Temperature Analysis Abidjan

Karen Wood

July 28, 2017

Climate Change Data Analysis

In this code, I would like to first apply the time series method as described in otexts.org/fpp, specifically the Classical Decomposition method and look at the results.

First we will import our data:

```
library(readr)
GlobalLandTemperaturesByMajorCity <- read_csv("C:/Users/Karen/Google Drive/Kaggle/Climate Change/climat
```

```
## Parsed with column specification:
## cols(
##   dt = col_date(format = ""),
##   AverageTemperature = col_double(),
##   AverageTemperatureUncertainty = col_double(),
##   City = col_character(),
##   Country = col_character(),
##   Latitude = col_character(),
##   Longitude = col_character()
## )
```

The data I am using is from Kaggle, posted by Berkeley Earth. It was downloaded on July 28, 2017 at 3:25pm Pacific Time. In particular, I will examine the data from a single major city. The data appears to have temperatures in Celcius. Date format is in y-m-d format, but only 1 data point for each month, at least for the first city: Abidjan.

Let's first do one city. The first city: Abidjan.

```
AbidjanData<-GlobalLandTemperaturesByMajorCity[GlobalLandTemperaturesByMajorCity$City=="Abidjan",]
```

Now let's see if there's missing data.

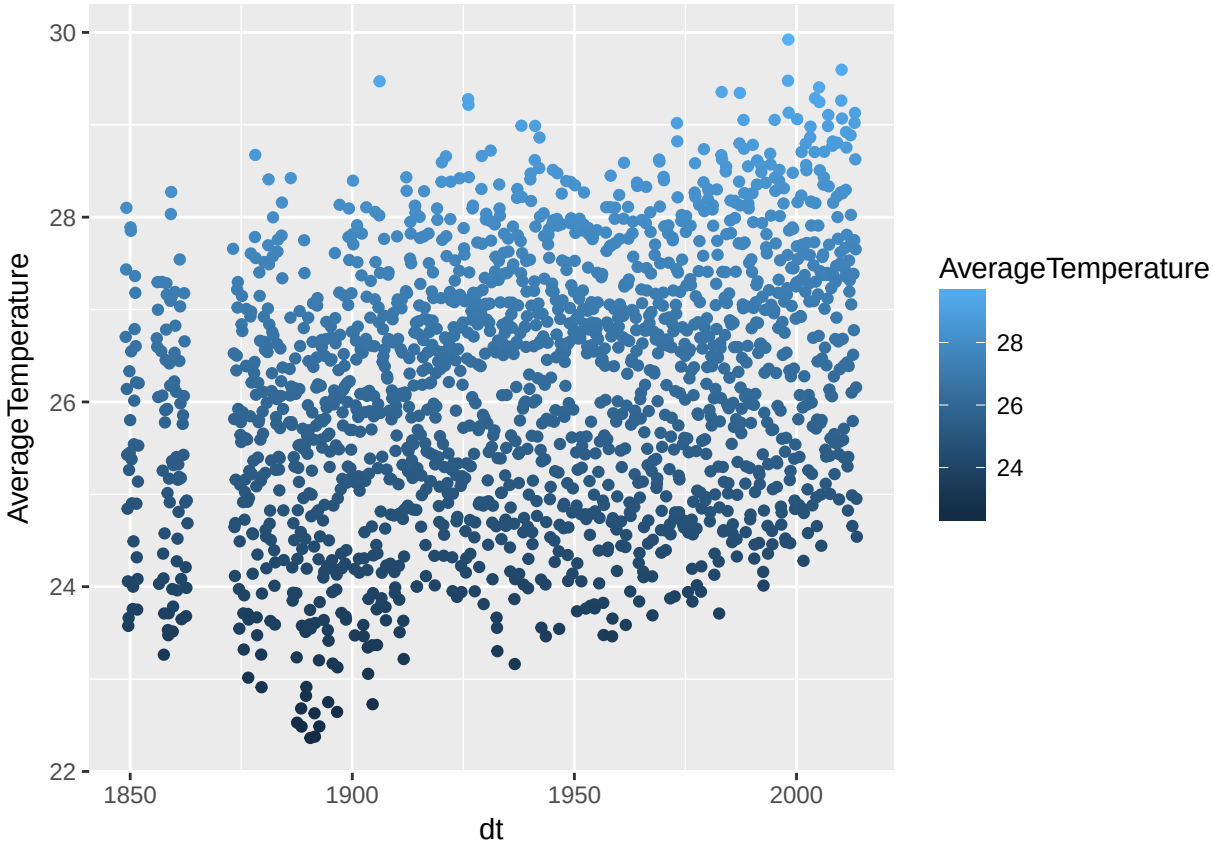
```
sum(is.na(AbidjanData$AverageTemperature))
```

```
## [1] 200
```

I got that there are 200 NAs. How will I deal with the missing data? Let's plot the data to see what the situation is. As it turns out, the dates are already in the format we would like.

```
library(ggplot2)
ggplot(AbidjanData, aes(dt,AverageTemperature))+geom_point(aes(color=AverageTemperature))
```

```
## Warning: Removed 200 rows containing missing values (geom_point).
```



It looks like much of the data missing is very old data. Let's restrict to just data more recent than 1900. Note that instead, we could use the forecast package and use tsclean() to input the missing values. However, with such large swaths of data missing, I think it better to restrict our data for now.

```
AbidjanData2<-AbidjanData[AbidjanData$dt>as.Date("1900-01-01"),]
```

Now how many missing pieces of data are there?

```
sum(is.na(AbidjanData2$AverageTemperature))
```

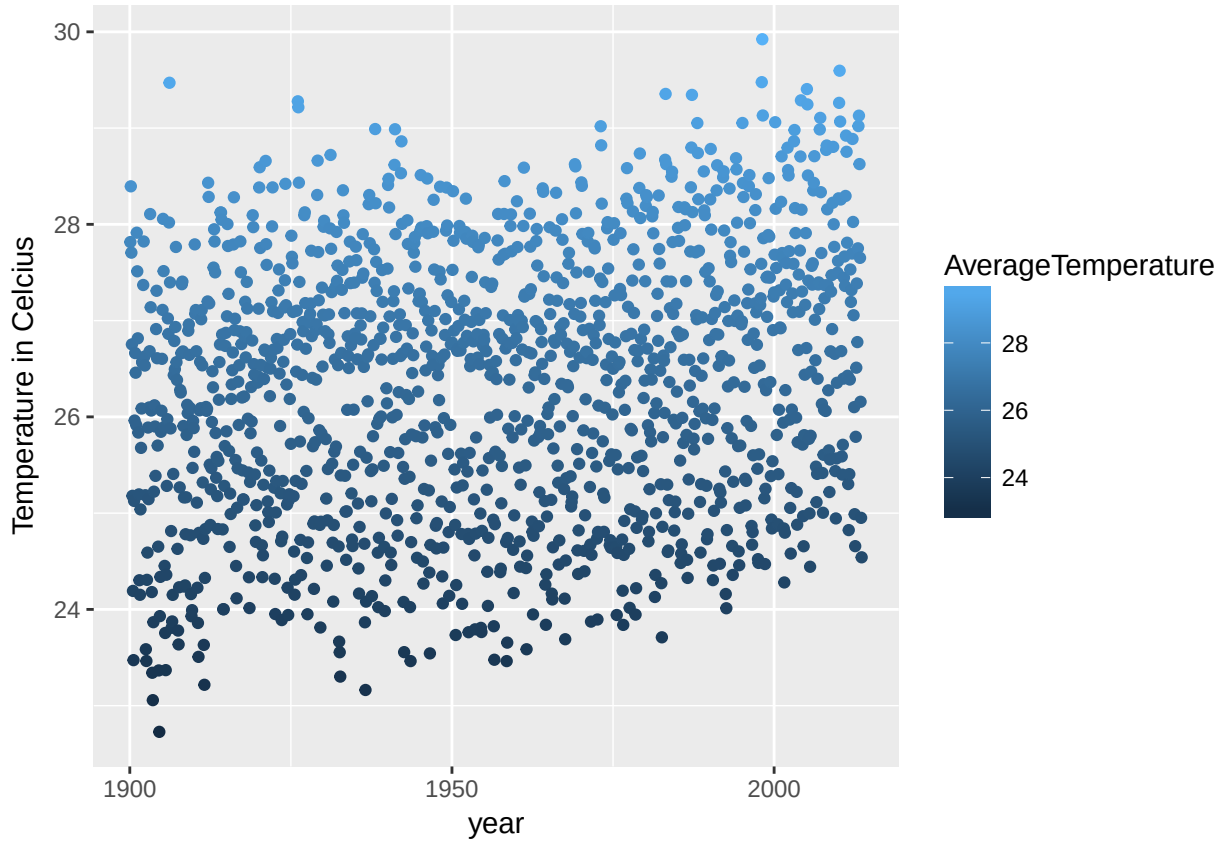
```
## [1] 1
```

Only 1. As it turns out, the last piece of data is missing.

Let's plot again.

```
ggplot(AbidjanData2, aes(dt,AverageTemperature))+geom_point(aes(color=AverageTemperature))+xlab("year")+
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Great. Let's move on to the analysis.

Let's fit an ARIMA model. I will now follow along with datascience.com/blog/introduction-to-forecasting-with-arima-in-r-learn-data-science-tutorials. They want to use the ggplot, forecast and tseries libraries, and we've already loaded ggplot.

```
library('forecast')
library('tseries')
```

As they suggest, I will make sure that the data is cleaned. They use count_ts=ts(AbidjanData2[, c('AverageTemperature')]), but I prefer different notation:

```
temp_ts=ts(AbidjanData2$AverageTemperature)
```

They then create a cleaned column:

```
AbidjanData2$cleaned_AverageTemperature= tsclean(temp_ts)
```

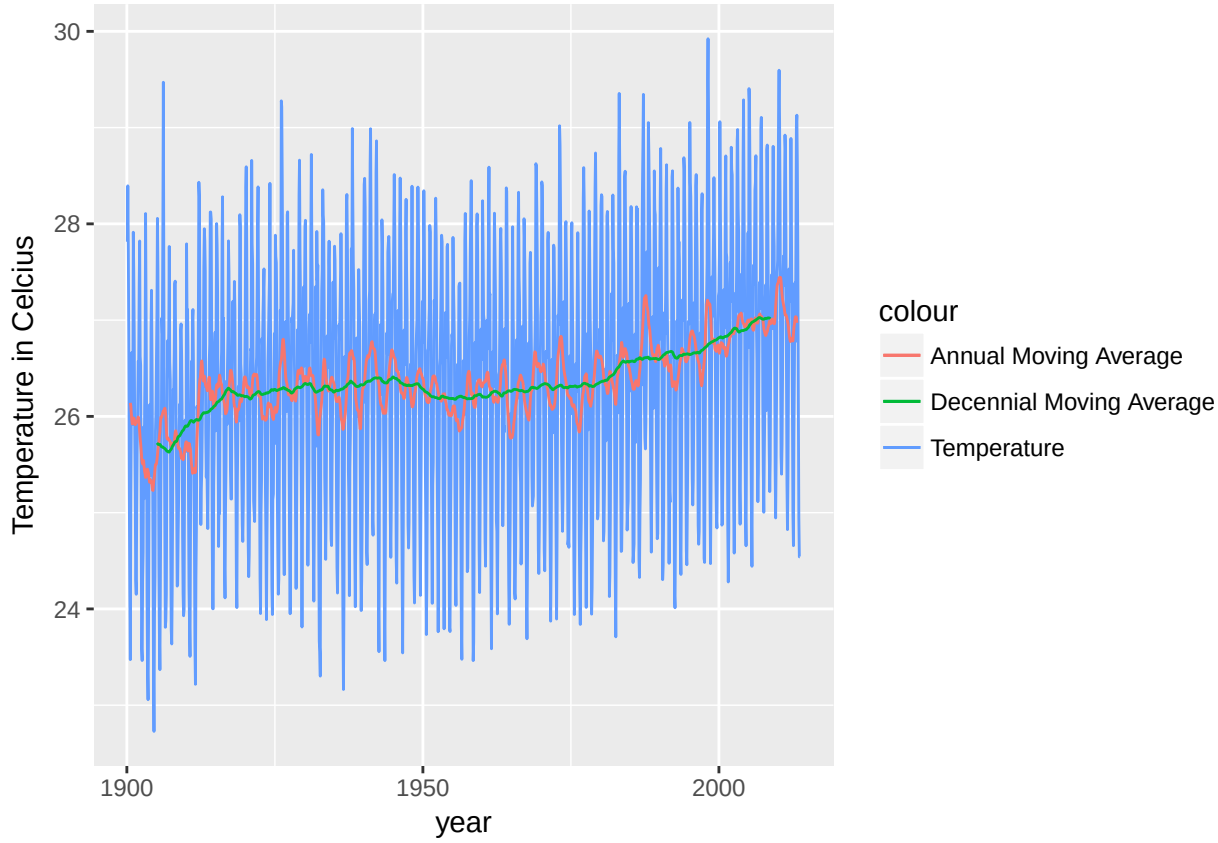
They next look at a moving average, which is what I want to look at too. R has a command: ma, which creates the moving average for us.

```
AbidjanData2$temp_ma=ma(AbidjanData2$cleaned_AverageTemperature,order=12)
AbidjanData2$temp_ma_decade=ma(AbidjanData2$cleaned_AverageTemperature,order=120)
```

Using an order 12 for the 12 months of the year. This will give our annual average. Using an order 120 for 10 years. This will give our Decennial average.

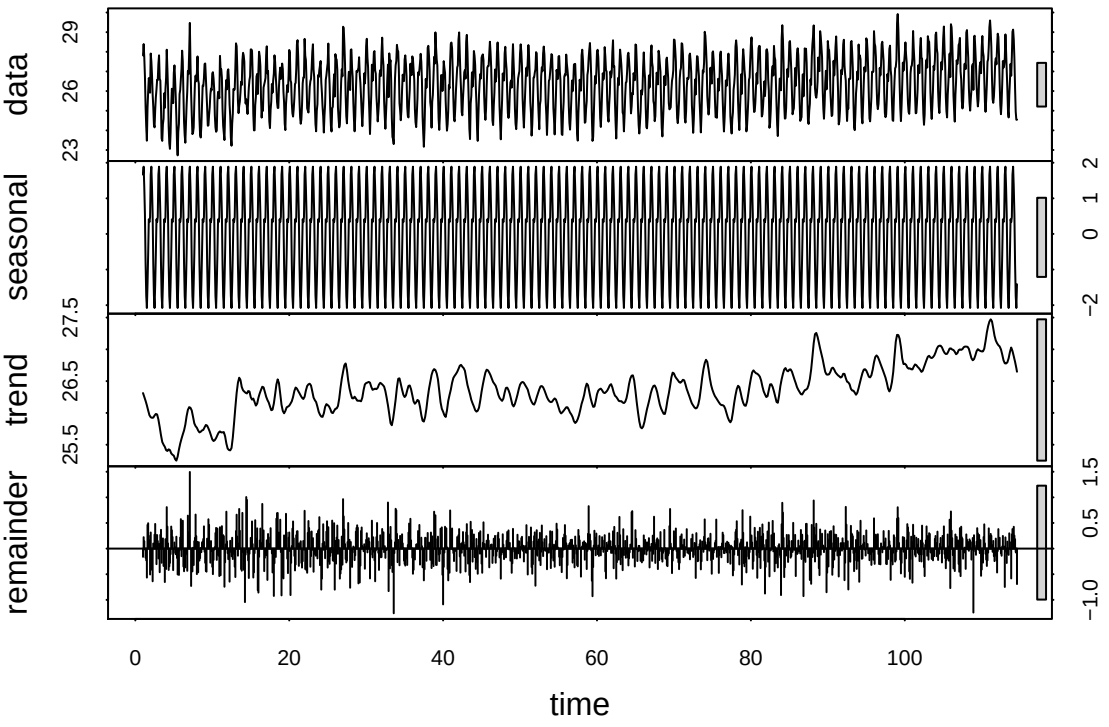
```
ggplot()+
  geom_line(data=AbidjanData2, aes(x=dt, y= cleaned_AverageTemperature, colour="Temperature"))+
  geom_line(data=AbidjanData2, aes(x=dt, y= temp_ma, colour="Annual Moving Average"))+
```

```
  geom_line(data=AbidjanData2, aes(x=dt, y= temp_ma_decade, colour="Decennial Moving Average"))+xlab("y
```



We will now calculate the seasonal component:

```
temp_ma=ts(na.omit(AbidjanData2$cleaned_AverageTemperature),frequency=12)
decomp=stl(temp_ma,s.window="periodic")
deseasonal_temp <- seasadj(decomp)
plot(decomp)
```



In our case, we have 12 observations per period, since a year is a period and we only have 1 data point per month.

They recommend checking for stationarity. To do this we will use the augmented Dickey-Fuller (ADF) test. Since temperatures depend largely on the month, and we've adjusted by using the mean temperature over the year, we shouldn't see any seasonality, this data shoud now be stationary.

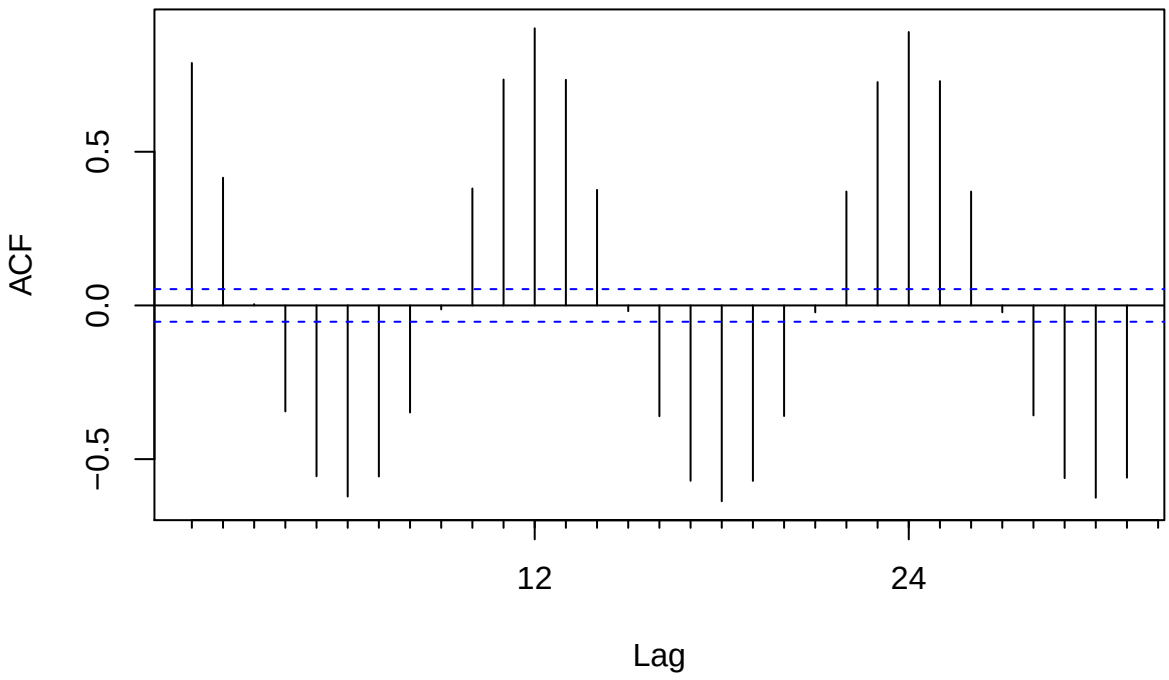
```
adf.test(temp_ma, alternative= "stationary")
```

```
## Warning in adf.test(temp_ma, alternative = "stationary"): p-value smaller
## than printed p-value
```

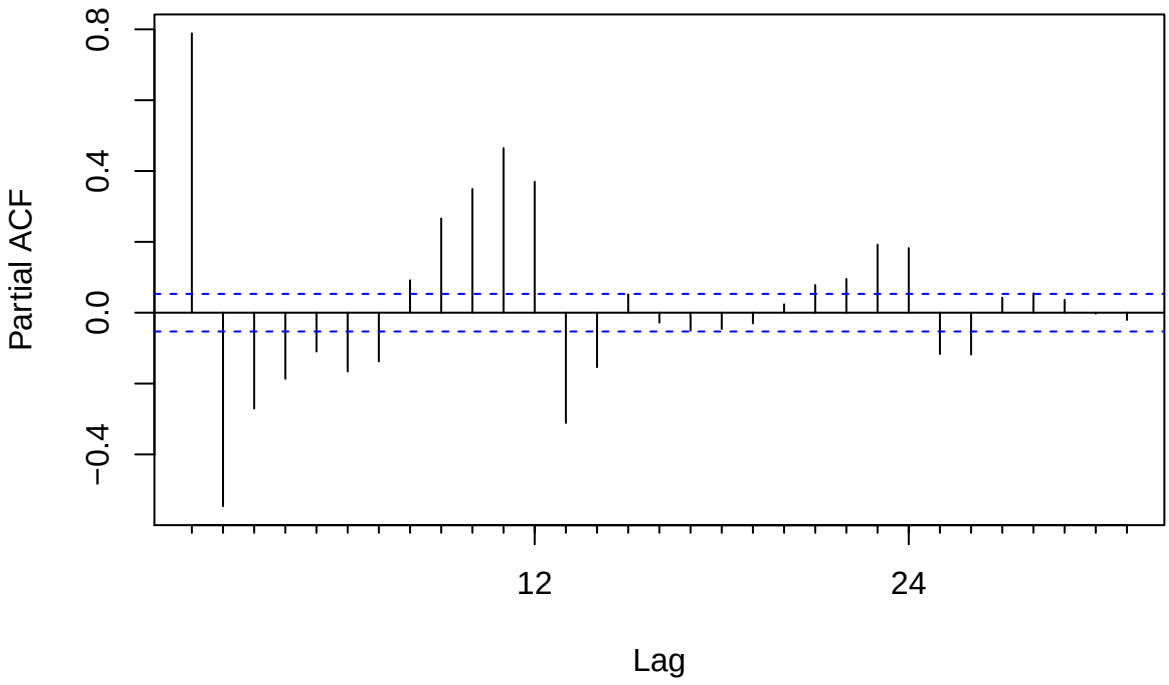
```
##
##  Augmented Dickey-Fuller Test
##
## data:  temp_ma
## Dickey-Fuller = -4.2686, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

The p-value of .01 indicates that we should assume that we reject the null hypothesis of non-stationarity. So that's good, we don't have to use differencing. If we were concerned with these, we would want to examine the AcF and Pacf for any peaks, but we already have stationary data.

```
Acf(temp_ma, main='')
```




```
Pacf(temp_ma,main='')
```



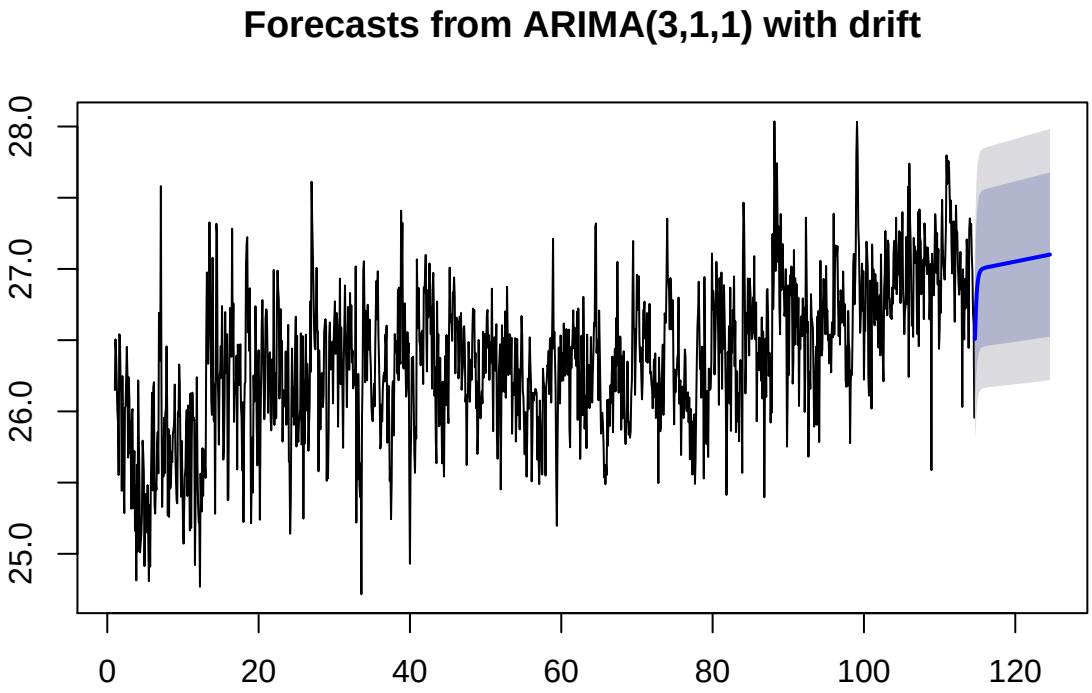
Let's try using auto.arima:

```
myarima=auto.arima(deseasonal_temp,seasonal=FALSE)
summary(myarima)
```

```
## Series: deseasonal_temp
## ARIMA(3,1,1)            with drift
##
## Coefficients:
##          ar1     ar2     ar3      ma1  drift
##       0.4089  0.1527  0.0150  -0.9846  9e-04
## s.e.  0.0276  0.0293  0.0276   0.0053  4e-04
##
## sigma^2 estimated as 0.1292:  log likelihood=-537.93
## AIC=1087.85   AICc=1087.92   BIC=1119.16
##
## Training set error measures:
##                        ME      RMSE       MAE         MPE     MAPE
## Training set -0.002858533 0.3586432 0.2737424 -0.02930644 1.040295
##                    MASE          ACF1
## Training set 0.6099427 -0.0002653413
```

and now let's plot it

```
q=forecast(myarima,h=120)
plot(q)
```


Forecasts from ARIMA(3,1,1) with drift
28.0
27.0
26.0
25.0
0
20
40
60
80
100
120