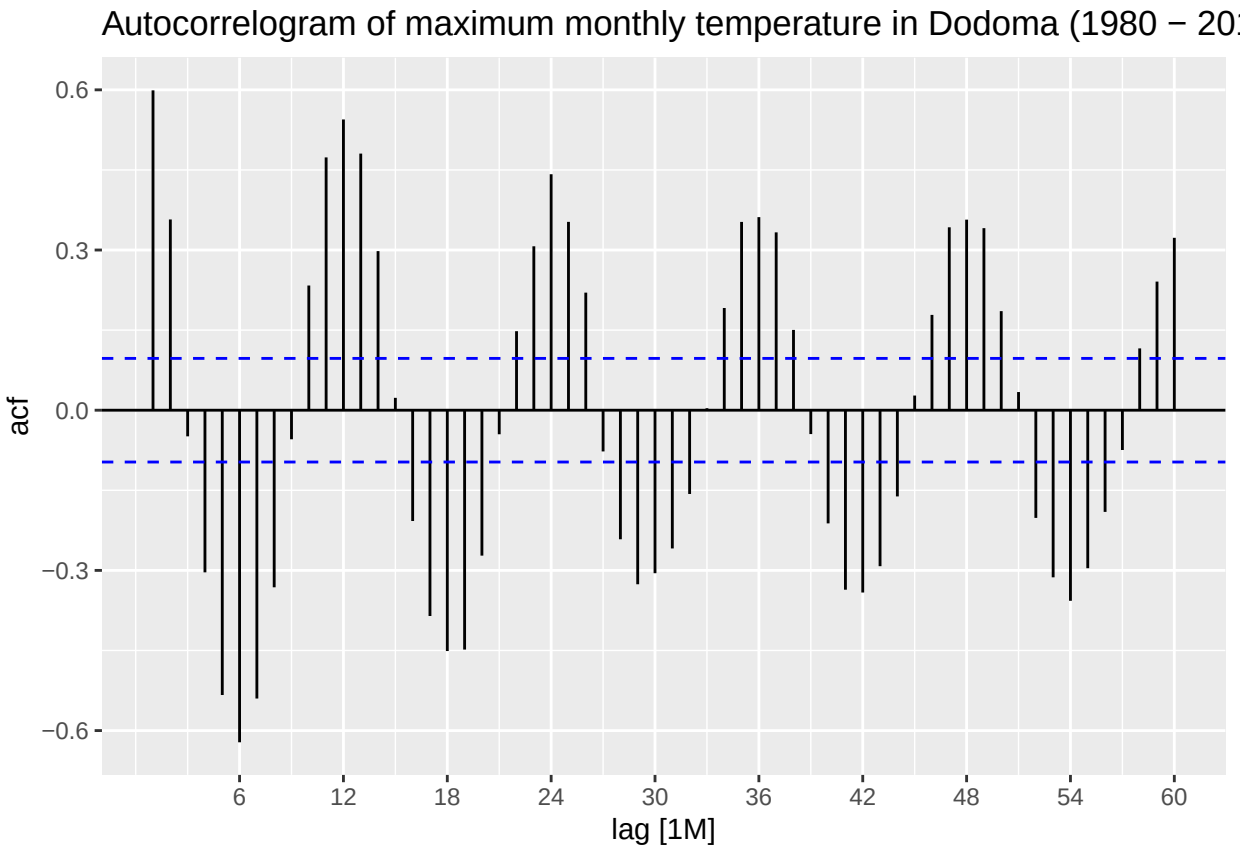
```
dodoma_daily %>%
  ACF(tmax, lag_max = 366)%>%
  autoplot()+
  scale_x_continuous(breaks = seq(0,360, 30))+
  ggtitle("Autocorrelogram of maximum daily temperature in Dodoma (2011 - 2014)")
```




```
dodoma_monthly%>%
  ACF(tmin, lag_max = 60)%>%
  autoplot()+
  ggtitle("Autocorrelogram of minimum monthly temperature in Dodoma (1980 - 2014)")
```




```
dodoma_monthly%>%
  ACF(tmax, lag_max = 60)%>%
  autoplot()+
  ggtitle("Autocorrelogram of maximum monthly temperature in Dodoma (1980 - 2014)")
```



STL decomposition

Time Series Components:

Many functions are included in the packages to compute the different components of a time series - the trend, the seasonal component and the remainder component. The daily data (2011 - 2014) shall be broken up into these components using the function `STL` from the `feasts` package. A classical decomposition was attempted but this did not handle the seasonality of the data very well due to this being daily data. STL ("Seasonal and Trend decomposition using Loess") is more capable of handling observations at the daily level.

Both of these methods require there are no missing data and no missing time periods, therefore rather than excluding any days with missing values, these have been interpolated using the `na.interp` function from the `forecasts` package.

The decomposition model is then specified using the `model` function from the `fabletools` package with the option for `robust` set to `TRUE`. This makes the decomposition robust to outliers such that they will not affect estimates of trend or season, only the remainder.

The `components` function can then be used to bring up the values generated by the decomposition (the trend, seasonal components (year, week), the remainder component and the seasonally adjusted value). The object which can be created using this function would be called a `dable` or a "decomposition table".

These can then be plotted quite easily using the `autoplot` function. This separates the data and the individual components into separate plots, the bar to the left of each plot is for scale reference (Demonstrating that the weekly component is quite small relative to other components). Here we can see that there is little evidence of an overall increasing/decreasing trend over the 3 years except for a sudden rise at the end,

however this is an artefact of the interpolation as the last 2 months of 2013 have no data attached and therefore they have all taken the value for October 31st 2013.

The yearly seasonal pattern is quite clear with and follows the same pattern we have observed in previous plots with it creating variations of between -2.5 and +2.5 degrees on top of the rest of the components, the weekly seasonal component is quite volatile and does not massively contribute to the overall seasonality of the data.

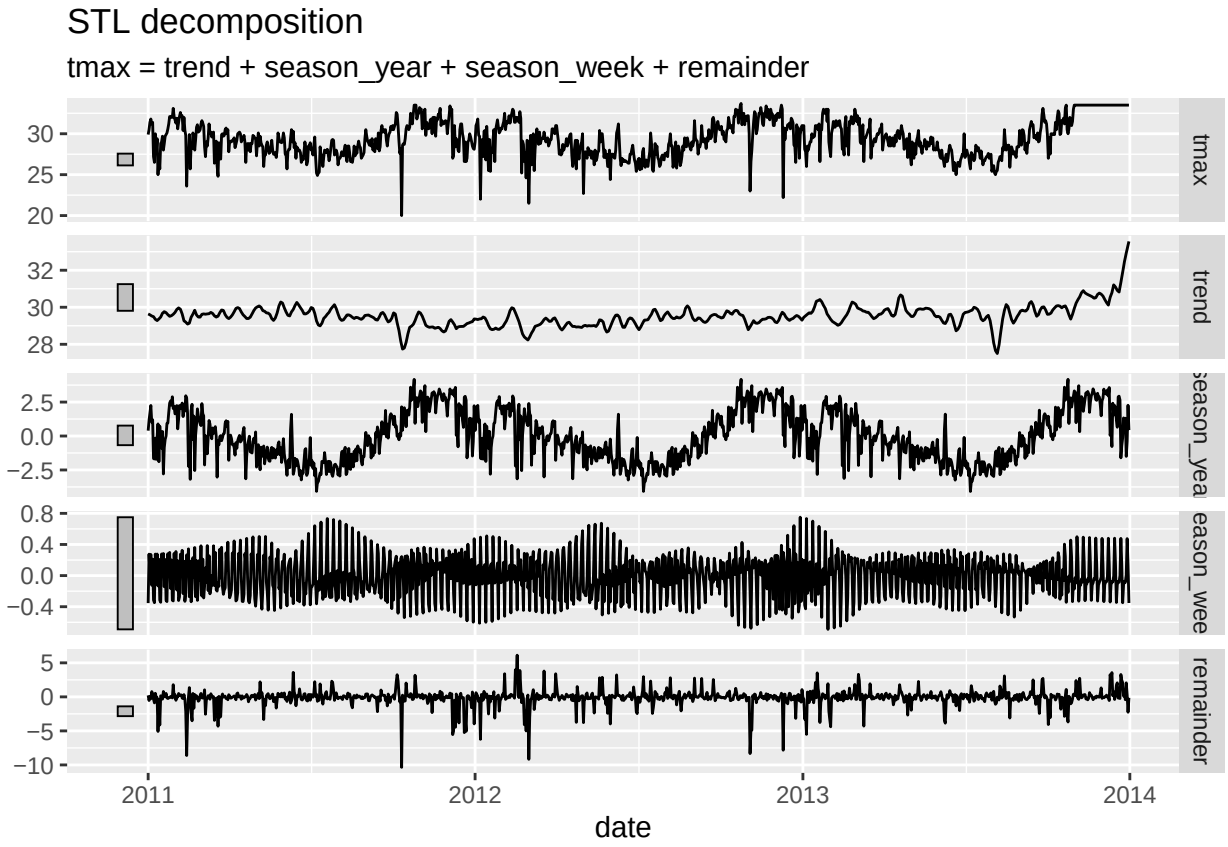
Much of the information about the outliers can be seen in the remainder component with some spikes causing additions of +5 or -10 to the rest of the components.

The next figure plots the trend only data against the actual data, this shows how significant the remainder and seasonal components are into forming the shape of the actual data as the trend fluctuates quite steadily at around 28 degrees Celsius.

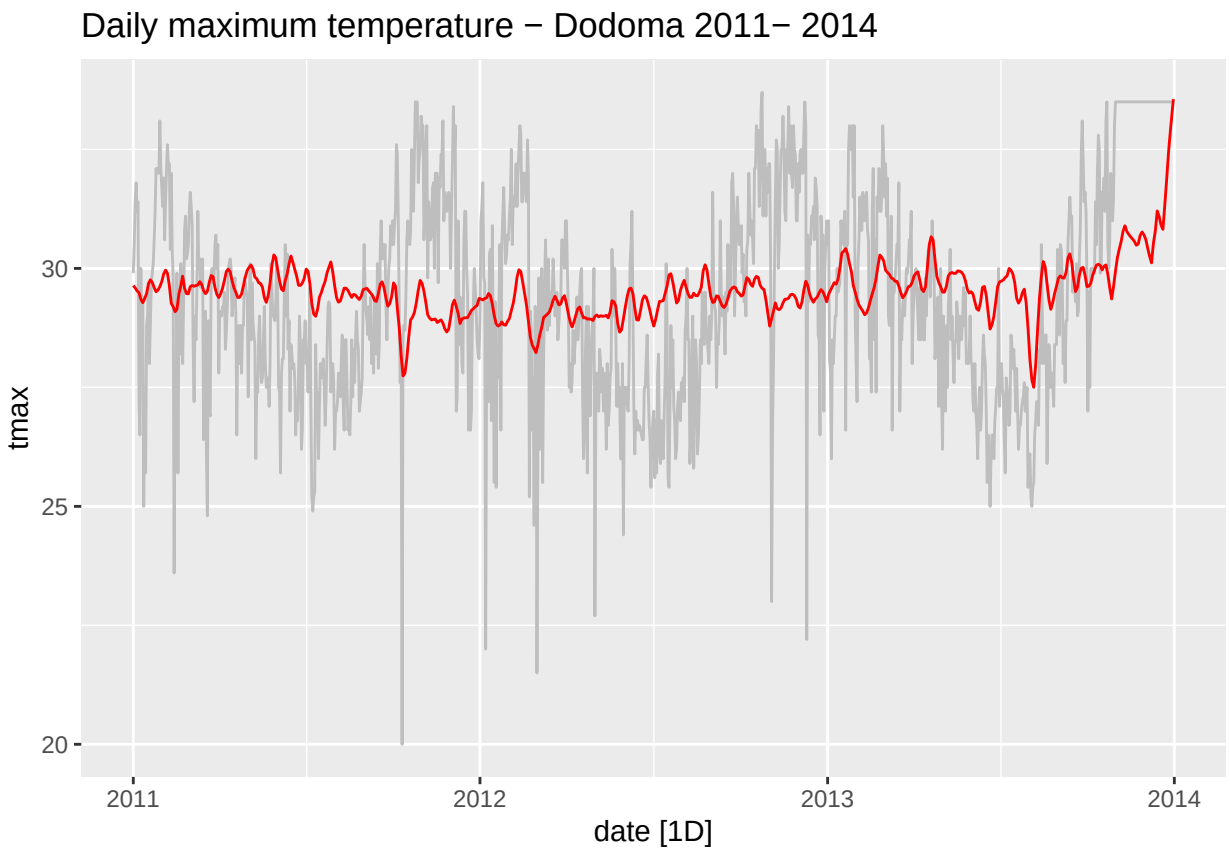
```
dodoma_daily$tmax<-na.interp(dodoma_daily$tmax)

dcmp1 <- dodoma_daily%>%
  model(STL(tmax, robust = TRUE))

components(dcmp1)
```

```
## # A dable:           1,096 x 8 [1D]
## # Key:               .model [1]
## # STL Decomposition: tmax = trend + season_year + season_week + remainder
##    .model date        tmax trend season_year season_week remainder season_adjust
##    <chr>  <date>     <dbl> <dbl>       <dbl>       <dbl>     <dbl>         <dbl>
##  1 STL(t~ 2011-01-01  29.9  29.6       0.395     -0.354     0.219          29.9
##  2 STL(t~ 2011-01-02  30.5  29.6       1.14       0.276    -0.534          29.1
##  3 STL(t~ 2011-01-03  31.4  29.6       1.70       0.0286    0.0812         29.7
##  4 STL(t~ 2011-01-04  31.8  29.6       2.25       0.271    -0.279          29.3
##  5 STL(t~ 2011-01-05  31.2  29.5       1.02      -0.123     0.777          30.3
##  6 STL(t~ 2011-01-06  31.4  29.5       1.20       0.211     0.476          30.0
##  7 STL(t~ 2011-01-07  27.2  29.5      -1.66      -0.315    -0.321          29.2
##  8 STL(t~ 2011-01-08  26.5  29.4      -1.69      -0.348    -0.881          28.5
##  9 STL(t~ 2011-01-09  30    29.4      -0.156      0.277     0.529          29.9
## 10 STL(t~ 2011-01-10  27.6  29.3      -1.72       0.0289   -0.0248         29.3
## # ... with 1,086 more rows
```

```
components(dcmp1)%>%
  autoplot()
```




```
dodoma_daily%>%
  autoplot(tmax, color = "gray") +
  autolayer(components(dcmp1), trend, color = "red")+
  ggtitle("Daily maximum temperature - Dodoma 2011- 2014")
```



It is possible to calculate many additional features on the basis of an STL decomposition using the `features` and `feat_stl` function from the `feasts` package.

These calculate a number of measures relating to the decomposition. The first table below shows the defaulted stl which has calculated the values using weekly periods, as we are more interested in annual seasonality the second table shows the values when using a year as the period.

- The strength of the trend component (0 - 1, this time series has a low trend strength of just 0.104 (seasonality period = year))
- The strength of the seasonality of a period (week in first table, year in second)
  - strong seasonality across the annual period ($F_S = 0.662$)
- Seasonal peak and troughs - day where maximum temperature is at its highest and lowest (November 22nd and May 2nd respectively)
- spikiness, linearity and curvature - various descriptors of the shape of the remainder component
- 2 autocorrelation measures of the remainder component

The `feat_acf` function can be used similarly to generate measures associated with the autocorrelation of time time series.

```
dodoma_daily%>%
  features(tmax, feat_stl)
```

```
## # A tibble: 1 x 9
##   trend_strength seasonal_streng~ seasonal_peak_w~ seasonal_trough~ spikiness
##            <dbl>            <dbl>            <dbl>            <dbl>     <dbl>
## 1          0.800            0.180                5                3   6.64e-6
## # ... with 4 more variables: linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

```
STL<-round(feat_stl(dodoma_daily$tmax, .period = 365.2425), 3)
STL[3:4]<-round(STL[3:4], 0)
knitr::kable(STL)
```

	x
trend_strength	0.104
seasonal_strength_365.2425	0.662
seasonal_peak_365.2425	326.000
seasonal_trough_365.2425	122.000
spikiness	0.000
linearity	8.684
curvature	8.754
stl_e_acf1	0.513
stl_e_acf10	0.563

```
dodoma_daily%>%
  features(tmax, feat_acf)
```

```
## # A tibble: 1 x 7
##    acf1 acf10 diff1_acf1 diff1_acf10 diff2_acf1 diff2_acf10 season_acf1
##   <dbl> <dbl>      <dbl>       <dbl>      <dbl>       <dbl>       <dbl>
## 1 0.761  3.53     -0.358       0.152     -0.617       0.427       0.534
```

Simple Forecasting Methods

Often the aim with time series data is to forecast the data into the future. In other words, estimate future values of the variable of interest, in this case maximum daily temperatures. There are many simple methods for achieving this process which shall be demonstrated in this section.

The methods demonstrated in this section may not be the best for climatic data but they are relatively simple to understand.

This section shall demonstrate how to conduct forecasts using the following 3 methods;

- Average Method - All forecast values are equal to the mean of the historical data
- Naive Method - All forecast values are equal to last observed value of the time series data
- Seasonal Naive - All forecast values are equal to last observed value from the same point in time (i.e the same day in the previous year)

In this code below there are first 2 lines to tidy/sort the data for this process. the first line removes the data for the last 2 months of 2013 as these contained all missing values therefore it is not useful at this time. The next line creates a training set of the data.

This is a portion of the data (usually around 75/80%) that is used to fit the model, this allows us to then compare the forecasts it produces with the real data from the time series to assess for accuracy. In this example, the months of August, September and October have been removed from the data by using the `filter_index` function from the `tsibble` package. This works similar to the `filter` function seen previously but focuses on the index variable (date). Here all values from the 1st of January 2011 to July 31st 2013 will be retained in the data.

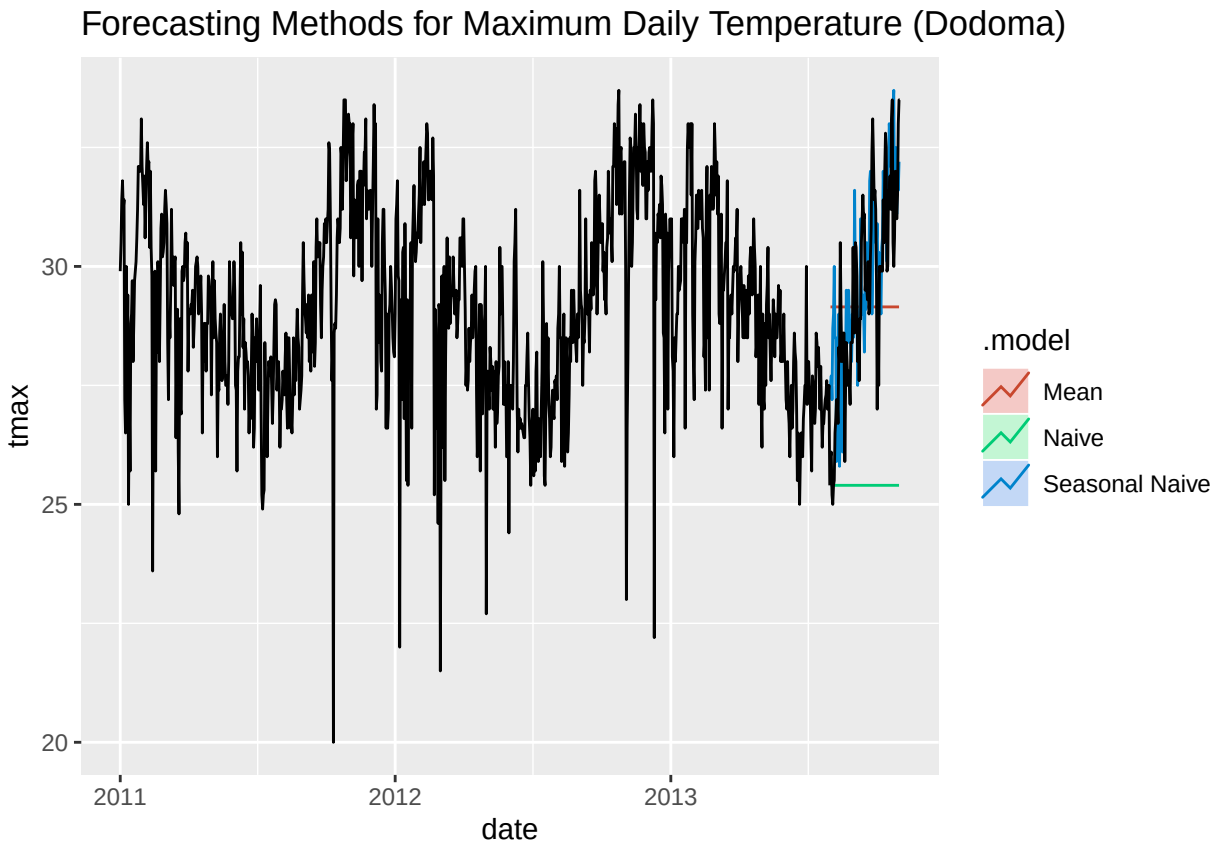
The 3 different models are then fitted on the `train_dodoma` data using the `model` function from the `fabletools` package already seen when conducting the STL decomposition. 3 more functions from the same package are then used to specify each model;

- `MEAN` = Average Method
- `NAIVE` = Naive Method
- `SNAIVE` = Seasonal Naive Method (with `lag("year")` to specify annual seasonality not weekly)

A forecasting table or `fable` is then created by using the fitted models (`tmax_fit`) and the `forecast` function. h is used to specify how many units of the index should be forecast past the end of the time series. In this case `h = 92` as there are 92 days across August, September and October (the months with real data to be compared against the forecasts).

The forecasts from the 3 models are then plotted against the real values from the data for those 3 months. The average and Naive methods create a simply straight line at one fixed value. The Naive model is quite far off from the real values as it does not expect the temperatures to rise as they do. The average method is closer but of course still assumes a constant maxim temperature and no variance. The seasonal naive method is much closer to reality as it allows for the inclusion of seasonality though of course it assumes the seasonal pattern is constant from the point it was last observed.

```
dodoma_daily<-dodoma_daily[1:1036,]

train_dodoma<-dodoma_daily%>%
  filter_index("2011-01-01" ~ "2013-07-31")

tmax_fit <-train_dodoma%>%
  model(Mean = MEAN(tmax),
        Naive = NAIVE(tmax),
        `Seasonal Naive` = SNAIVE(tmax  ~ lag("year")))

tmax_dc <- tmax_fit %>% forecast(h=92)

tmax_dc %>%
  autoplot(train_dodoma, level = NULL)+
  autolayer(filter_index(dodoma_daily, "2013-08-01" ~.), tmax, colour = "black")+
  ggtitle("Forecasting Methods for Maximum Daily Temperature (Dodoma)")
```



The `augment` function can be used to access the fitted and residual values from these models for each point in time, here you can see that for everyday in this time series the fitted value of the Mean model will be 29.15 degrees Celsius.

```
augment(tmax_fit)
```

```
## # A tsibble: 2,829 x 5 [1D]
## # Key:       .model [3]
##    .model date        tmax .fitted .resid
##    <chr>  <date>     <dbl>   <dbl>  <dbl>
##  1 Mean   2011-01-01  29.9    29.1  0.751
##  2 Mean   2011-01-02  30.5    29.1  1.35
##  3 Mean   2011-01-03  31.4    29.1  2.25
##  4 Mean   2011-01-04  31.8    29.1  2.65
##  5 Mean   2011-01-05  31.2    29.1  2.05
##  6 Mean   2011-01-06  31.4    29.1  2.25
##  7 Mean   2011-01-07  27.2    29.1 -1.95
##  8 Mean   2011-01-08  26.5    29.1 -2.65
##  9 Mean   2011-01-09  30      29.1  0.851
## 10 Mean   2011-01-10  27.6    29.1 -1.55
## # ... with 2,819 more rows
```

Additionally, the `accuracy` function can be used to calculate various measures regarding the forecast accuracy of the fitted models. These measures include the Mean Error (ME), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Scaled Error (MASE) and Mean Absolute Percentage Error(MAPE). The idea for each method is that the "best" model will be the one with the minimum value for a given measure

of accuracy. Sometimes different methods will result in different methods being considered the best. However, in this case the best method according to each measure is the Seasonal Naive model. This is not surprising given the plot of the forecasts from each model.

```
accuracy(tmax_dc, dodoma_daily)
```

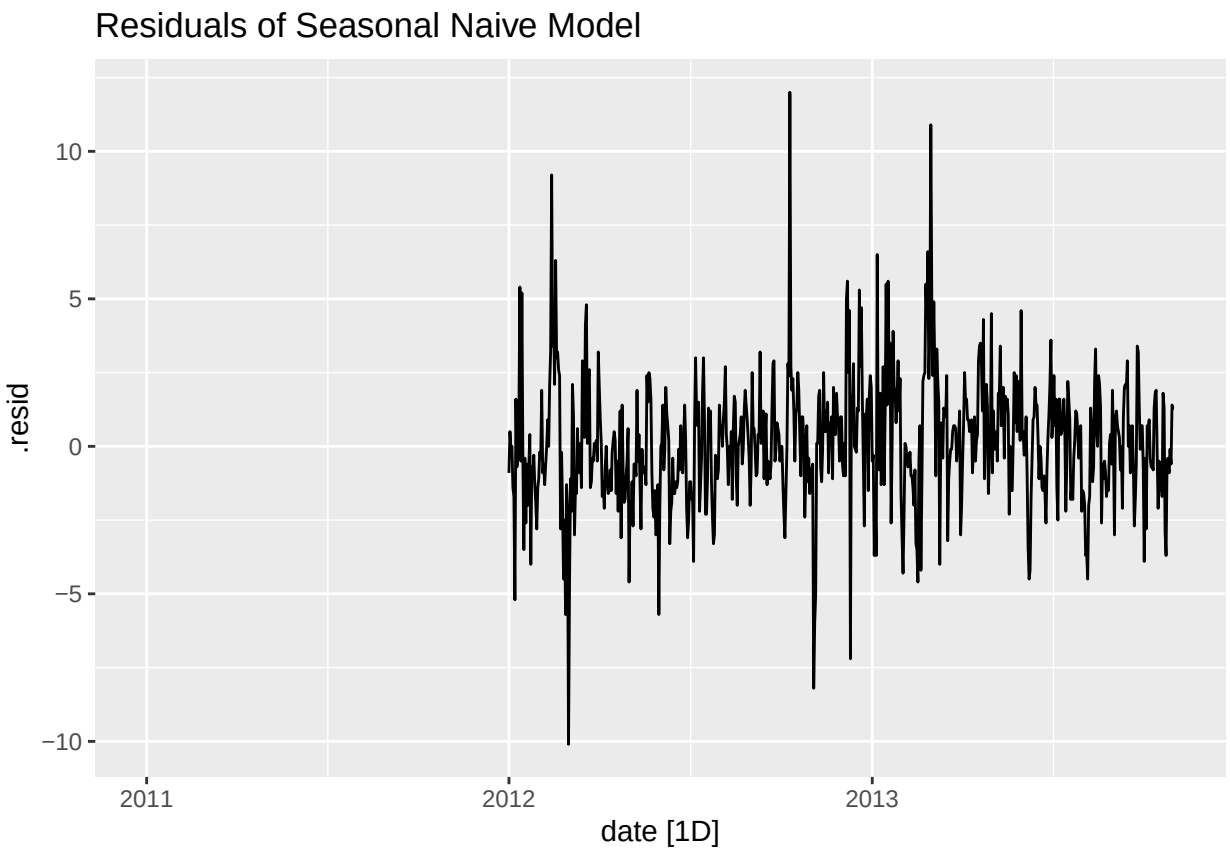
```
## # A tibble: 3 x 9
##   .model          .type     ME  RMSE   MAE    MPE  MAPE  MASE  ACF1
##   <chr>           <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 Mean            Test   0.379  2.07  1.73  0.801  5.86 1.09  0.806
## 2 Naive           Test   4.13   4.60  4.14 13.6   13.6  2.61  0.806
## 3 Seasonal Naive Test  -0.204  1.71  1.37 -0.929  4.71 0.863 0.473
```

Residuals As with any statistical model it is always advisable to analyse the residuals to check for either violations of model assumptions or other features which indicate that the model can still be improved. This section shall analyse the residuals of the seasonal naive model.

First the seasonal model has been recreated and augmented into a new data frame. The residuals are then plotted using `autoplot`.

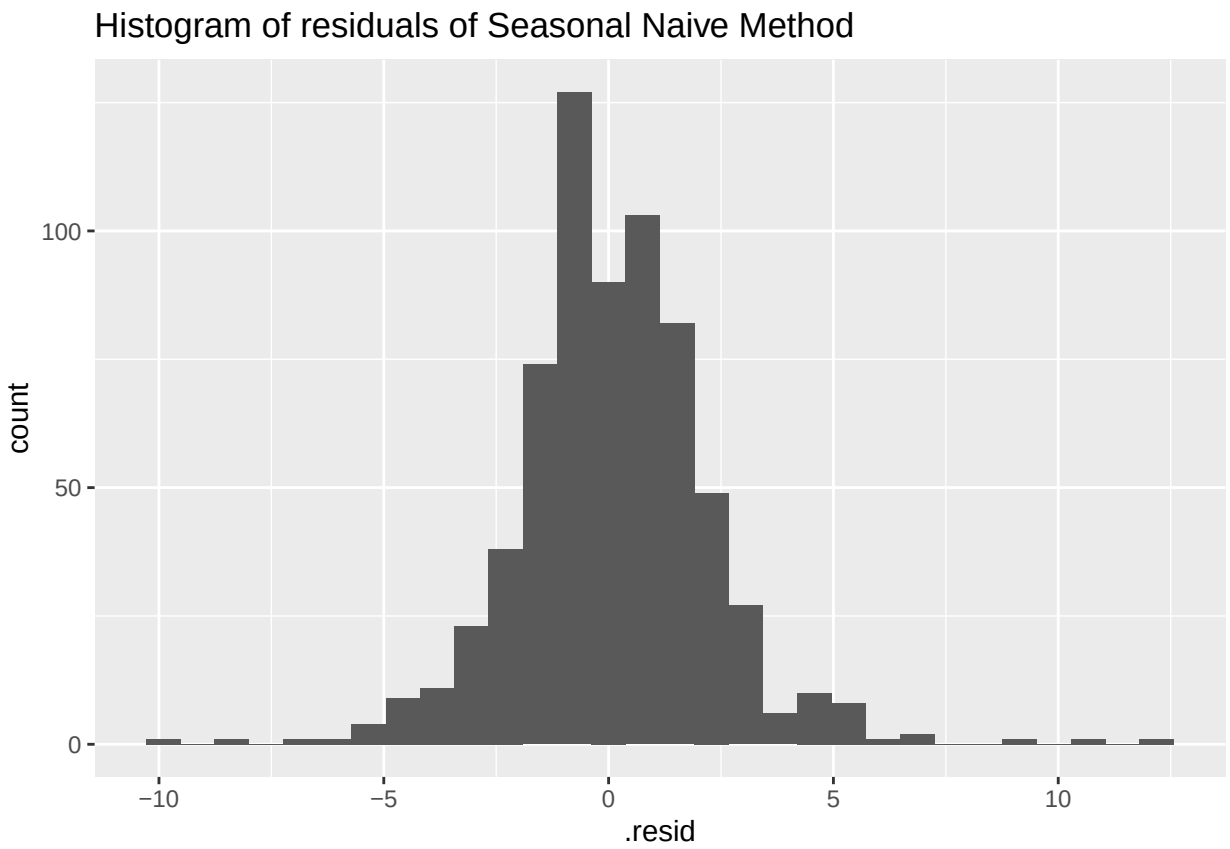
Because the Seasonal Naive model uses the value from the same day in the previous year in our example, there are no residuals for observations in 2011 as there is no previous observation to model within our data. However it seems that while there is not much of a pattern to the residuals, some are very large at between 5 and 10 degrees (plus or minus) with 3 observations actually more than 10 degrees off. This plot is used to check whether he residuals have a mean of 0, if they do not then this suggests the forecasts are biased. This plot suggests this assumption has not been violated.

```
aug_tmax <- dodoma_daily%>%
  model(SNAIVE(tmax ~ lag("year")))%>%
  augment()

aug_tmax %>%
  autoplot(.resid)+
  ggtitle("Residuals of Seasonal Naive Model")
```



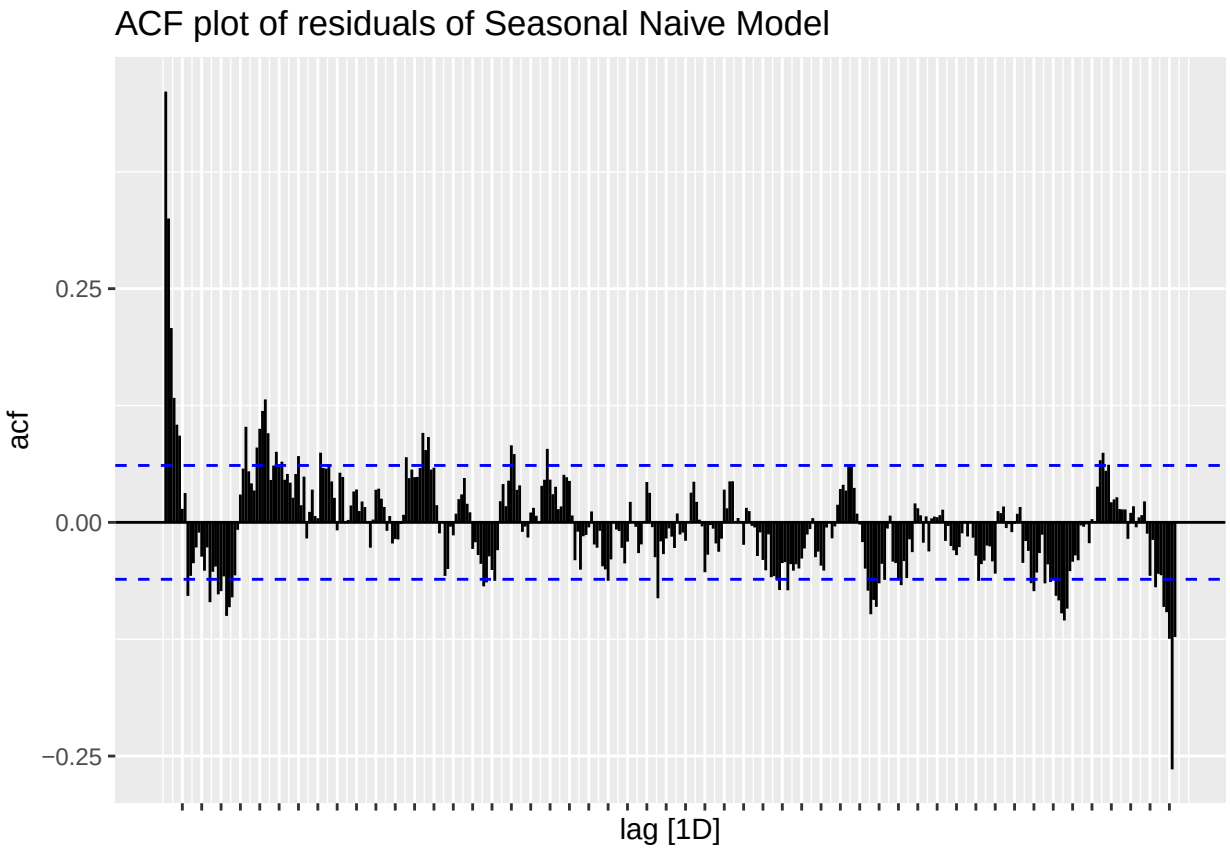
The next plot, a histogram, plots the distribution of the residuals. Another assumption of the model is that the residuals are normally distributed. This assumption appears to hold as there does not seem to be an indication of a positive or negative skew and the distribution is not bimodal.

```
aug_tmax%>%
  ggplot(aes(x = .resid))+
  geom_histogram()+
  ggtitle("Histogram of residuals of Seasonal Naive Method")
```



Lastly, residuals should not be correlated. If there is then that means there is information left in the residuals which should be used in the forecasting. This can be checked using an ACF plot as previously demonstrated with the main data itself. This tells us that the first few lagged residuals are actually significantly positively correlated and that residuals nearly a year prior are negatively correlated. This suggest this model may not be good enough as there is additional information which could be used to calculate better forecasts.

```
aug_tmax%>%
  ACF(.resid, lag_max = 366)%>%
  autoplot()+
  theme(axis.text.x = element_blank())+
  ggtitle("ACF plot of residuals of Seasonal Naive Model")
```



The feasts package has provided a few functions to conduct portmanteau tests for autocorrelation. These are formal tests which test whether or not the ACF of residuals is equivalent to white noise, i.e a time series with no trends, seasonality or correlation. In the code below, one of these tests (The Ljung - box test) has been conducted using the `ljung_box` function with the degrees of freedom set to 0 as there are no parameters in our model and the lag set to 10 days. As the p-value is less than 0.05 then this tells us that the ACF of the residuals is significantly different from a white noise series and therefore the assumption of uncorrelated residuals is unfortunately violated. Therefore the model can be improved.

```
aug_tmax %>%
  features(.resid, ljung_box, lag = 10, dof=0)
```

```
## # A tibble: 1 x 3
##   .model                          lb_stat lb_pvalue
##   <chr>                             <dbl>     <dbl>
## 1 "SNAIVE(tmax ~ lag(\"year\"))"    276.         0
```

ljung_box()

ARIMA Modelling

One more advanced approach to modelling and forecasting a time series is ARIMA modelling (Auto-Regressive Integrated Moving Average). These methods use the autocorrelations in the data for modelling. In other words they create regression models using only the values of the time series as predictors. This section shall briefly explore different components of this modelling procedure including; the AR model, the MA model and combining these into an ARIMA model with seasonality. This section is designed to be a

general overview and is not exhaustive. This section models daily data which is not quite as well suited to forecasting, especially when attempting to incorporate annual seasonality, at least this is true of modelling in R. It is recommended to look towards more advanced sources for more detailed descriptions and examples of how to conduct ARIMA modelling.

For this section a new data frame has been made which includes all daily observations from 1980 onwards and a new tsibble has been created. The new data frame `dodoma_daily1980` runs from January 1st 1980 to 31st October 2013.

```
dodoma_daily1980<- dodoma%>%
  filter(year>=1980)

dodoma_daily1980<-dodoma_daily1980%>%
  mutate(date = as_date(date))%>%
  select(-year, -month, -day, -month_abbr, -doy_366)%>%
  as_tsibble(index = date)%>%
  arrange(date)

dodoma_daily1980<-dodoma_daily1980%>%
  filter_index("1980-01-01" ~ "2013-10-31")
```

AR model

The AR component or Auto-regression is a model which uses lagged values of a time series as predictors instead of covariates. The term auto regression implies that it is a regression of a variable against itself. The model is a linear combination of past values of the variable. It is refereed to as an AR(p) model where p denotes how many lagged values are being used as predictors. If we used values from the previous 5 days then this would be an AR(5) model, if we used the past 10 days then it would be an AR(10) model. Though it is unlikely we will need models of orders this high as most of the information required will likely be within a few time periods.

The model can fitted quite similarly to the STL decomposition by using the `model` function from fabletools. Here the `ARIMA` function is utilised, this can require slightly more specification than previous models however. `pdq` is used to specify the different non-seasonal components of the model while `PDQ` is used for the seasonal components.

- p/P specifies the order of the AR component
- d/D specifies the order of differencing - this is the number of differencing operations needed to make the time series stationary (for this demonstration this has been set to 0 but ideally would be incorporated and decided before modelling)
- q/Q specifies the order of the MA (Moving Average) Component - shall come back to this shortly (for now has been set to 0)

As d and q have been set to 0, a simple AR model has been specified using a constant and using just the previous days maximum temperature as a predictor.

The aim of these modelling procedures is to minimise the AIC, the AICc or BIC. The AICc is the recommended measure as it corrects for bias introduced by using too many predictors i.e over fitting. For the AR model the AICc (Corrected Akaike's Information Criterion) is 44175.67.

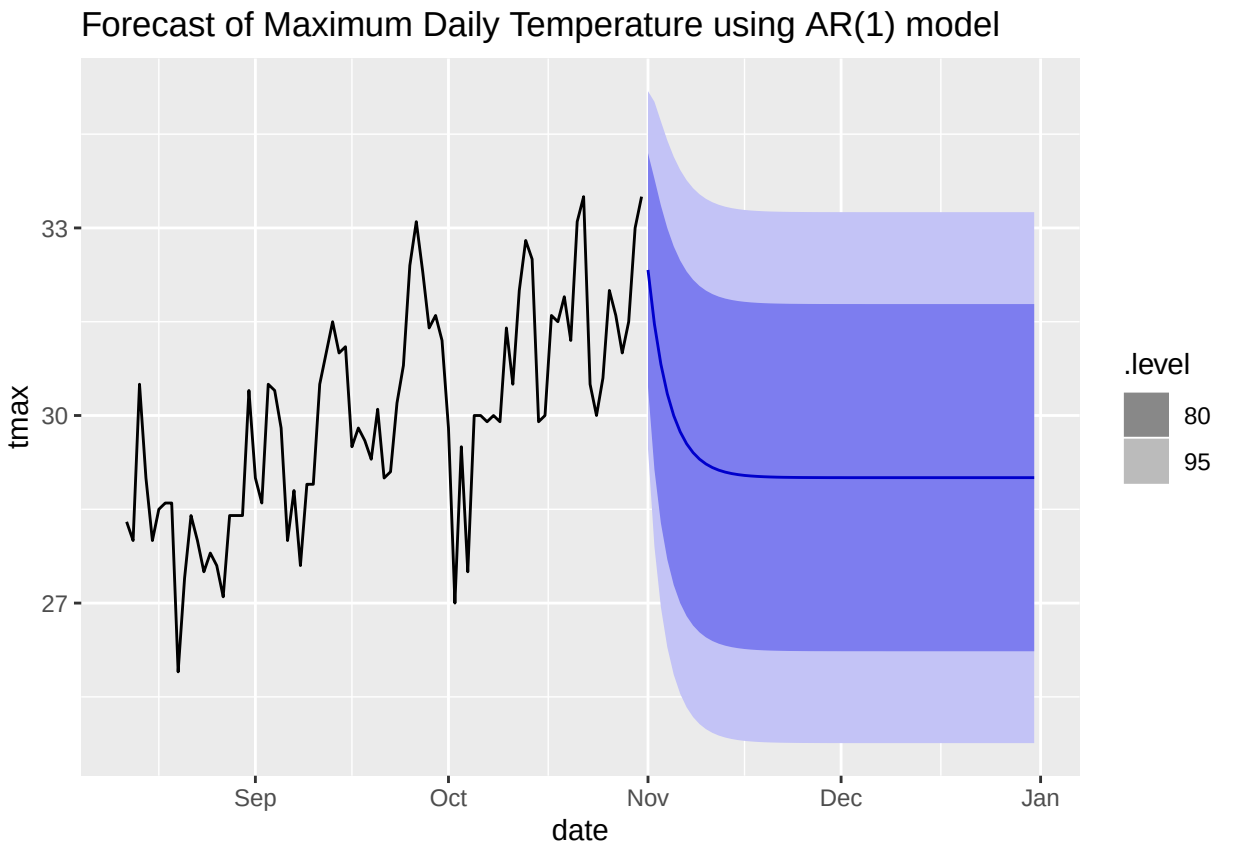
```
fitAR <- dodoma_daily1980%>%
  model(ARIMA(tmax ~ pdq(1,0,0)+PDQ(0,0,0)))
report(fitAR)
```

```
## Series: tmax
## Model: ARIMA(1,0,0) w/ mean
##
## Coefficients:
##          ar1  constant
##       0.7392    7.5653
## s.e.  0.0061    0.0132
##
## sigma^2 estimated as 2.129:  log likelihood=-22084.84
## AIC=44175.67   AICc=44175.67   BIC=44197.94
```

The `forecast` function can be used on this type of model just like it was used on the seasonal naive, naive and average models. With the added benefit that this will automatically include prediction intervals. If you needed prediction intervals from the other more basic methods demonstrated in the previous section you could pipe through the function `hilo()` to calculate these intervals.

This figure shows that the maximum temperature is predicted to first decline down towards roughly 28 degrees Celsius within the first few days of November at which point the prediction becomes fixed at this temperature. The prediction interval stays the same width across the prediction period.

```
fitAR %>%
  forecast(h=61)%>%
  autoplot(slice(dodoma_daily1980, (n()-80):n()))+
  ggtitle("Forecast of Maximum Daily Temperature using AR(1) model")
```



MA model

The MA or Moving Average part of the model works slight differently. It does not use the past values of the forecast variable but rather it uses past forecast errors in a model which is similar to regression. It is not a true regression as we do not observe the values of the forecast errors. Similar to the AR(p) model, we specificity an MA(q) model where q denotes the number of forecast errors being used to model the data. In this case, how many days prior are contributing to the model.

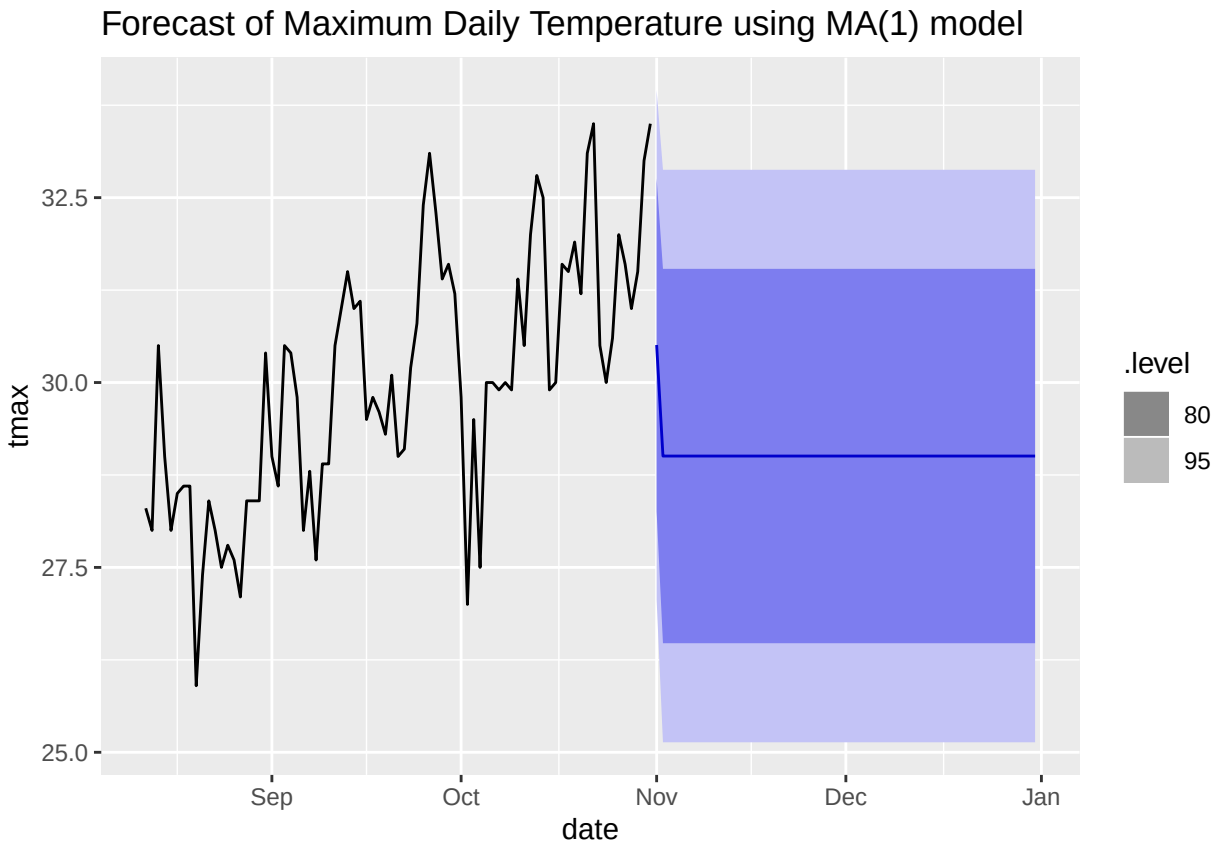
In terms of the code, the model is specified in much the same way as the AR model, the difference being that q has been set to 1 instead of p. All other terms in the model have been set to 0, therefore we only have a constant and the forecast error from the previous day as predictors. The AICc is much larger than it was for the AR(1) model, meaning the AR(1) model is a better fit to the data than the MA(1) model. Though this does not mean it is necessarily the best model to fit.

```
fitMA <- dodoma_daily1980%>%
  model(ARIMA(tmax ~ pdq(0,0,1)+PDQ(0,0,0)))
report(fitMA)
```

```
## Series: tmax
## Model: ARIMA(0,0,1) w/ mean
##
## Coefficients:
##          ma1  constant
##       0.5132   29.0060
## s.e.  0.0060    0.0241
##
## sigma^2 estimated as 3.088:  log likelihood=-24363.77
## AIC=48733.53   AICc=48733.53   BIC=48755.8
```

As the plot shows the forecast is similar in shape except that it drops down to roughly 28 degrees Celsius before becoming constant a few days earlier.

```
fitMA %>%
  forecast(h=61)%>%
  autoplot(slice(dodoma_daily1980, (n()-80):n()))+
    ggtitle("Forecast of Maximum Daily Temperature using MA(1) model")
```



###ARIMA Model

The ARIMA model itself is designed to incorporate both of these models together as well as allow for the differencing of a time series (the steps needed to make the time series stationary). Additionally it can incorporate seasonal terms which will likely improve forecast accuracy if there is strong seasonality in the data. This section includes incorporation of seasonal terms (capital PDQ to distinguish them form the non-seasonal pdq terms), however the functions struggle with annual seasonality of daily measures as they are technically irregular intervals (the average length of a year is not an integer due to leap years). Therefore the seasonality incorporated into these models is technically a weekly pattern, which is not something we would necessarily expect to have a significant pattern given this is climatic data.

An advantage of the ARIMA function being used is that you do not necessarily have to specify the pdq or PDQ terms in order to fit the model. If no terms are specified then the function will iterate through different models to find the one which minimises the AICc. It is possible a better model could be missed therefore you increase the power of the function by specifying `stepwise = FALSE` and `approximation = FALSE`, this alters the modelling algorithm to not use approximations or a stepwise modelling procedure.

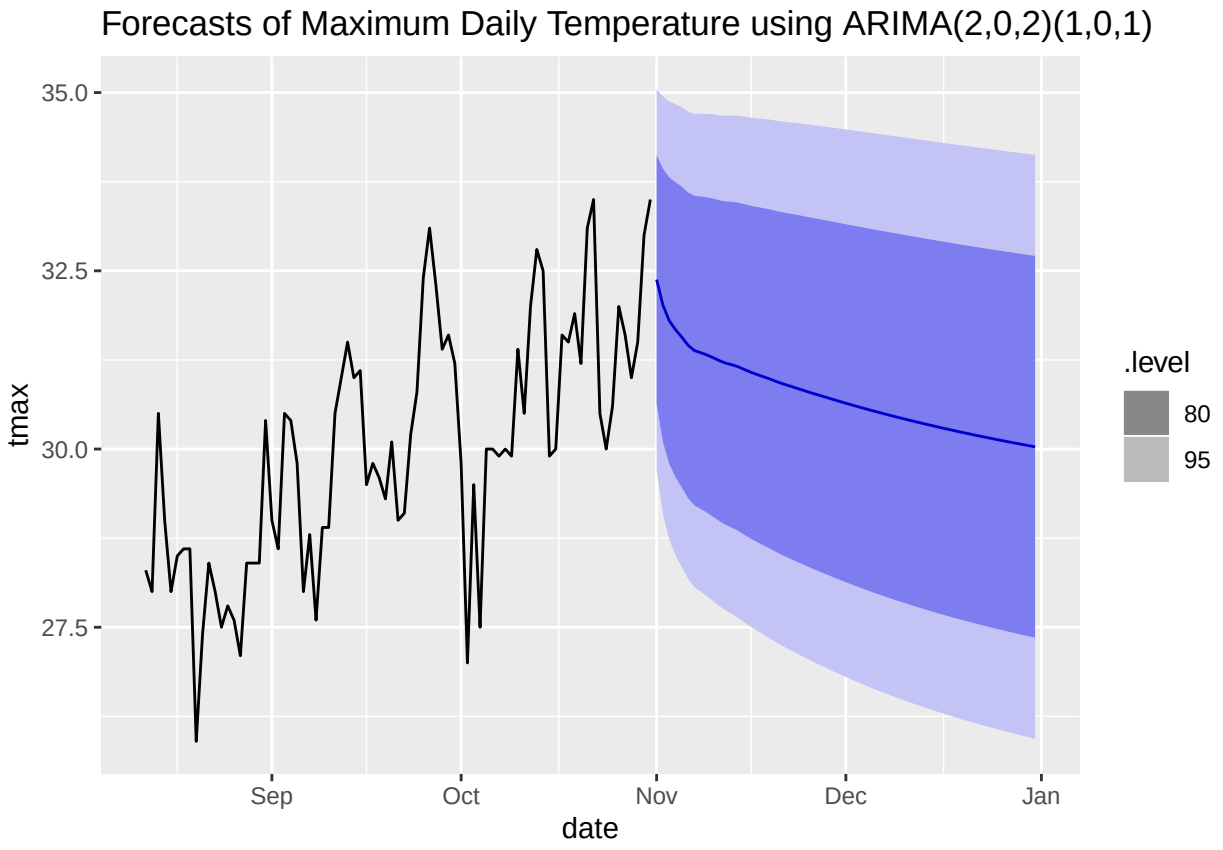
The code below has run this automatic procedure to generate the best model in terms of minimising the AICc. The function determines that a model with order (2,0,2)(1,0,1) is the best fit for this time series data. In other words there are 2 non-seasonal autoregressive terms, 1 seasonal autoregressive term, 2 non-seasonal moving average terms and 1 seasonal moving average term. There are no differencing terms applied to make the series stationary. The model output from report shows the various coefficients of the model. The AICc has reached a low point of 42483.53.

```
fitARMA <- dodoma_daily1980%>%
  model(ARIMA(tmax))
report(fitARMA)
```

```
## Series: tmax
## Model: ARIMA(2,0,2)(1,0,1)[7] w/ mean
##
## Coefficients:
##          ar1      ar2      ma1     ma2     sar1    sma1  constant
##       1.5822  -0.5884  -1.1286  0.2168  -0.3145  0.2984    0.2377
## s.e.  0.0277   0.0270   0.0300  0.0223   0.5641  0.5645    0.0014
##
## sigma^2 estimated as 1.854:  log likelihood=-21233.76
## AIC=42483.52   AICc=42483.53   BIC=42542.89
```

In the plot below the ARIMA model has been forecast for the months of November and December 2013. This model does not decline as rapidly as previous models and fluctuates slightly to reflect the incorporation of the weekly seasonal terms. This is still unlikely an optimal model as ideally we would also incorporate the annual seasonal pattern as based of the seasonal plots from earlier in this report November and December should have higher maximum temperatures than this forecast suggests.

```
fitARMA %>%
  forecast(h=61)%>%
  autoplot(slice(dodoma_daily1980, (n()-80):n()))+
  ggtitle("Forecasts of Maximum Daily Temperature using ARIMA(2,0,2)(1,0,1)")
```



Finally, it is worth comparing this "optimal" model to other similar models from the data. In the table below the AIC, AICc and BIC is shown for various model specifications including the AR(1), MA(1) and ARIMA(2,0,2)(1,0,1) models. The model specifications are shown in the first column [(p,d,q)(P,D,Q)]. The AICc varies considerably across the different specifications with the AR(1), and MA(1) models performing

relatively poorly compared to those which include both AR and MA terms. The seasonal models also tend perform slightly better but it is the (2,0,2)(1,0,1) which minimises the AICc.

```
knitr::kable(Models)
```

Model(ARIMA,SARIMA)	AIC	AICc	BIC
(1,1,1)(0,0,0)	42666.24	42666.24	42688.50
(1,1,1)(1,0,1)	42654.26	42654.27	42691.37
(1,0,1)(1,0,1)	42806.31	42806.31	42850.84
(2,0,1)(1,0,1)	42557.61	42557.62	42609.57
(1,0,2)(1,0,1)	42658.72	42658.73	42710.67
(1,0,0)(0,0,0)	44175.67	44175.67	44197.94
(0,0,1)(0,0,0)	48733.53	48733.53	48755.80
(2,0,2)(1,0,1)	42483.52	42483.53	42542.89