R Notebook

Change global setting

This is a knitr command that leaves NAs blank in tables.

```
options(knitr.kable.NA = '')
```

Loading my R packages

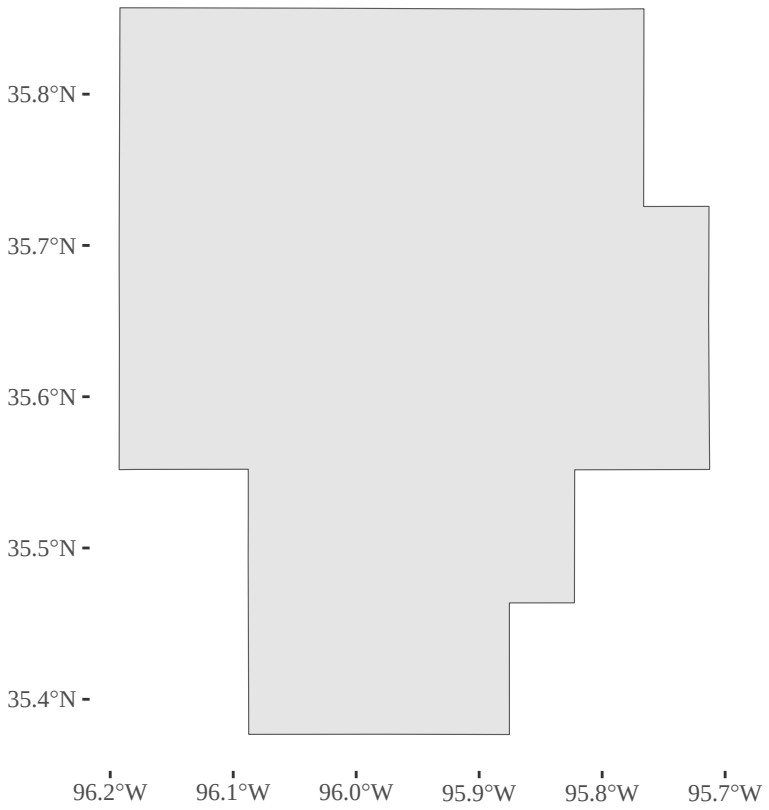
```
library(tidyverse)
library(tidync)
library(cft)
library(sf)
library(ggplot2)
library(ggthemes)
library(ggpattern)
library(magick)
library(future)
library(forecast)
library(tidytable)
library(janitor)
options(timeout = 600)
library(ggfortify)
library(reticulate)
library(osmdata)
library(osmextract)
library(changepoint)
library(weathermetrics)
library(TSstudio)
 library(ggridges)
library(plotly)
library(htmlwidgets)
library(IRdisplay)
library(knitr)
```

```
our_blue <- "#3A4E8B"
our_yellow <- "#FFD60F"
our_beige <- "#EDEDF0"
our_purple <- "#3E1471"
```

```
bb <- getbb("Okmulgee, Ok")
bb
```

```
##         min       max
## x -96.19273 -95.71259
## y  35.37669  35.85709
```

```
bb_sf <- getbb("Okmulgee, Ok", format_out = "sf_polygon")
ggplot(data=bb_sf) + geom_sf() + theme_tufte()
```




```
bbb <- bb
bbb[1,1] <- bbb[1,1] - 0.001
bbb[1,2] <- bbb[1,2] + 0.001
bbb[2,1] <- bbb[2,1] - 0.03
bbb[2,2] <- bbb[2,2] + 0.001
xlimit <- bbb[1,]
ylimit <- bbb[2,]
xmid <- xlimit[1] + diff(xlimit) / 2
ratio <- diff(xlimit) / diff(ylimit)
```

```
haskel_centroid <- st_coordinates(st_centroid(bb_sf))
lat_pt <- haskel_centroid[1,2]
lon_pt <- haskel_centroid[1,1]
```

```
web_link = "https://cida.usgs.gov/thredds/dodsC/macav2metdata_daily_future"

# Change to "https://cida.usgs.gov/thredds/catalog.html?dataset=cida.usgs.gov/macav2metdata_daily_histo

src <- tidync::tidync(web_link)
```

```
lons <- src %>% activate("D2") %>% hyper_tibble()
lats <- src %>% activate("D1") %>% hyper_tibble()
```

```
known_lon <- lons[which(abs(lons-lon_pt)==min(abs(lons-lon_pt))),]
known_lat <- lats[which(abs(lats-lat_pt)==min(abs(lats-lat_pt))),]

chosen_pt <- st_as_sf(cbind(known_lon,known_lat), coords = c("lon", "lat"), crs = "WGS84", agr = "consta
```

```
input_variables <- inputs$variable_names %>%
  filter(Variable %in% c("Maximum Relative Humidity",
                         "Minimum Relative Humidity",
                         "Maximum Temperature",
                         "Minimum Temperature",
                         "Precipitation",
                         "Eastward Wind",
                         "Northward Wind")) %>%
  filter(Scenario %in% c( "RCP 8.5")) %>%
  filter(Model %in% c(
    "Beijing Climate Center - Climate System Model 1.1",
    "Beijing Normal University - Earth System Model",
    "Canadian Earth System Model 2",
  "Centre National de Recherches Météorologiques - Climate Model 5",
  "Commonwealth Scientific and Industrial Research Organisation - Mk3.6.0",
  "Community Climate System Model 4",
  "Geophysical Fluid Dynamics Laboratory - Earth System Model 2 Generalized Ocean Layer Dynamics",
  "Geophysical Fluid Dynamics Laboratory - Earth System Model 2 Modular Ocean",
  "Hadley Global Environment Model 2 - Climate Chemistry 365 (day) ",
 "Hadley Global Environment Model 2 - Earth System 365 (day)",
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5A - Low Resolution",
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5A - Medium Resolution",
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5B - Low Resolution",
 "Institute of Numerical Mathematics Climate Model 4",
 "Meteorological Research Institute - Coupled Global Climate Model 3",
 "Model for Interdisciplinary Research On Climate - Earth System Model",
 "Model for Interdisciplinary Research On Climate - Earth System Model - Chemistry",
 "Model for Interdisciplinary Research On Climate 5",
 "Norwegian Earth System Model 1 - Medium Resolution"  )) %>%

  pull("Available variable")

head(as.data.frame(input_variables))
```

```
##                 input_variables
## 1       pr_BNU-ESM_r1i1p1_rcp85
## 2         pr_CCSM4_r6i1p1_rcp85
## 3      pr_CNRM-CM5_r1i1p1_rcp85
## 4 pr_CSIRO-Mk3-6-0_r1i1p1_rcp85
## 5       pr_CanESM2_r1i1p1_rcp85
## 6    pr_GFDL-ESM2G_r1i1p1_rcp85
```

```
# ask how many cores are available to be farmed out. I subtract one so I still have a core to use for c
n_cores <- availableCores() - 1
```

```
# set plan to take all cores except one.
plan(multisession, workers = n_cores)
```

```
out <- single_point_firehose(input_variables, known_lat, known_lon )
head(out)
```

```
MuscogeeNation <- out
save(MuscogeeNation, file = "MuscogeeNation.RData")
```

```
load(file = "MuscogeeNation.RData")
```

Organize climate data

Our requested climate data are returned from the api server as two data frames. The first data frame is the columns of data that are indexed by a time reference number and the second, which is the list translations from time reference number to actual time. We will join those tables here to make those data easier to work with. Once joined, we convert the time labels from characters to POSIX. POSIX is a special way of handling time and date data in R. We reorder the columns so that the POSIX data is in the first column. This will make it easy to later create a time series object (ts) that can go into our statistical and forecasting functions. Finally, we print the column names of the final transformed data frame to verify that we have time data in the first column and all the requested data as columns after that.

Join data with dates

```
# make the time output into it's own dataframe and change column name
available_times <- inputs[[2]]
colnames(available_times)[1] <- "time"

# left join the time data into the spatial data
haskell_posix <- MuscogeeNation %>%
  left_join(available_times, by="time")

# convert time format into POSIX, which is a format that deals with all the confusion of time and data
haskell_posix$dates <- as.POSIXct(haskell_posix$dates)
class(haskell_posix$dates)
```

```
## [1] "POSIXct" "POSIXt"
```

```
#reorder so that dates are the first column
haskell_posix <- haskell_posix[,c(130,2, 1,3:129)]
colnames(haskell_posix)
```

```
##   [1] "dates"                                "time"
##   [3] "pr_BNU-ESM_r1i1p1_rcp85"              "pr_CCSM4_r6i1p1_rcp85"
##   [5] "pr_CNRM-CM5_r1i1p1_rcp85"             "pr_CSIRO-Mk3-6-0_r1i1p1_rcp85"
##   [7] "pr_CanESM2_r1i1p1_rcp85"              "pr_GFDL-ESM2G_r1i1p1_rcp85"
##   [9] "pr_GFDL-ESM2M_r1i1p1_rcp85"           "pr_HadGEM2-ES365_r1i1p1_rcp85"
##  [11] "pr_IPSL-CM5A-LR_r1i1p1_rcp85"         "pr_IPSL-CM5A-MR_r1i1p1_rcp85"
```

```
##  [13] "pr_IPSL-CM5B-LR_r1i1p1_rcp85"       "pr_MIROC-ESM-CHEM_r1i1p1_rcp85"
##  [15] "pr_MIROC-ESM_r1i1p1_rcp85"          "pr_MIROC5_r1i1p1_rcp85"
##  [17] "pr_MRI-CGCM3_r1i1p1_rcp85"          "pr_NorESM1-M_r1i1p1_rcp85"
##  [19] "pr_inmcm4_r1i1p1_rcp85"             "rhsmax_BNU-ESM_r1i1p1_rcp85"
##  [21] "rhsmax_CNRM-CM5_r1i1p1_rcp85"       "rhsmax_CSIRO-Mk3-6-0_r1i1p1_rcp85"
##  [23] "rhsmax_CanESM2_r1i1p1_rcp85"        "rhsmax_GFDL-ESM2G_r1i1p1_rcp85"
##  [25] "rhsmax_HadGEM2-CC365_r1i1p1_rcp85"  "rhsmax_HadGEM2-ES365_r1i1p1_rcp85"
##  [27] "rhsmax_IPSL-CM5A-LR_r1i1p1_rcp85"   "rhsmax_IPSL-CM5A-MR_r1i1p1_rcp85"
##  [29] "rhsmax_MIROC-ESM-CHEM_r1i1p1_rcp85" "rhsmax_MIROC-ESM_r1i1p1_rcp85"
##  [31] "rhsmax_MIROC5_r1i1p1_rcp85"         "rhsmax_inmcm4_r1i1p1_rcp85"
##  [33] "rhsmin_BNU-ESM_r1i1p1_rcp85"        "rhsmin_CNRM-CM5_r1i1p1_rcp85"
##  [35] "rhsmin_CSIRO-Mk3-6-0_r1i1p1_rcp85"  "rhsmin_CanESM2_r1i1p1_rcp85"
##  [37] "rhsmin_GFDL-ESM2G_r1i1p1_rcp85"     "rhsmin_GFDL-ESM2M_r1i1p1_rcp85"
##  [39] "rhsmin_HadGEM2-CC365_r1i1p1_rcp85"  "rhsmin_HadGEM2-ES365_r1i1p1_rcp85"
##  [41] "rhsmin_IPSL-CM5A-LR_r1i1p1_rcp85"   "rhsmin_IPSL-CM5B-LR_r1i1p1_rcp85"
##  [43] "rhsmin_MIROC-ESM-CHEM_r1i1p1_rcp85" "rhsmin_MIROC-ESM_r1i1p1_rcp85"
##  [45] "rhsmin_MIROC5_r1i1p1_rcp85"         "rhsmin_MRI-CGCM3_r1i1p1_rcp85"
##  [47] "tasmax_BNU-ESM_r1i1p1_rcp85"        "tasmax_CCSM4_r6i1p1_rcp85"
##  [49] "tasmax_CNRM-CM5_r1i1p1_rcp85"       "tasmax_CSIRO-Mk3-6-0_r1i1p1_rcp85"
##  [51] "tasmax_CanESM2_r1i1p1_rcp85"        "tasmax_GFDL-ESM2G_r1i1p1_rcp85"
##  [53] "tasmax_GFDL-ESM2M_r1i1p1_rcp85"     "tasmax_HadGEM2-CC365_r1i1p1_rcp85"
##  [55] "tasmax_HadGEM2-ES365_r1i1p1_rcp85"  "tasmax_IPSL-CM5A-LR_r1i1p1_rcp85"
##  [57] "tasmax_IPSL-CM5A-MR_r1i1p1_rcp85"   "tasmax_IPSL-CM5B-LR_r1i1p1_rcp85"
##  [59] "tasmax_MIROC-ESM-CHEM_r1i1p1_rcp85" "tasmax_MIROC-ESM_r1i1p1_rcp85"
##  [61] "tasmax_MIROC5_r1i1p1_rcp85"         "tasmax_MRI-CGCM3_r1i1p1_rcp85"
##  [63] "tasmax_NorESM1-M_r1i1p1_rcp85"      "tasmin_BNU-ESM_r1i1p1_rcp85"
##  [65] "tasmin_CCSM4_r6i1p1_rcp85"          "tasmin_CNRM-CM5_r1i1p1_rcp85"
##  [67] "tasmin_CanESM2_r1i1p1_rcp85"        "tasmin_GFDL-ESM2G_r1i1p1_rcp85"
##  [69] "tasmin_GFDL-ESM2M_r1i1p1_rcp85"     "tasmin_HadGEM2-CC365_r1i1p1_rcp85"
##  [71] "tasmin_HadGEM2-ES365_r1i1p1_rcp85"  "tasmin_IPSL-CM5A-LR_r1i1p1_rcp85"
##  [73] "tasmin_IPSL-CM5A-MR_r1i1p1_rcp85"   "tasmin_IPSL-CM5B-LR_r1i1p1_rcp85"
##  [75] "tasmin_MIROC-ESM-CHEM_r1i1p1_rcp85" "tasmin_MIROC-ESM_r1i1p1_rcp85"
##  [77] "tasmin_MIROC5_r1i1p1_rcp85"         "tasmin_MRI-CGCM3_r1i1p1_rcp85"
##  [79] "tasmin_inmcm4_r1i1p1_rcp85"         "uas_BNU-ESM_r1i1p1_rcp85"
##  [81] "uas_CCSM4_r6i1p1_rcp85"             "uas_CNRM-CM5_r1i1p1_rcp85"
##  [83] "uas_CSIRO-Mk3-6-0_r1i1p1_rcp85"     "uas_CanESM2_r1i1p1_rcp85"
##  [85] "uas_GFDL-ESM2G_r1i1p1_rcp85"        "uas_GFDL-ESM2M_r1i1p1_rcp85"
##  [87] "uas_HadGEM2-CC365_r1i1p1_rcp85"     "uas_HadGEM2-ES365_r1i1p1_rcp85"
##  [89] "uas_IPSL-CM5A-LR_r1i1p1_rcp85"      "uas_IPSL-CM5B-LR_r1i1p1_rcp85"
##  [91] "uas_MIROC-ESM-CHEM_r1i1p1_rcp85"    "uas_MIROC-ESM_r1i1p1_rcp85"
##  [93] "uas_MIROC5_r1i1p1_rcp85"            "uas_MRI-CGCM3_r1i1p1_rcp85"
##  [95] "uas_NorESM1-M_r1i1p1_rcp85"         "uas_inmcm4_r1i1p1_rcp85"
##  [97] "vas_BNU-ESM_r1i1p1_rcp85"           "vas_CCSM4_r6i1p1_rcp85"
##  [99] "vas_CNRM-CM5_r1i1p1_rcp85"          "vas_CSIRO-Mk3-6-0_r1i1p1_rcp85"
## [101] "vas_CanESM2_r1i1p1_rcp85"           "vas_GFDL-ESM2G_r1i1p1_rcp85"
## [103] "vas_GFDL-ESM2M_r1i1p1_rcp85"        "vas_HadGEM2-CC365_r1i1p1_rcp85"
## [105] "vas_HadGEM2-ES365_r1i1p1_rcp85"     "vas_IPSL-CM5B-LR_r1i1p1_rcp85"
## [107] "vas_MIROC-ESM_r1i1p1_rcp85"         "vas_MIROC5_r1i1p1_rcp85"
## [109] "vas_NorESM1-M_r1i1p1_rcp85"         "vas_inmcm4_r1i1p1_rcp85"
## [111] "pr_HadGEM2-CC365_r1i1p1_rcp85"      "rhsmax_IPSL-CM5B-LR_r1i1p1_rcp85"
## [113] "rhsmax_MRI-CGCM3_r1i1p1_rcp85"      "rhsmin_IPSL-CM5A-MR_r1i1p1_rcp85"
## [115] "tasmax_inmcm4_r1i1p1_rcp85"         "tasmin_CSIRO-Mk3-6-0_r1i1p1_rcp85"
## [117] "tasmin_NorESM1-M_r1i1p1_rcp85"      "vas_IPSL-CM5A-LR_r1i1p1_rcp85"
## [119] "vas_IPSL-CM5A-MR_r1i1p1_rcp85"      "vas_MIROC-ESM-CHEM_r1i1p1_rcp85"
```

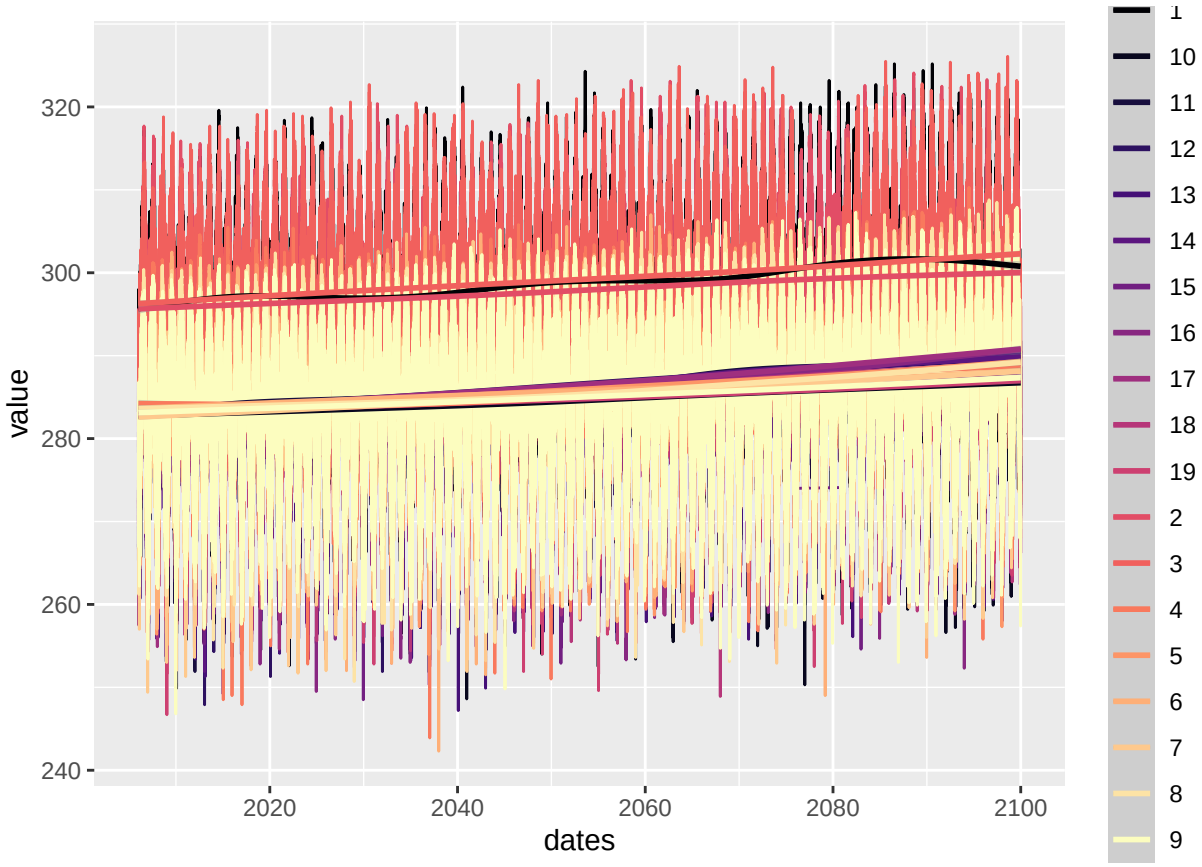
```
## [121] "vas_MRI-CGCM3_r1i1p1_rcp85"          "pr_bcc-csm1-1_r1i1p1_rcp85"
## [123] "rhsmax_bcc-csm1-1_r1i1p1_rcp85"      "rhsmin_bcc-csm1-1_r1i1p1_rcp85"
## [125] "rhsmin_inmcm4_r1i1p1_rcp85"          "tasmax_bcc-csm1-1_r1i1p1_rcp85"
## [127] "tasmin_bcc-csm1-1_r1i1p1_rcp85"      "uas_bcc-csm1-1_r1i1p1_rcp85"
## [129] "vas_bcc-csm1-1_r1i1p1_rcp85"         "geometry"
```

```
df_min_temp <- haskell_posix %>%
  st_drop_geometry() %>%
  select(dates, dates,  which(startsWith(colnames(MuscogeeNation), "tasmin"))) %>%
  gather(key = "variable", value = "value", -dates)

df_min_temp <- df_min_temp[which(df_min_temp$value > 200), ]
df_min_temp$variable <- as.character(as.numeric(as.factor(df_min_temp$variable)))

colnames(df_min_temp)
```

```
## [1] "dates"    "variable" "value"
```

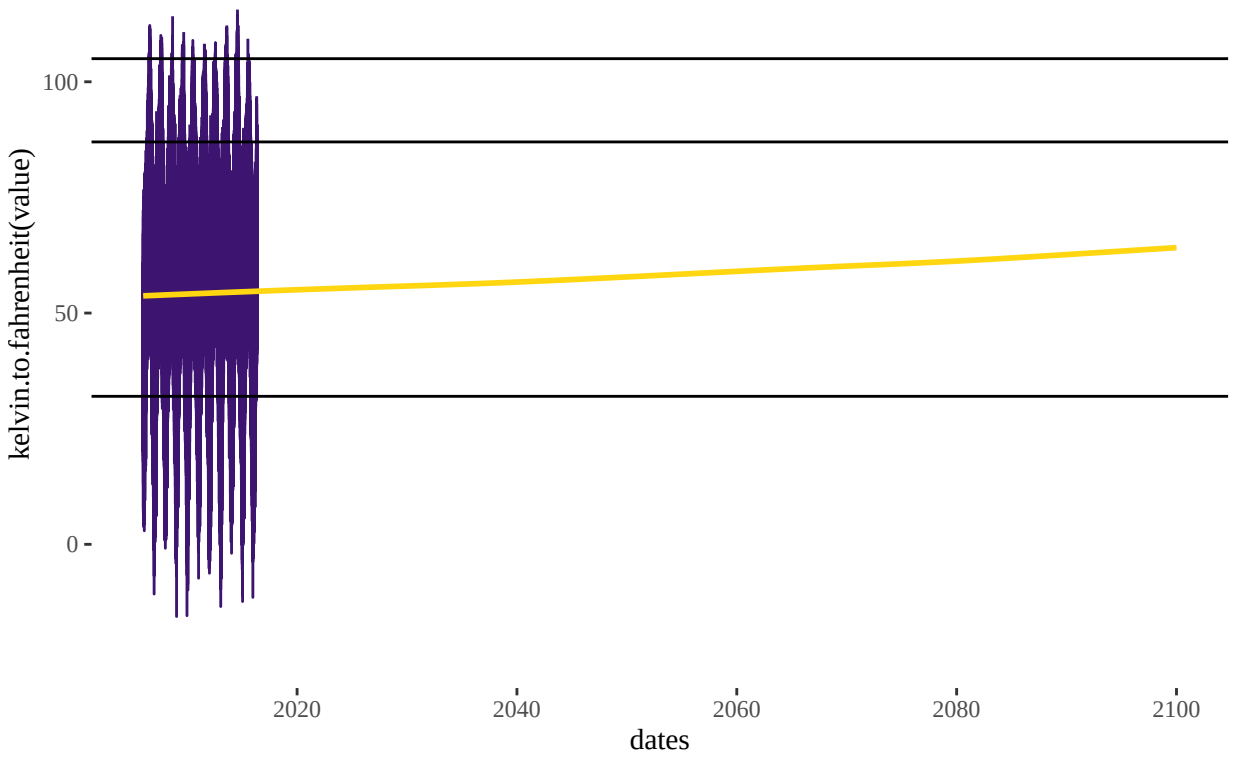
2. Plot data If we plot our filtered minimum temperature data, we see a chaotic mess because all 18 models are represented on the same graph. We probably want to consider all of the models together.

```
all_climate_projections <- ggplot(data= df_min_temp, aes(x = dates, y = value, color = variable)) +
  geom_line()+
  geom_smooth() +
  scale_colour_viridis_d(option="A")

all_climate_projections
```



3. Plot as ensemble and fit a general additive model (GAM) to those data If we plot those same data again without the model distinction, we see the ensemble of all 18 climate models. These data were standardized during downscaling, so they are directly comparable now without any more fuss. Notice that these data are in Kelvin. You should alway complet all of your analyses in Kelvin and only translate to Celsius or Fahrenheit for final figures for a general audience. We apply a general additive model to the data to see what the basic trend looks like. It looks like we should expect a steady increasing in temperature from 281K to 289K between now and 2099. This line doesn't offer much refinement to our existing expectation. Visually, it sits below an area of high data density and it doesn't help us explain any of the season variation we see in the data. This fit makes us want something better.

```
ensemble_climate_projections <- ggplot(data= df_min_temp, aes(x = dates, y = kelvin.to.fahrenheit(value)
  geom_line(color=our_purple)+
  geom_smooth(color=our_yellow) + #This applies a GAM (general additive model) to the data to find a tr
  theme_tufte() +
  geom_hline(yintercept=32)+
  geom_hline(yintercept=87)+
  geom_hline(yintercept=105)

ensemble_climate_projections
```

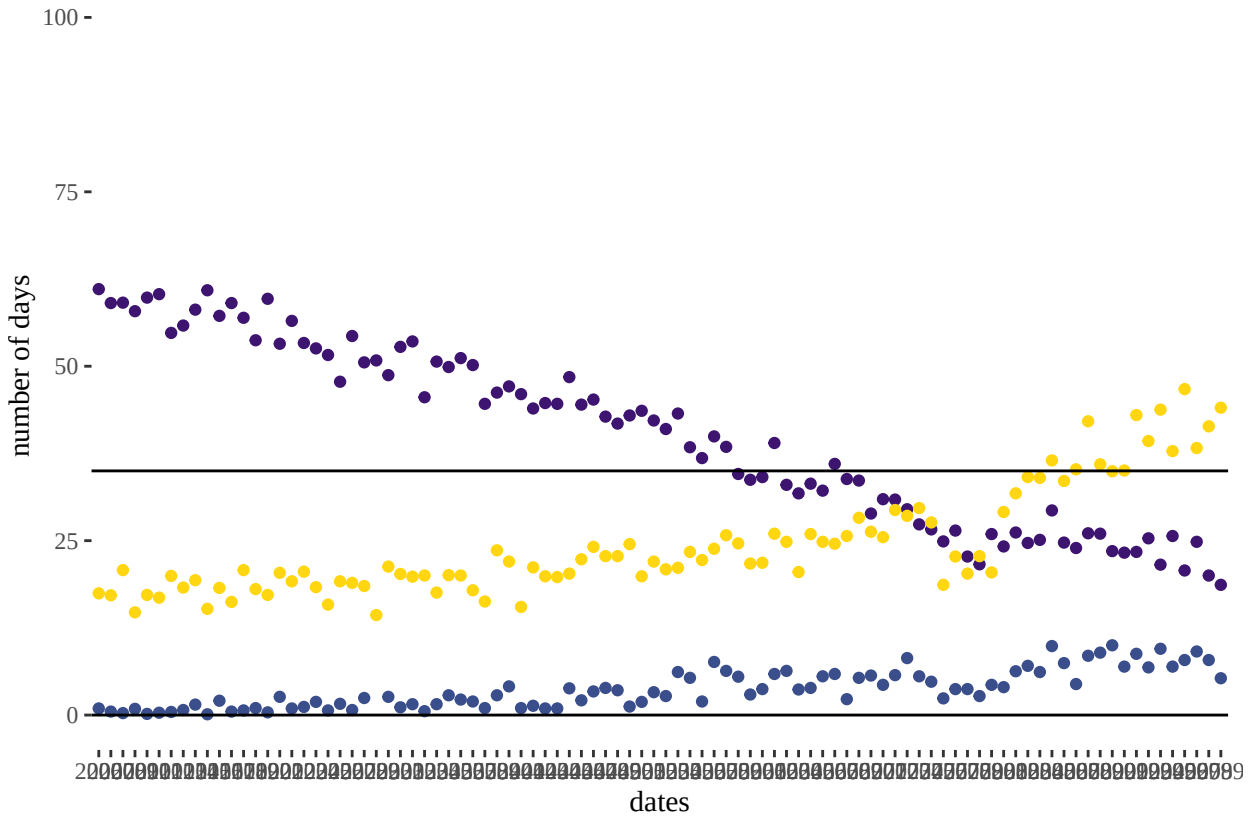


Number of days below freezing. Each generation will have one less month of freeze.

Highest daily low. Each generation experiences an additional 10 days per year with a daily low above 87 degrees F. Three generations see a lot of change. for those trying to accomplish 8 generation planning, modern science can only project less than half way there.

```
df_min_temp_F <- df_min_temp
df_min_temp_F$value <- kelvin.to.fahrenheit(df_min_temp_F$value)
df_min_temp_F$dates <- format(df_min_temp_F$dates, format = "%Y")

below_freezing <- df_min_temp_F %>%
  filter(value <=32) %>%
  count(dates) %>%
  mutate(cold_counts = n/18)


high_lows <- df_min_temp_F %>%
  filter(value >=87) %>%
  count(dates) %>%
  mutate(high_lows_counts = n/18)


 scorching_nights <- df_min_temp_F %>%
  filter(value >=105) %>%
   count(dates) %>%
  mutate(scorching_nights_counts = n/18)


  ggplot(data = below_freezing, aes(x = dates, y=cold_counts)) +
```

```
          geom_point(color=our_purple)+
  geom_point(data = high_lows, aes(x = dates, y=high_lows_counts),color=our_yellow)+
  geom_point(data = scorching_nights, aes(x = dates, y=scorching_nights_counts),color=our_blue)+
theme_tufte() +
  ylim(0,100) +
  ylab("number of days") +
geom_hline(yintercept=0)+
geom_hline(yintercept=35)
```



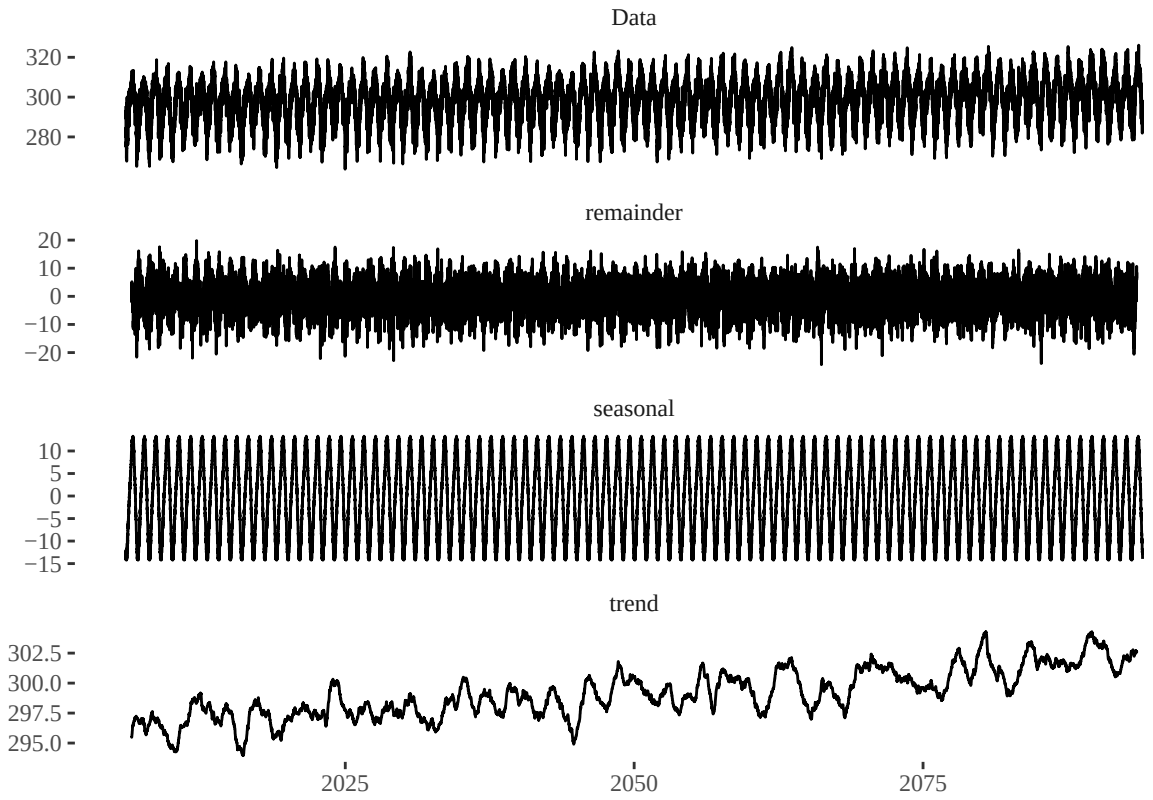
Purple is the number of days per year that drop below freezing at any point. You see these decline significantly, losing nearly a month per year of days that drop below freezing. Yellow is the number of days where the lowest nightime temperature never drops below 87, which is a temperature when people start experiencing health problems or sleeping issues. Those who can will run air conditioning all night. The blue dots show the emergency of scorching nights, where the nightime low temperatre never drops below 105 F, which is currently the historic maximum low temperature for the area. People die when its this hot.

```
min_temp <- ts(df_min_temp[,3], frequency = 365, start = c(2006, 1), end = c(2093,365))
```

4. Decompose the time series using fourier analysis. Fourier analyses are a convenient way to decompose repeating waves and are a mainstay of time series analyses. The analysis presented here finds the seasonal harmonic in the data and subtracts that harmonic from the data to show the difference between trend and noise. We start by converting our data into a ts object and passes that ts object to the decompose() function. When we plot that deconstruction, we see that the resultant trend line is much more nuanced than our previous fit.

Here is the GAM fit

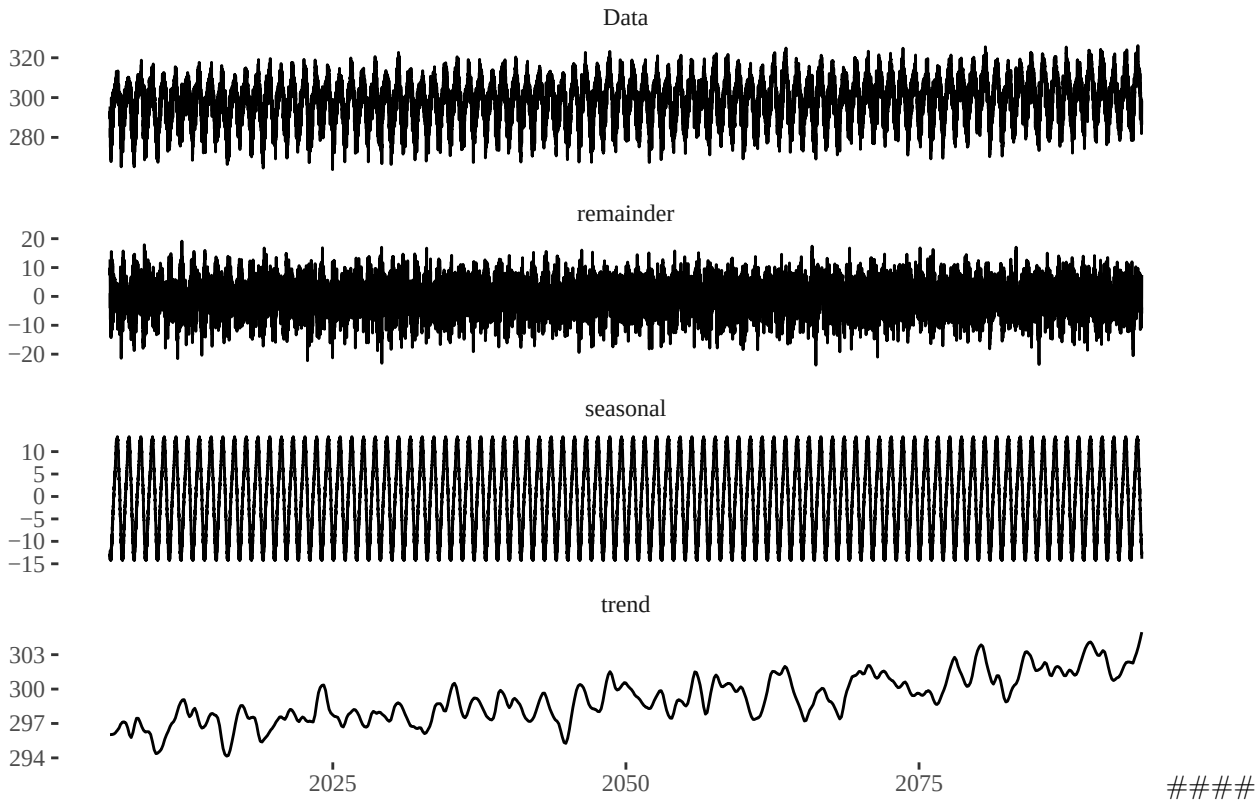
```
min_temp %>%
  decompose() %>%
  autoplot() + theme_tufte()
```



We can also use a different decompositions model, here the STL model, which is a loess model for time series data. The STL

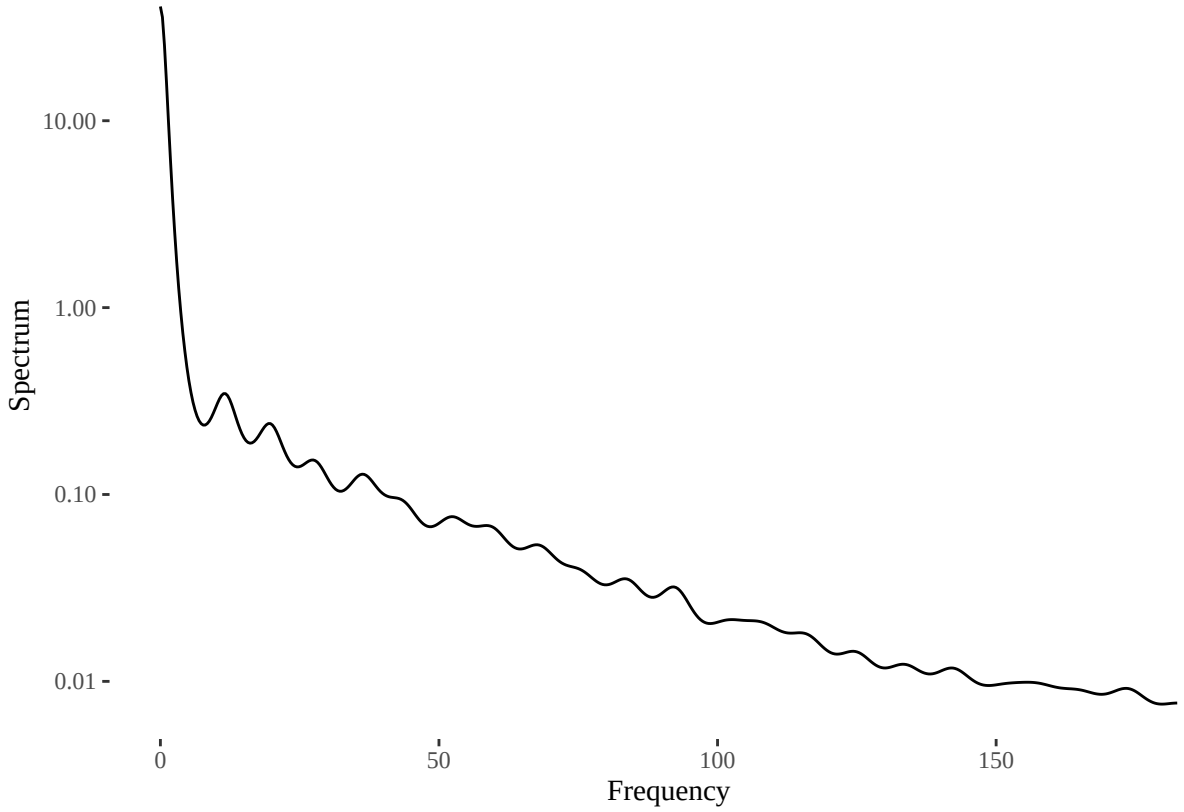
Here is the loess fit

```
min_temp %>%
 stl(s.window = "periodic") %>%
  autoplot() + theme_tufte()
```



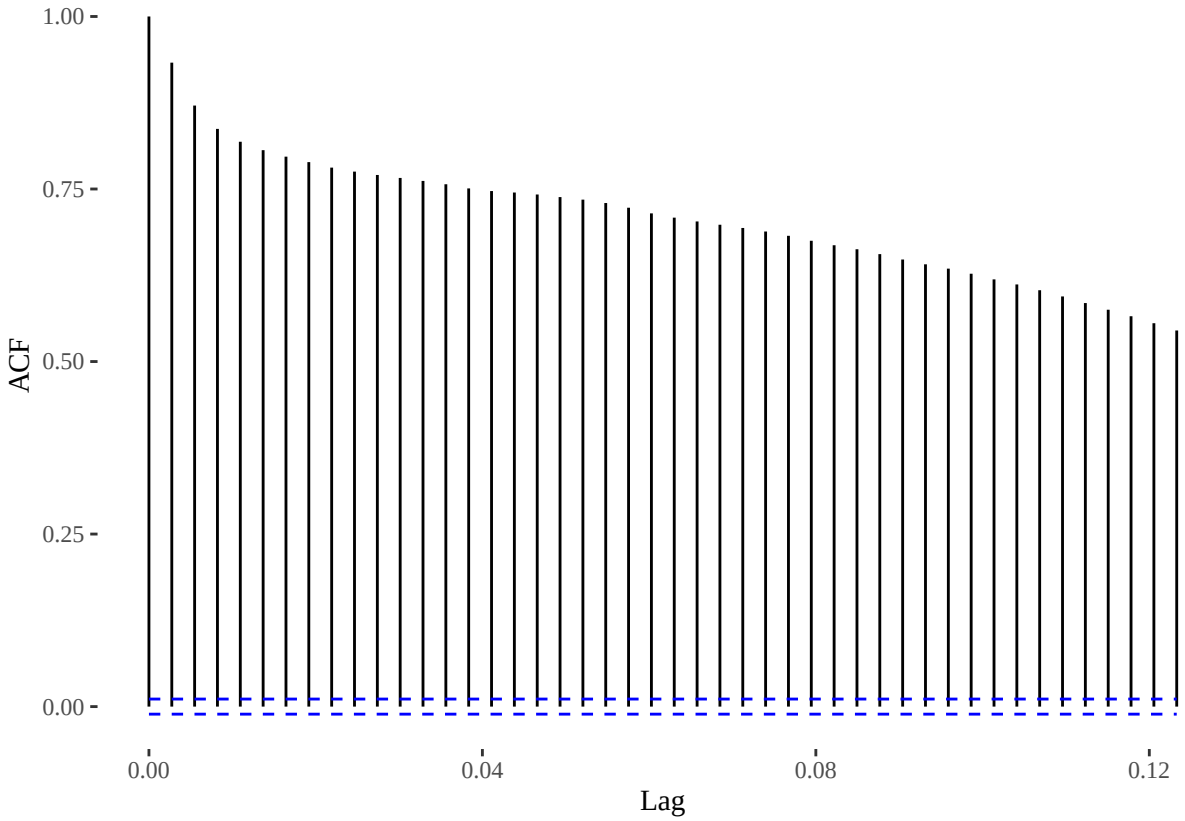
Estimate Spectral Density of a Time Series This is the fourier spectral breakdown for the decomposition. You can see the strong first harmonic that is easy to pull out and makes the decomposition of this data go relatively fast for this size of dataset.

```
autoplot(spec.ar(min_temp, plot = FALSE))+ theme_tufte()
```



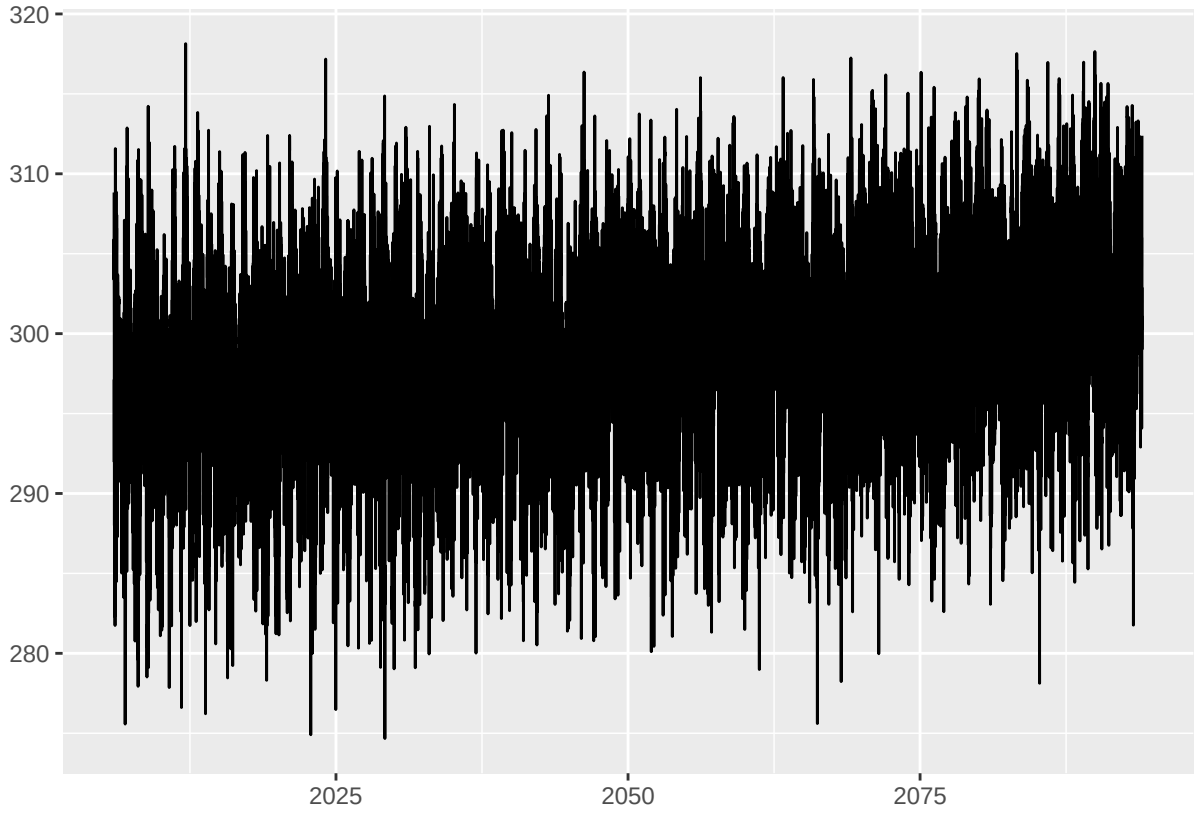
Verify that our use of an additive model was appropriate. Time series can come in a couple different flavors. The two that are common in decomposition analyses are additive and multiplicative harmonics. We made the assumption that temperature increase was additive and we can validate that assumption here with an autocorrelation function (ACF), which is the coefficient of correlation between two values in a time series. The results show the gradual decline of ACF along the time series. This is the result of our one-step-ahead climate predictions that use the previous year to predict the next year. this means that there is strong correlation between adjoining years, but that the correlation degrades in one direction from no into the future. You see that years in the near future are tightly coupled with each other but that covariance degrades over time so we have less confidence in the end of the time series than we do about the beginning. A multiplicative relationship would inflate rapidly over time and show an exponential relationship here. You might expect multiplicative relationships with time series of demographic growth, stock trends, or social media 'likes' because those all present mechanisms that can multiply rather than add.

```
autoplot(acf(min_temp, plot = FALSE))+ theme_tufte()
```



Forecast models We need to make an important distinctions between the different types of forecasting happening in this analysis. Our climate data are forecast into the future using global mechanistic models simulating the collision of air molecules and the accumulation of gasses that change climate and weather patterns over decades. Those models produce the data we download and use as raw data to describe our local future. The forecasting we're about to do, we're looking for statistical trends contained within that data. There is no natural or environmental mechanism in this forecasting. The arima forecast is a statistical forecasting method that fits a model to the data using the same decomposition method we described above (GAN) and then calculates an acceptable forecast window based on the predictability of the trend relative to the data.

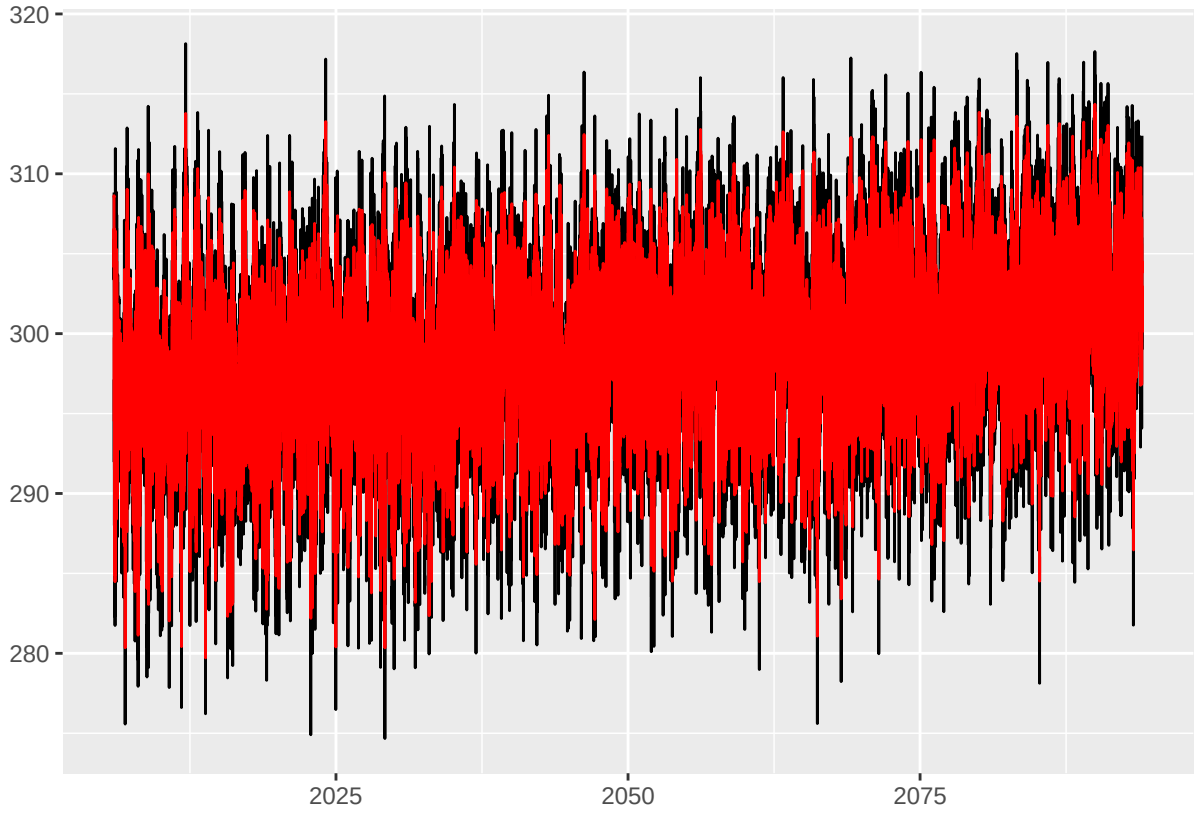
```
seasonally_adjusted_min_temp <- min_temp %>% stl(s.window='periodic') %>% seasadj()
min_plot <- autoplot(seasonally_adjusted_min_temp)
#+
 # theme_tufte() +
  #geom_smooth(col=our_yellow)

min_plot
```



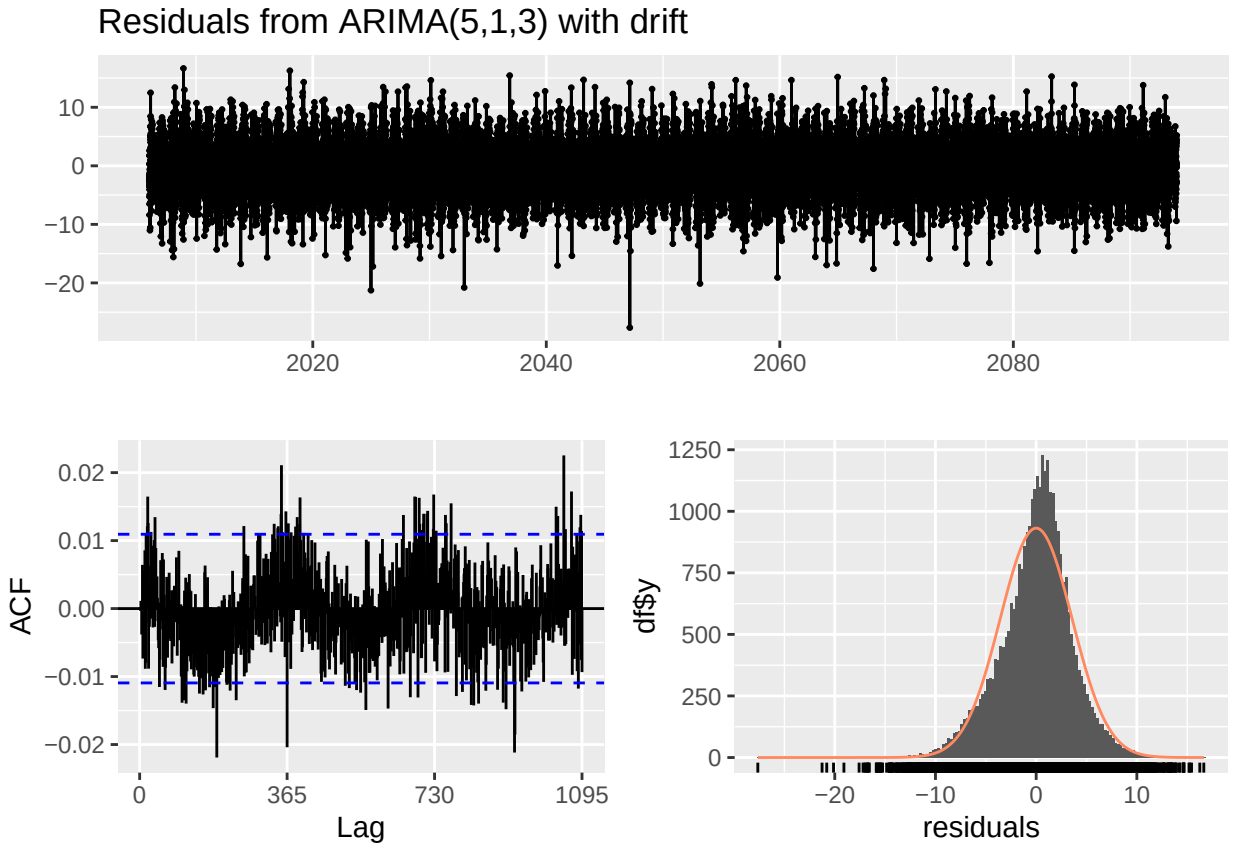

```
arima_min_temp <- auto.arima(seasonally_adjusted_min_temp)
arima_min_temp
```

```
## Series: seasonally_adjusted_min_temp
## ARIMA(5,1,3) with drift
##
## Coefficients:
##          ar1     ar2      ar3     ar4      ar5      ma1      ma2     ma3
##       0.4313  0.7141  -0.4887  0.1344  -0.0131  -0.6090  -0.9579  0.5796
## s.e.  0.1742  0.0418   0.1068  0.0212   0.0140   0.1741   0.0274  0.1691
##        drift
##       0.0002
## s.e.  0.0011
##
## sigma^2 = 12.7:  log likelihood = -86382.61
## AIC=172785.2   AICc=172785.2   BIC=172869
```

```
bullseye <- autoplot(arima_min_temp)
bullseye
```

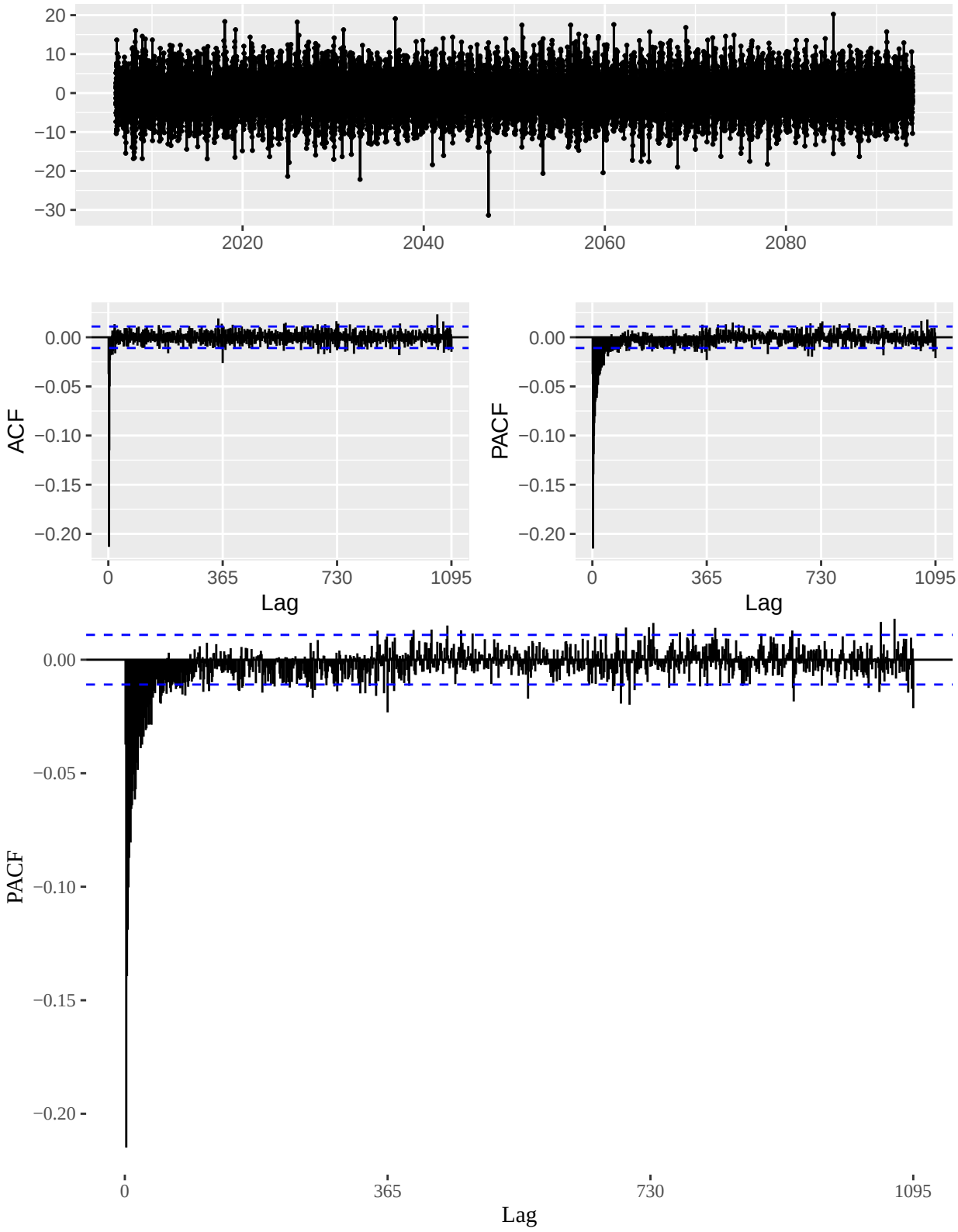



```
checkresiduals(arima_min_temp)
```

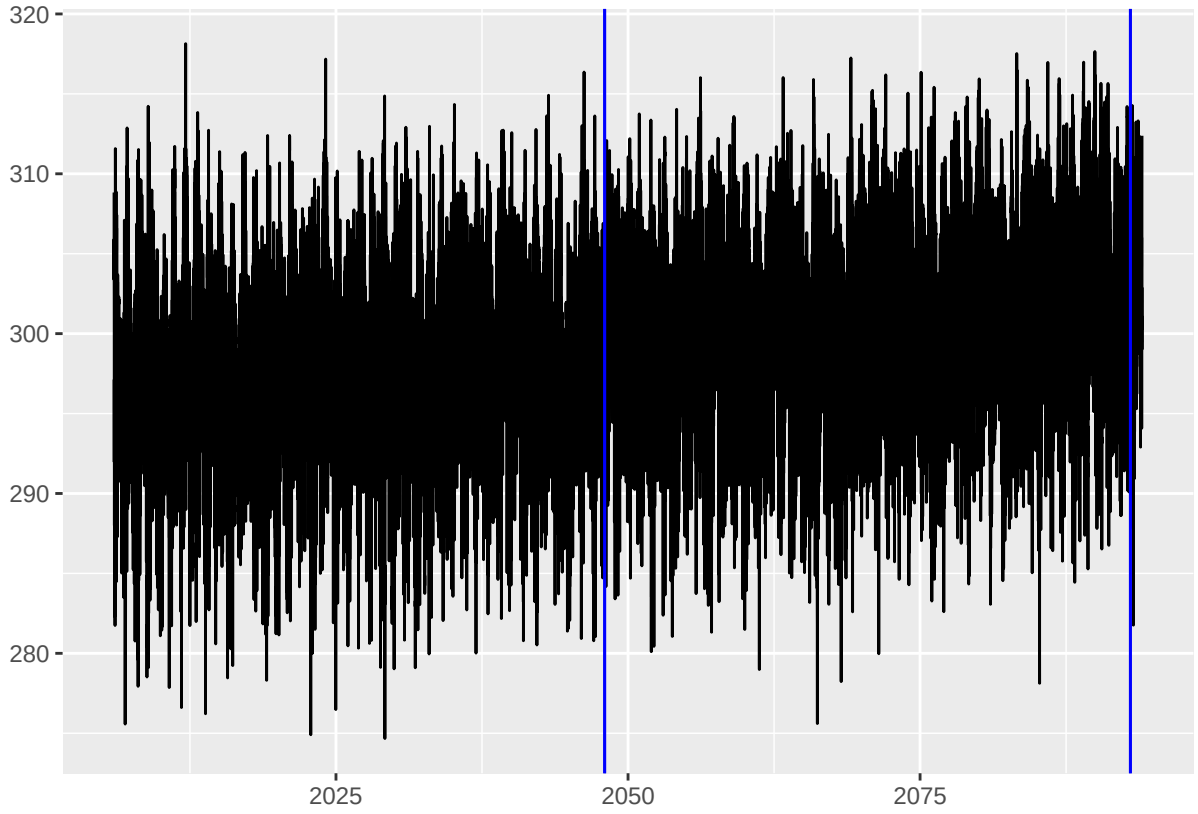



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(5,1,3) with drift
## Q* = 967.66, df = 722, p-value = 2.224e-09
##
## Model df: 8.   Total lags used: 730
```

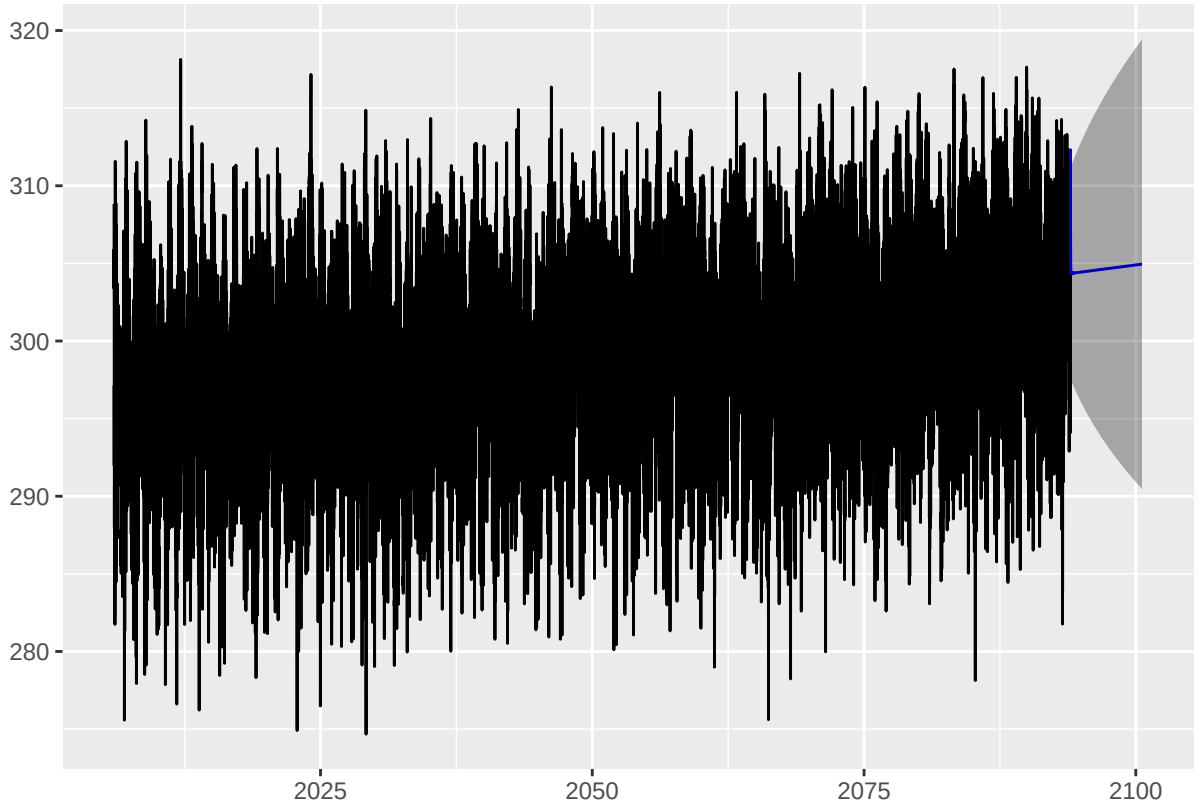
```
seasonally_adjusted_min_temp %>% diff() %>% ggtsdisplay(main="") + theme_tufte()
```




```
autoplot(cpt.meanvar(seasonally_adjusted_min_temp), cpt.colour = 'blue', cpt.linetype = 'solid')
```




```
arima_min_temp %>% forecast(h=2400) %>% autoplot()
```

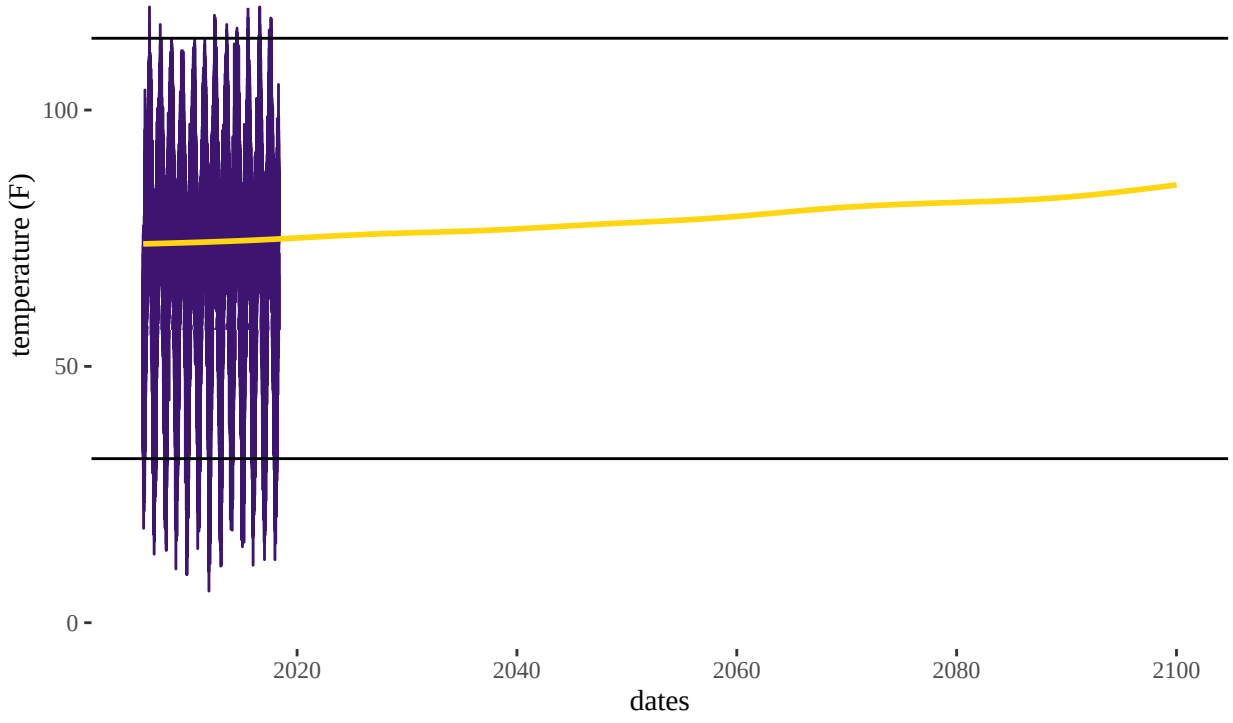


```
df_max_temp <- haskell_posix %>%
  st_drop_geometry() %>%
  select(dates, dates,  which(startsWith(colnames(MuscogeeNation), "tasmax"))) %>%
  gather(key = "variable", value = "value", -dates)

df_max_temp <- df_max_temp[which(df_max_temp$value > 200), ]
df_max_temp$variable <- as.character(as.numeric(as.factor(df_max_temp$variable)))

#df_min_temp_TS <- as.xts(df_min_temp)
colnames(df_max_temp)
```

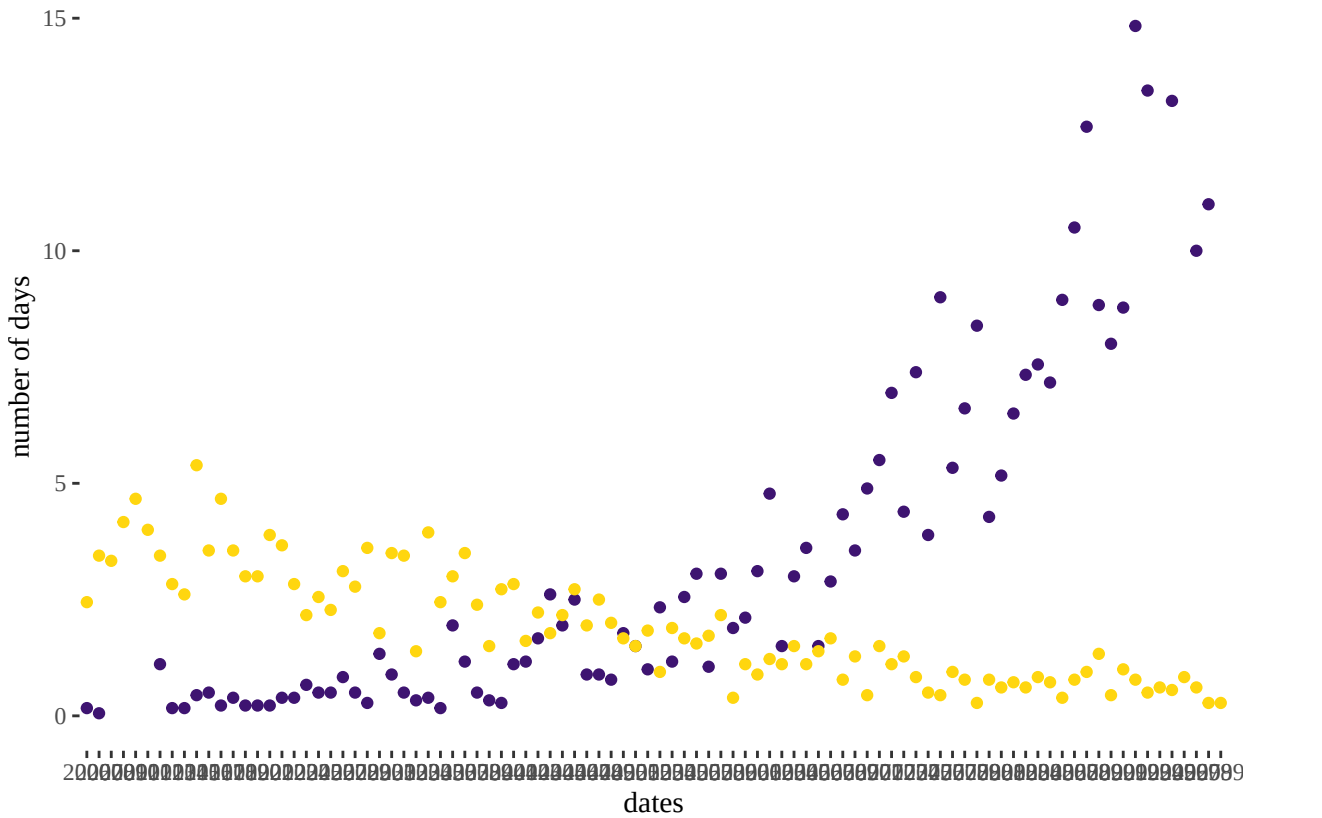
Maximum temperature

```
## [1] "dates"    "variable" "value"
```

```
ensemble_climate_projections <- ggplot(data= df_max_temp, aes(x = dates, y = kelvin.to.fahrenheit(value)
  geom_line(color=our_purple)+
  geom_smooth(color=our_yellow) + #This applies a GAM (general additive model) to the data to find a tr
  theme_tufte() +
  geom_hline(yintercept=114)+
  geom_hline(yintercept=32) +
  ylab("temperature (F)")

ensemble_climate_projections
```



```
df_max_temp_F <- df_max_temp
df_max_temp_F$value <- kelvin.to.fahrenheit(df_max_temp_F$value)
df_max_temp_F$dates <- format(df_max_temp_F$dates, format = "%Y")

beyond_max_temp <- df_max_temp_F %>%
  filter(value >=114) %>%
  count(dates) %>%
  mutate(beyond_max = n/18)


max_below_freezing <- df_max_temp_F %>%
  filter(value <=32) %>%
  count(dates) %>%
  mutate(below_freezing = n/18)


  ggplot(data = beyond_max_temp, aes(x = dates, y=beyond_max)) +
           geom_point(color=our_purple)+
    geom_point(data = max_below_freezing, aes(x = dates, y=below_freezing), color=our_yellow)+
  theme_tufte() +
    ylim(0,15) +
    ylab("number of days")
```

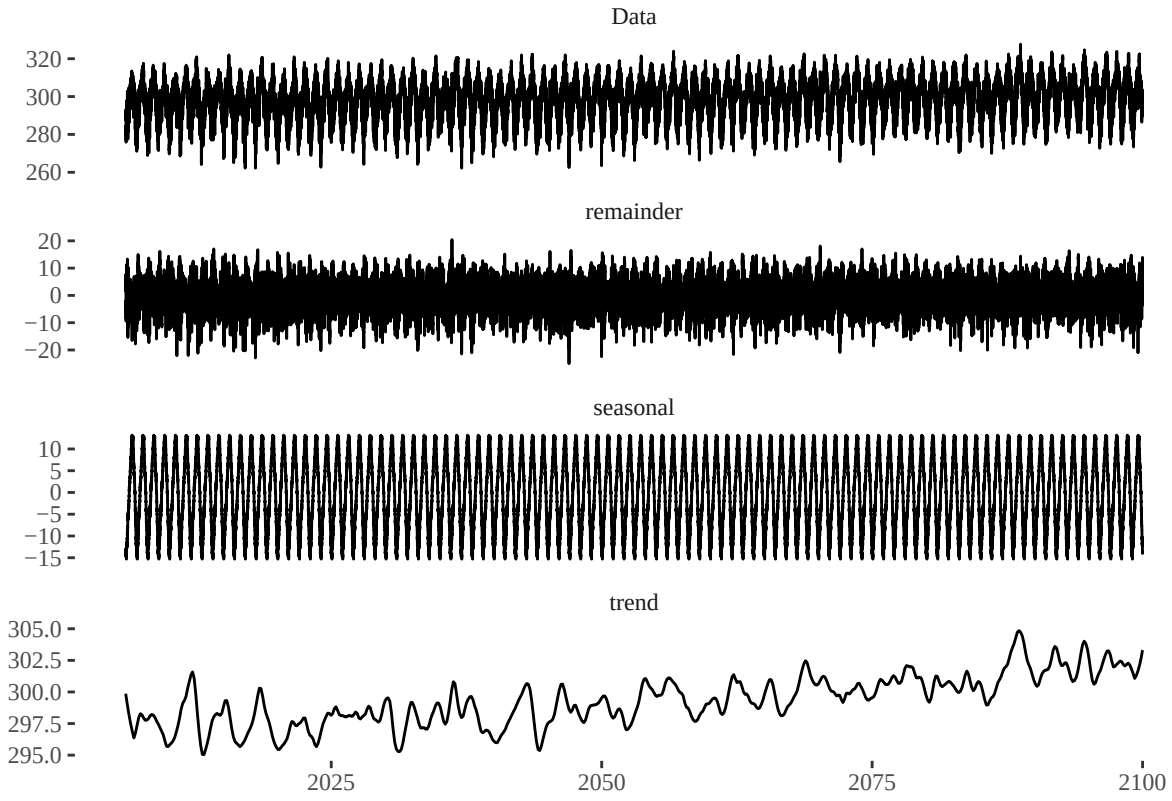


Purple is the the number of days where the max temperature is above 114 F, which is the current all-time max temperature as of now. Yellow is the number of days where the maximum temperature never gets above freezing. You see the rapid increase in record breaking highs and the slow extinction of days that never go above freezing.

```
max_temp <- ts(df_max_temp[,3], frequency = 365, start = c(2006, 1), end = c(2099,365))
```

```
seasonally_adjusted_max_temp <- max_temp %>% stl(s.window='periodic') %>% seasadj()
```

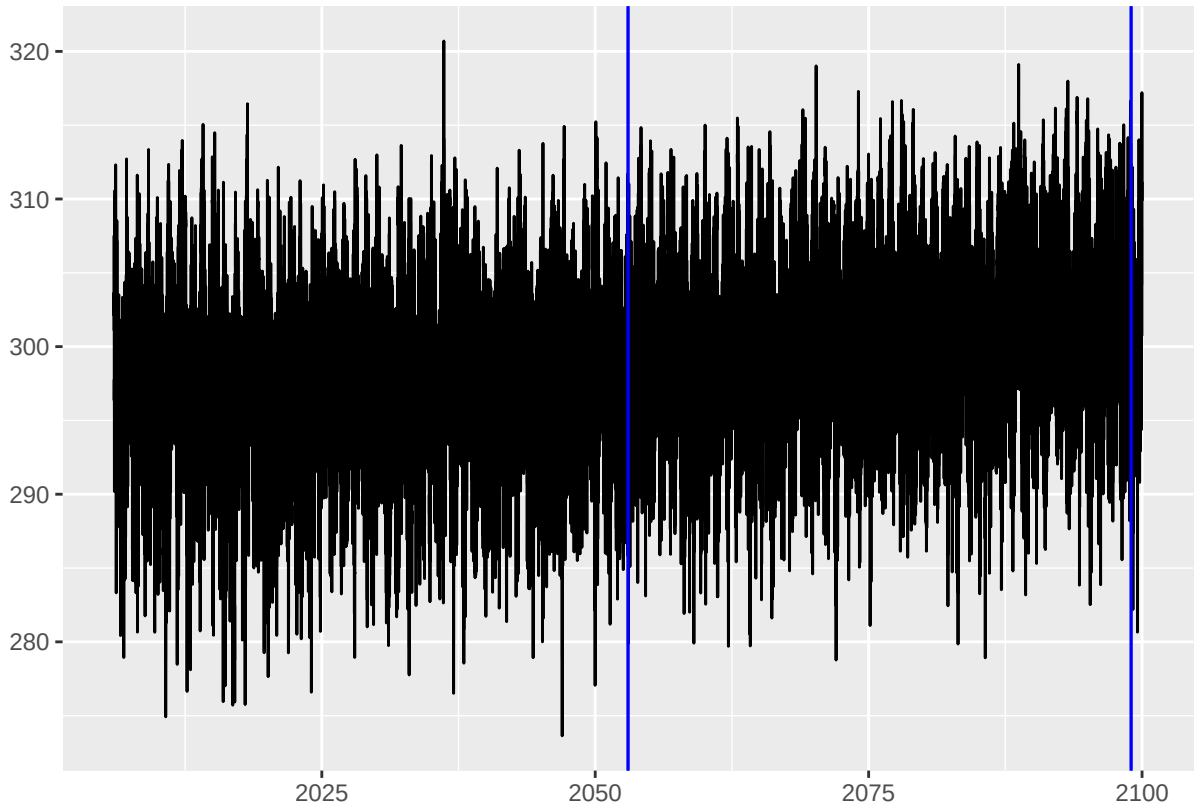
```
max_temp %>%
 stl(s.window = "periodic") %>%
  autoplot() + theme_tufte()
```



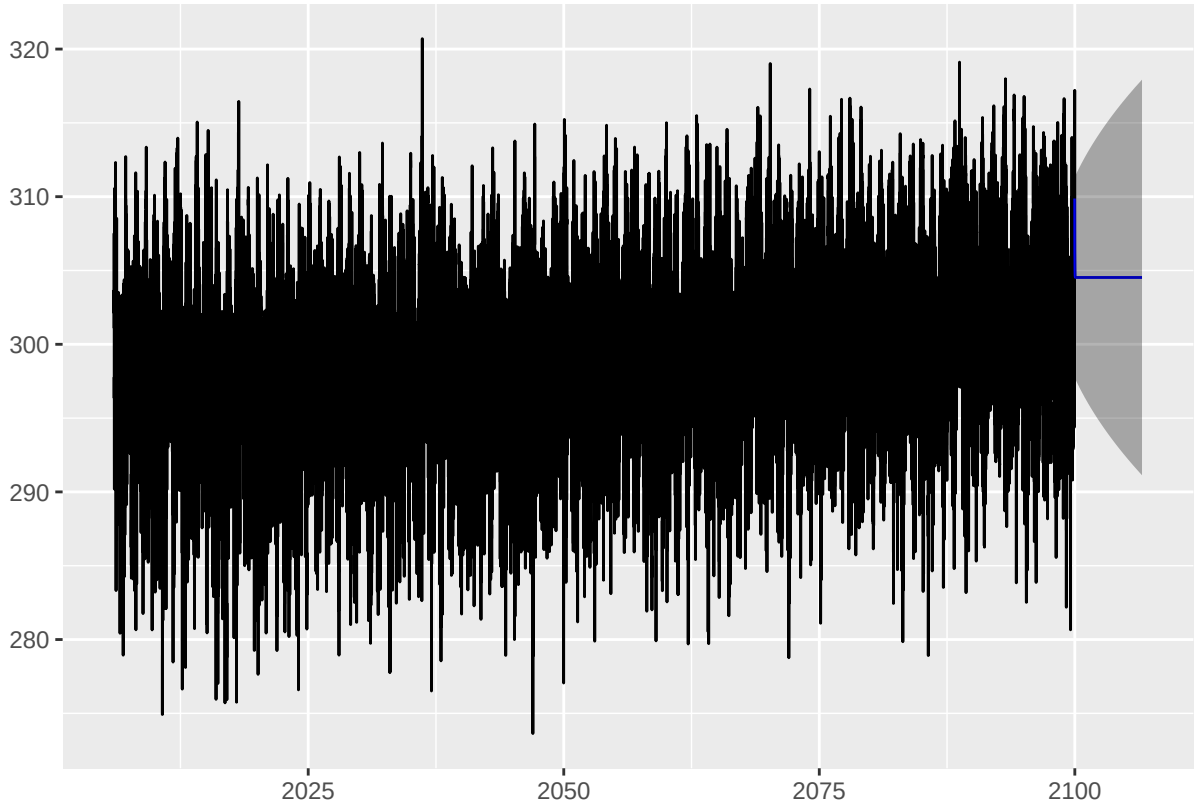

```
arima_max_temp <- auto.arima(seasonally_adjusted_max_temp)
arima_max_temp
```

```
## Series: seasonally_adjusted_max_temp
## ARIMA(4,1,1)
##
## Coefficients:
##          ar1      ar2     ar3     ar4      ma1
##       0.8328  -0.2068  0.0599  0.0370  -0.9855
## s.e.  0.0057   0.0070  0.0070  0.0057   0.0019
##
## sigma^2 = 12.3:  log likelihood = -91725.57
## AIC=183463.1   AICc=183463.1   BIC=183513.8
```

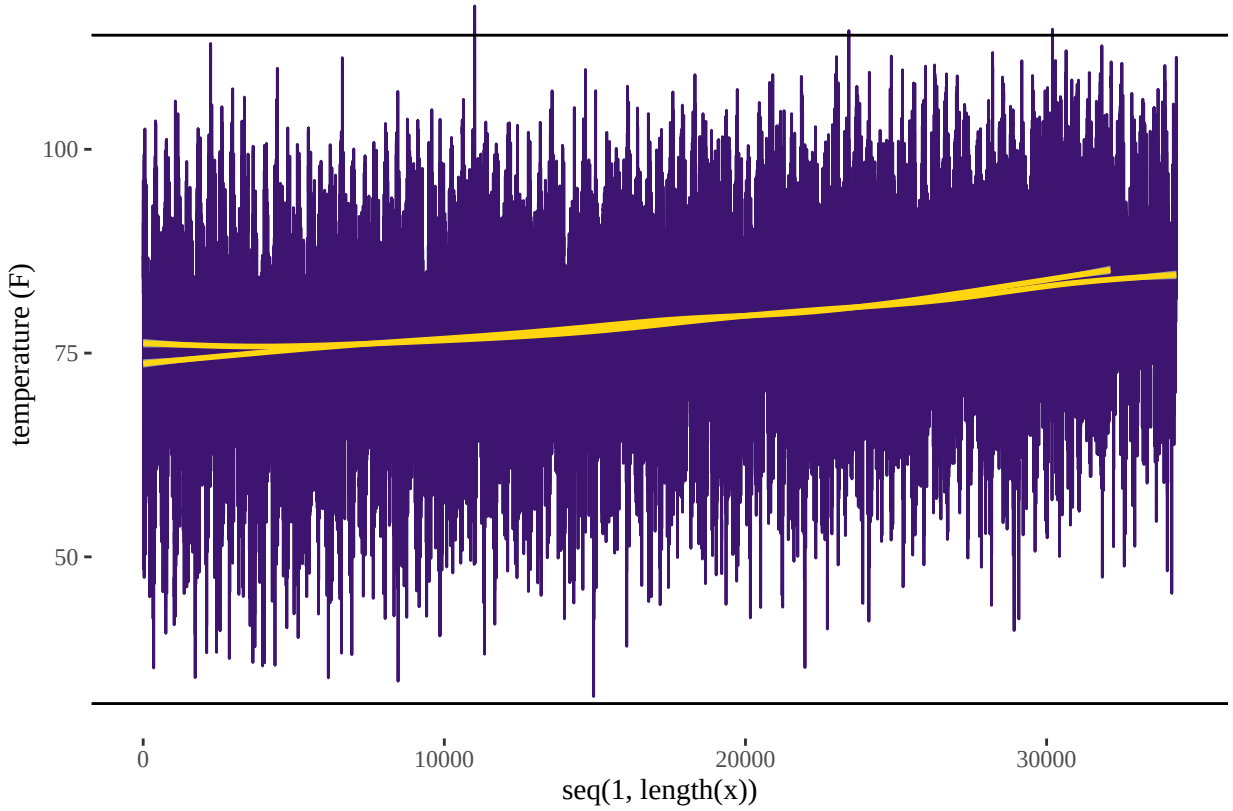
```
autoplot(cpt.meanvar(seasonally_adjusted_max_temp), cpt.colour = 'blue', cpt.linetype = 'solid')
```




```
arima_max_temp %>% forecast(h=2400) %>% autoplot()
```



```
min_temp_df <- kelvin.to.fahrenheit(as.data.frame(seasonally_adjusted_min_temp))
max_temp_df <- kelvin.to.fahrenheit(as.data.frame(seasonally_adjusted_max_temp))
ggplot(data=min_temp_df, aes(y=x, x=seq(1, length(x)))) +
  geom_line(data=min_temp_df, color=our_purple) +
  geom_line(data=max_temp_df, color=our_purple) +
  geom_smooth(data=min_temp_df,color=our_yellow, alpha=1) +
  geom_smooth(data=max_temp_df,color=our_yellow, alpha=1) +
  theme_tufte() +

  geom_hline(yintercept=114)+
  geom_hline(yintercept=32) +
  ylab("temperature (F)")
```



Relative Humidity

```
cft_time_series <- function(data, variable){
require(dplyr)
require(ggplot2)
require(ggfortify)
require(changepoint)
require(weathermetrics)

inputs <- cft::available_data()
available_times <- inputs[[2]]
colnames(available_times)[1] <- "time"
```

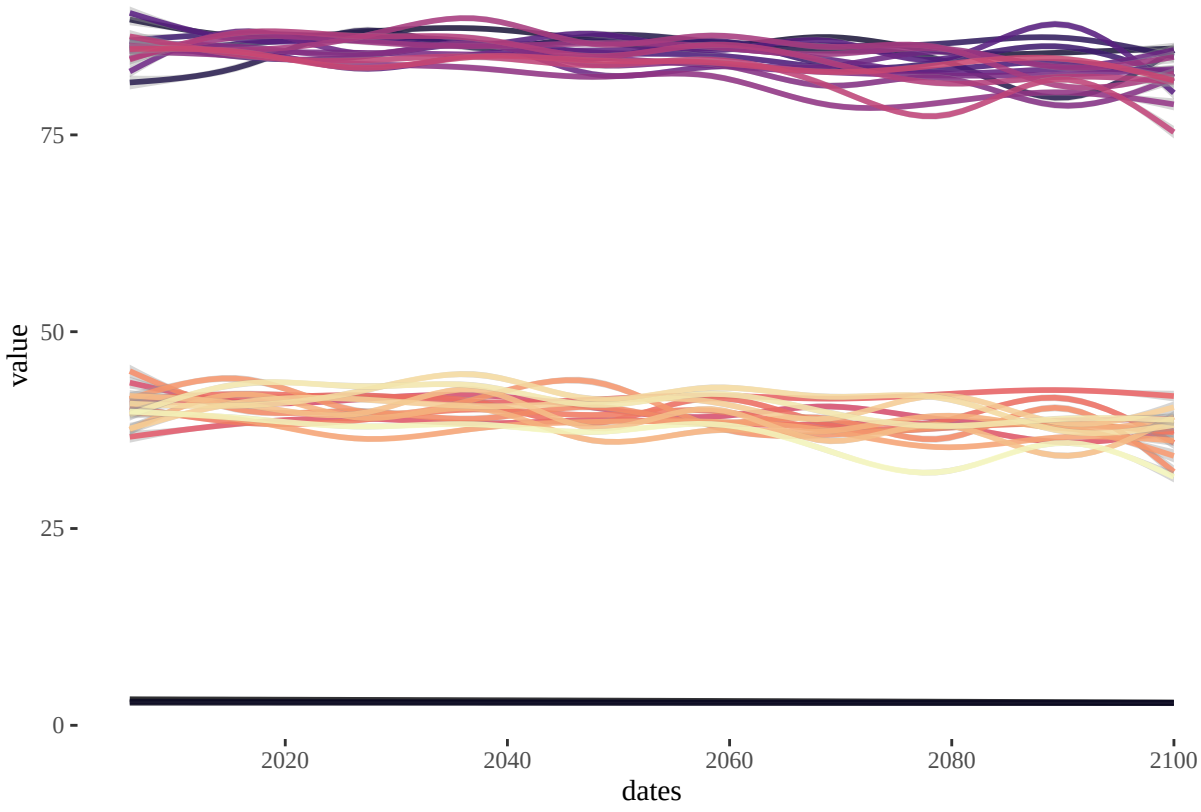
```r
# left join the time data into the spatial data
data_posix <- data %>%
  left_join(available_times, by="time")

# convert time format into POSIX, which is a format that deals with all the confusion of time and data
data_posix$dates <- as.POSIXct(data_posix$dates)

#reorder so that dates are the first column
data_posix <- data_posix[,c(130,2, 1,3:129)]

print("Combined data with verbose dates.")

df <- data_posix %>%
  st_drop_geometry() %>%
  select(dates,which(startsWith(colnames(data), variable)) ) %>%
  gather(key = "variable", value = "value", -dates)

print("regroup data")

plot1 <- ggplot(data= df, aes(x = dates, y = value, color = variable)) +
  scale_colour_viridis_d(option="A", alpha = 0.8) +
  geom_smooth() +
  theme_tufte() +
  theme(legend.position = "none")

df_min <- data_posix %>%
  st_drop_geometry() %>%
  select(dates,which(startsWith(colnames(data_posix), paste0(variable,"min")))) %>%
  gather(key = "variable", value = "value", -dates)

df_max <- data_posix %>%
  st_drop_geometry() %>%
  select(dates,which(startsWith(colnames(data_posix), paste0(variable,"max")))) %>%
  gather(key = "variable", value = "value", -dates)

plot2 <- ggplot(data= data_posix, aes(x = dates, y = value) )+
  geom_smooth(data= df_min, color=our_purple) +
  geom_smooth(data= df_max, color=our_purple) +
  theme_tufte()

print("Filtered to seperate out min and max values")

min_ts <- ts(df_min[,3], frequency = 365, start = c(2006, 1), end = c(2099,365))
max_ts <- ts(df_max[,3], frequency = 365, start = c(2006, 1), end = c(2099,365))

print("Converted to time series object")

seasonally_adjusted_min_ts <- min_ts %>% stl(s.window='periodic') %>% seasadj()
seasonally_adjusted_max_ts <- max_ts %>% stl(s.window='periodic') %>% seasadj()

plot3 <- min_ts %>%
 stl(s.window = "periodic") %>%
  autoplot() + theme_tufte()
```

```
plot4 <- max_ts %>%
 stl(s.window = "periodic") %>%
  autoplot() + theme_tufte()

print("Fit moving window model to max and min")

arima_min_ts <- auto.arima(seasonally_adjusted_min_ts)
arima_min_ts

arima_max_ts <- auto.arima(seasonally_adjusted_max_ts)
arima_max_ts

plot5 <- autoplot(cpt.meanvar(seasonally_adjusted_min_ts), cpt.colour = 'purple', cpt.linetype = 'solid
plot6 <- autoplot(cpt.meanvar(seasonally_adjusted_max_ts), cpt.colour = 'purple', cpt.linetype = 'solid

print("Fit arima model to min and max.")

plot7 <- arima_min_ts %>% forecast(h=2400) %>% autoplot()
plot8 <- arima_max_ts %>% forecast(h=2400) %>% autoplot()

print("Statistical Forecast for 20 years.")

min_ts_df <- as.data.frame(seasonally_adjusted_min_ts)
max_ts_df <- as.data.frame(seasonally_adjusted_max_ts)
plot9 <- ggplot(data=min_ts_df, aes(y=x, x=seq(1, length(x)))) +
  geom_line(data=min_ts_df, color=our_purple) +
  geom_line(data=max_ts_df, color=our_purple) +
  geom_smooth(data=min_ts_df,color=our_yellow, alpha=1) +
  geom_smooth(data=max_ts_df,color=our_yellow, alpha=1) +
  theme_tufte() +
  geom_hline(yintercept=100)+
  geom_hline(yintercept=0) +
  ylab("humidity")

print("done!")

  return(list(df=list(df, df_min, df_max, min_ts, max_ts, seasonally_adjusted_min_ts, seasonally_adjust
}

RH_summary <- cft_time_series(MuscogeeNation, "rhs")
```

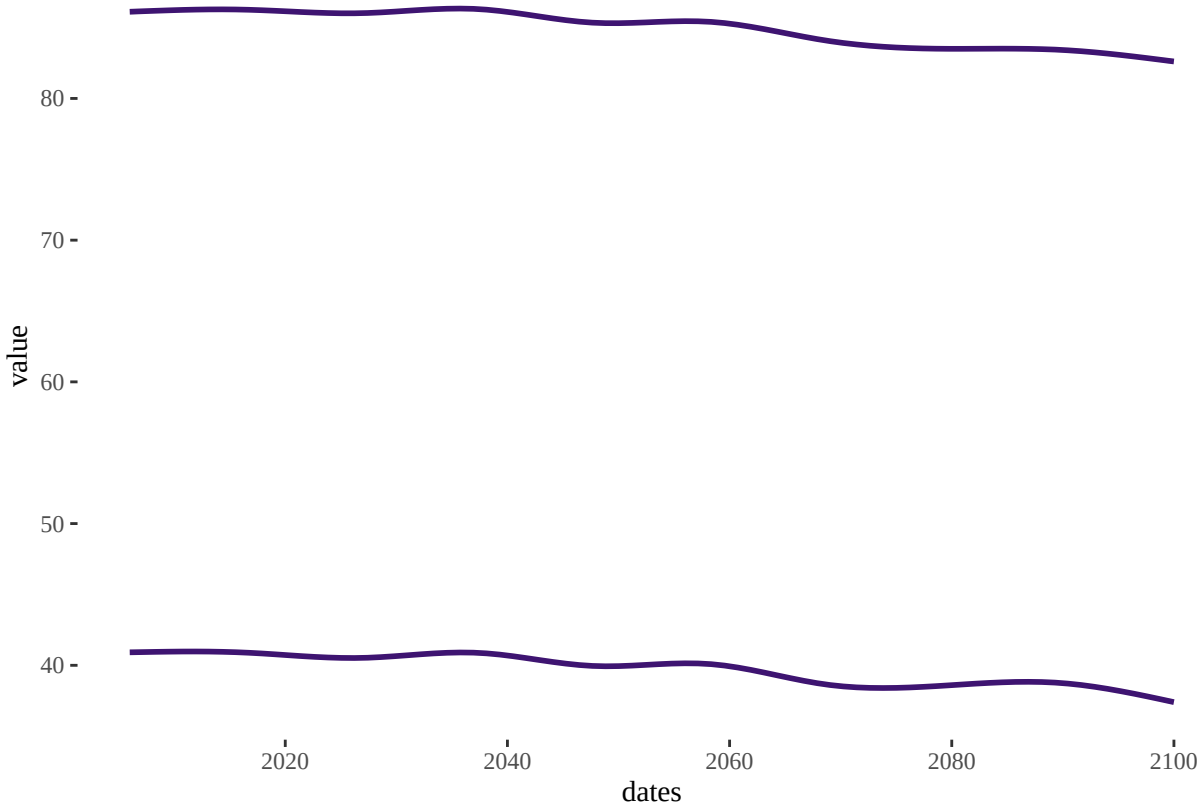
```
## [1] "Combined data with verbose dates."
## [1] "regroup data"
## [1] "Filtered to seperate out min and max values"
## [1] "Converted to time series object"
## [1] "Fit moving window model to max and min"
## [1] "Fit arima model to min and max."
## [1] "Statistical Forecast for 20 years."
## [1] "done!"
```

```
RH_summary$plots
```

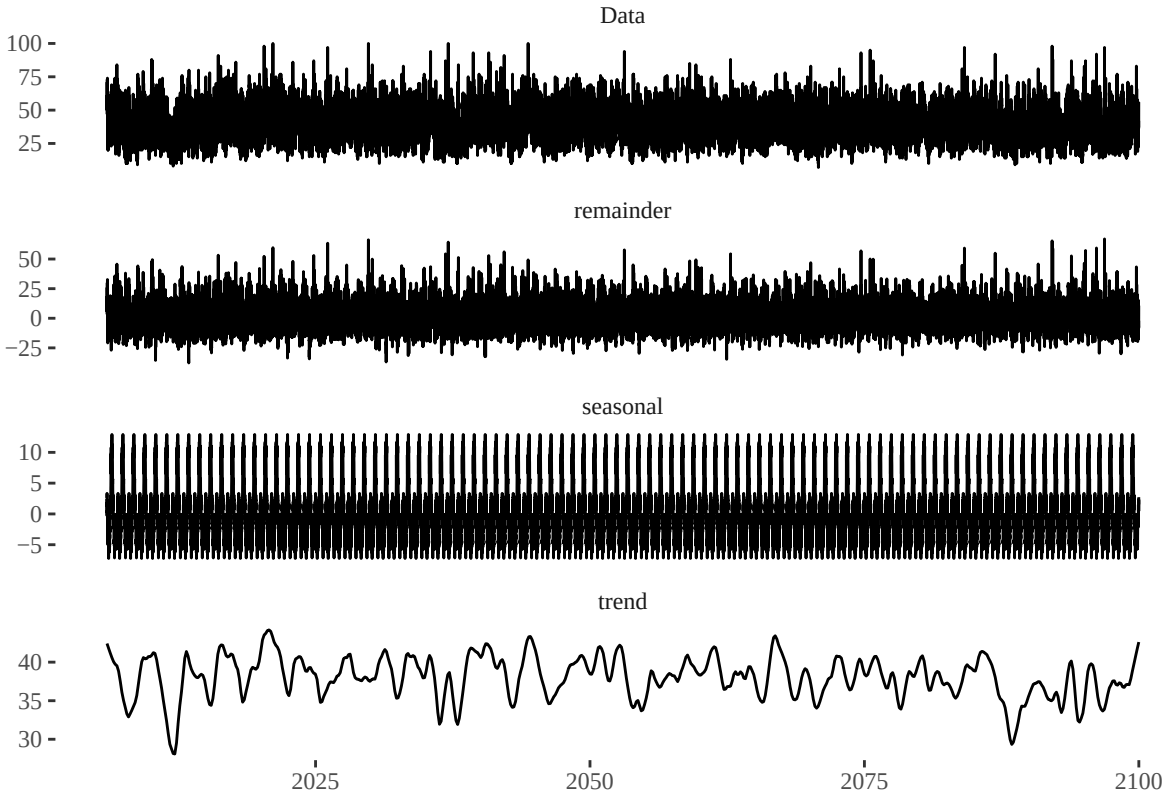
```
## [[1]]
```


75
50
25
0
value
2020
2040
2060
2080
2100
dates

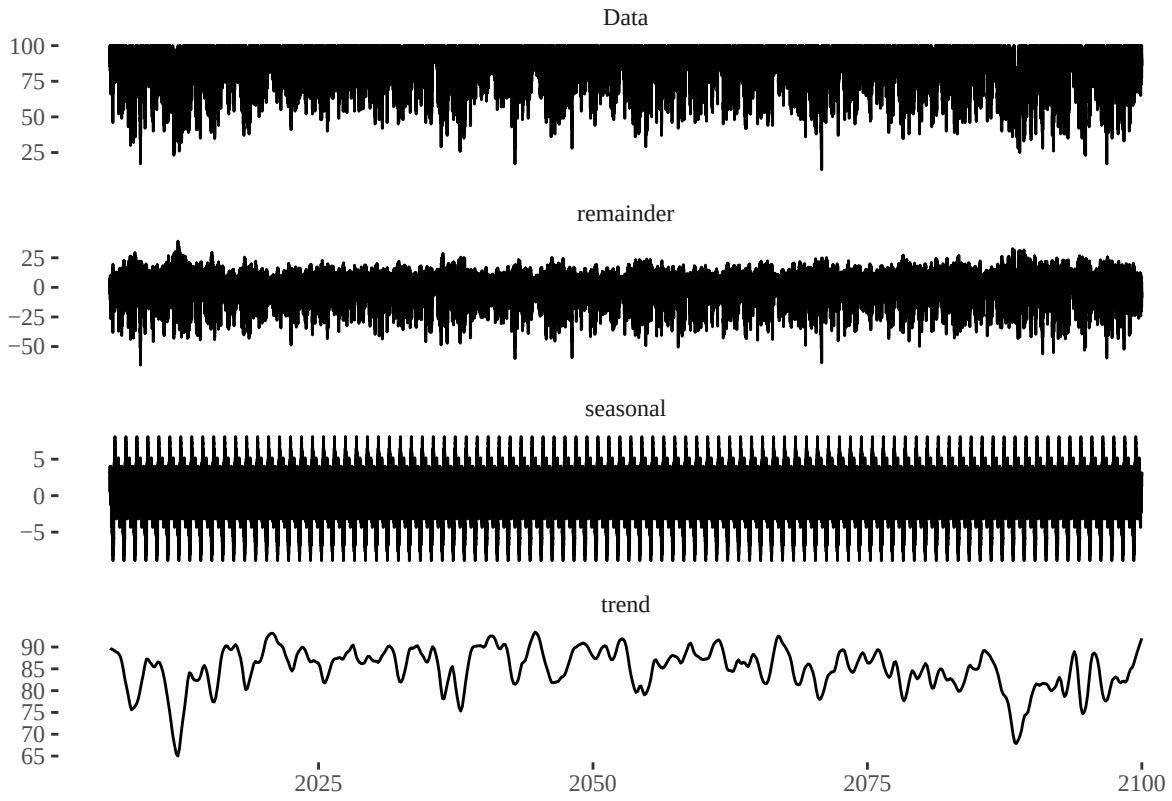
```
##
## [[2]]
```


80
70
60
50
40
value
2020
2040
2060
2080
2100
dates

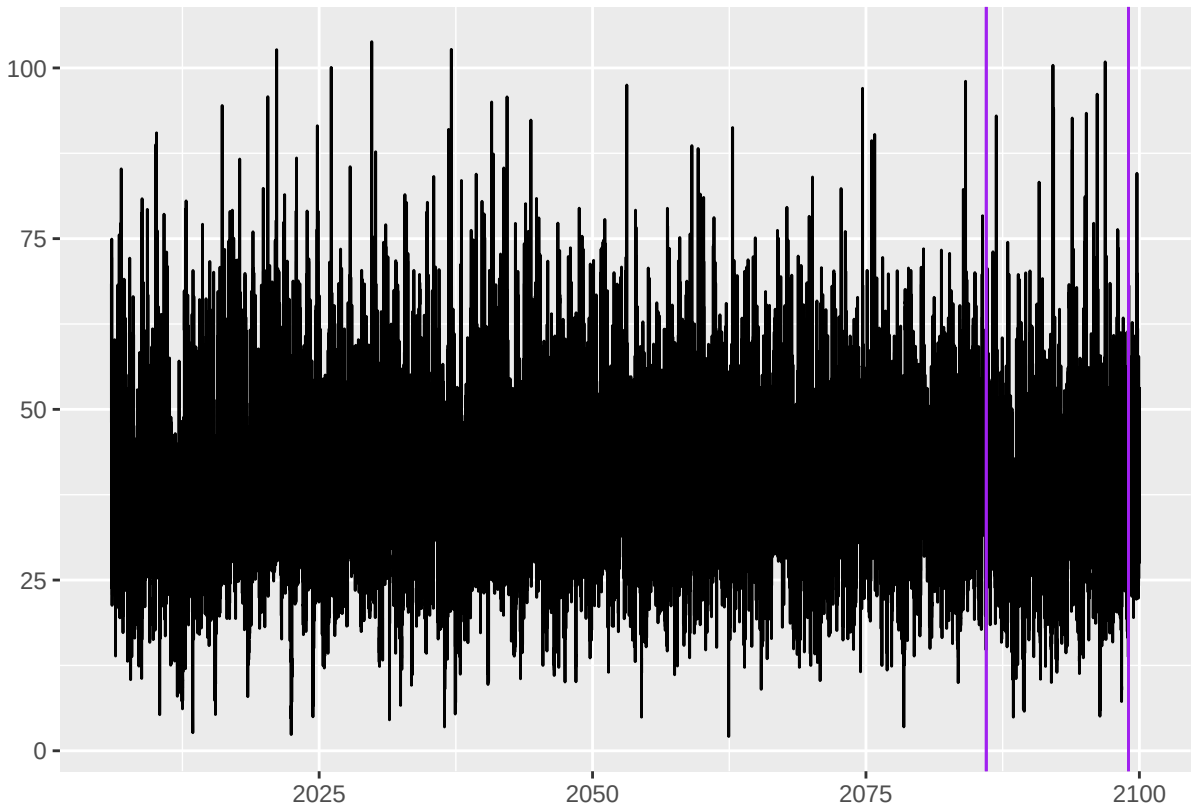
```
##
## [[3]]
```


Data
100
75
50
25
remainder
50
25
0
−25
seasonal
10
5
0
−5
trend
40
35
30
2025
2050
2075
2100

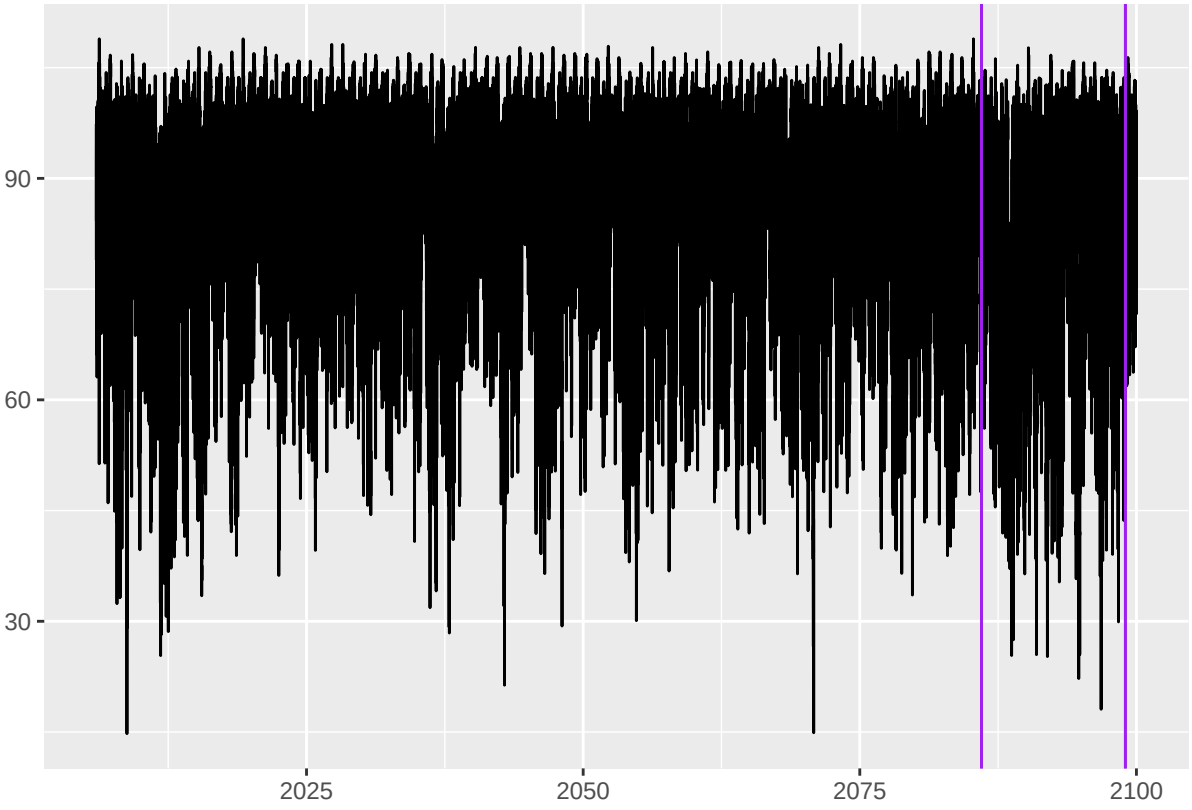
```
##
## [[4]]
```


Data
100
75
50
25
remainder
25
0
−25
−50
seasonal
5
0
−5
trend
90
85
80
75
70
65
2025
2050
2075
2100

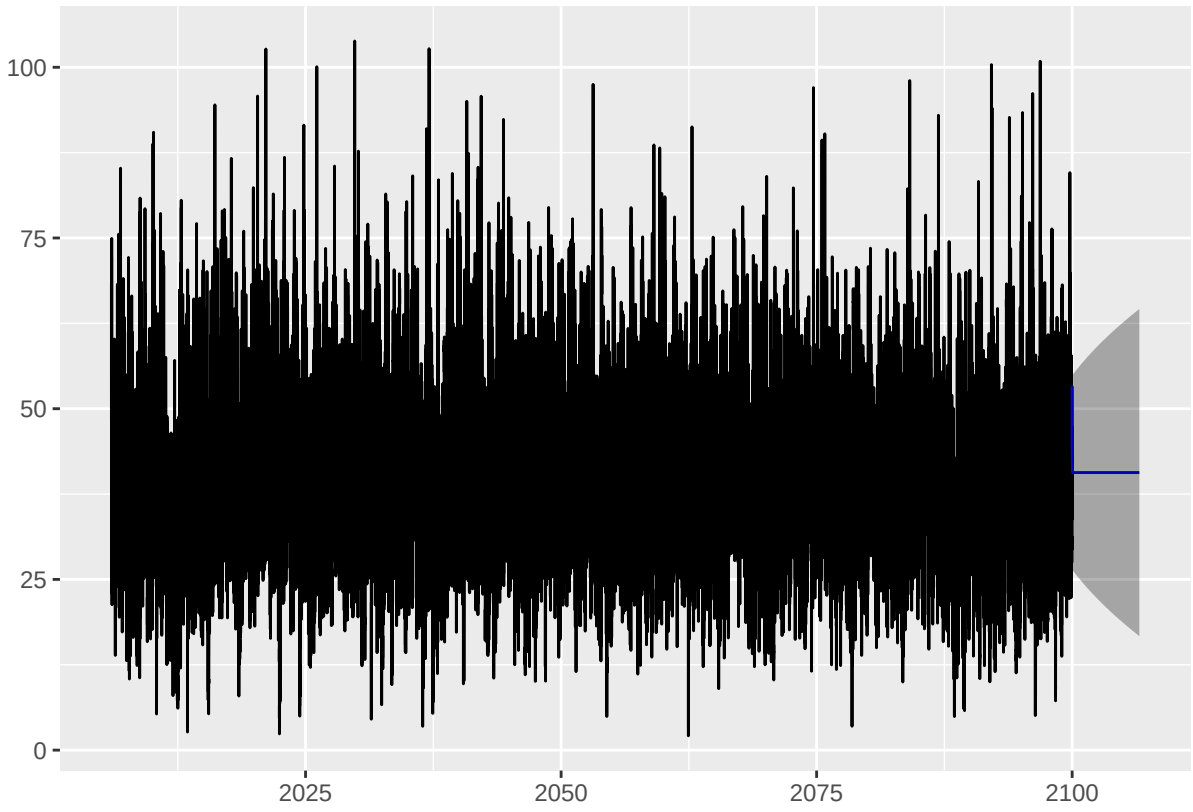
```
##
## [[5]]
```


100
75
50
25
0
2025
2050
2075
2100

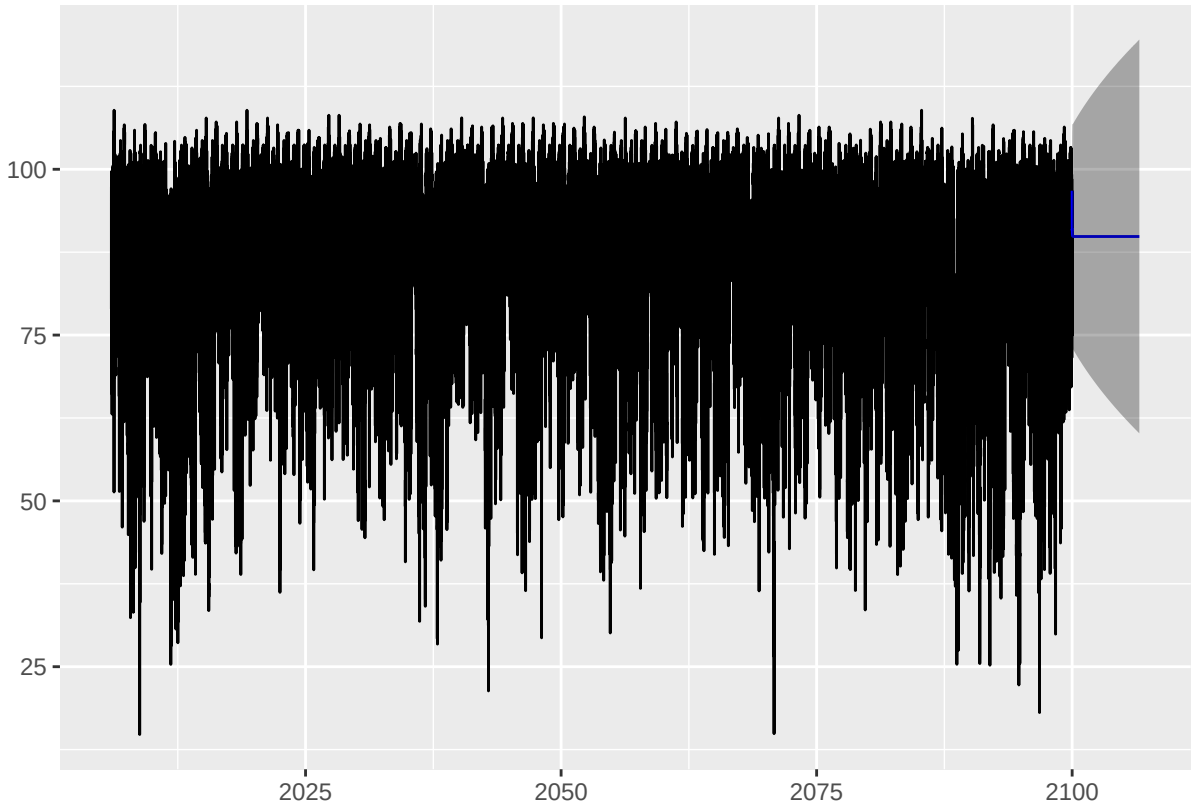
```
##
## [[6]]
```



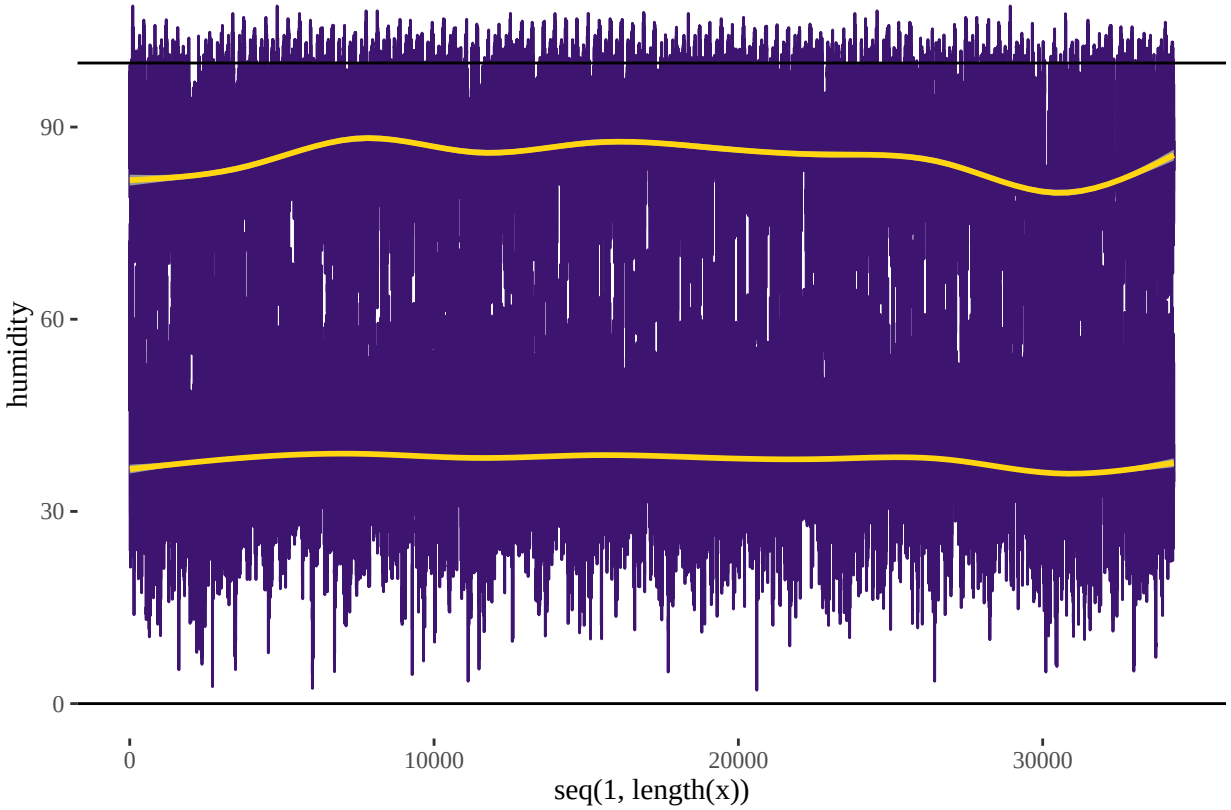

```
##
## [[7]]
```


100
75
50
25
0
2025
2050
2075
2100

```
##
## [[8]]
```


100
75
50
25
2025
2050
2075
2100

```
##
## [[9]]
```



Precipitation

```
cft_time_series.precip <- function(data, variable){
require(dplyr)
require(ggplot2)
require(ggfortify)
require(changepoint)
require(weathermetrics)

  inputs <- cft::available_data()
available_times <- inputs[[2]]
colnames(available_times)[1] <- "time"

# left join the time data into the spatial data
data_posix <- data %>%
  left_join(available_times, by="time")

# convert time format into POSIX, which is a format that deals with all the confusion of time and data
data_posix$dates <- as.POSIXct(data_posix$dates)

#reorder so that dates are the first column
data_posix <- data_posix[,c(130,2, 1,3:129)]
```

```
print("Combined data with verbose dates.")

df <- data_posix %>%
  st_drop_geometry() %>%
  select(dates,which(startsWith(colnames(data), "pr")) ) %>%
  gather(key = "variable", value = "value", -dates)

print("regroup data")

precip_ts <- ts(df[,3], frequency = 365, start = c(2006, 1), end = c(2099,365))

plot1 <- ts_decompose(precip_ts)


plot2 <- ts_plot(precip_ts, slider=TRUE)
plot3 <- ts_heatmap(precip_ts)
plot4 <- ts_seasonal(precip_ts, type = "all")
plot5 <- ts_seasonal(precip_ts - decompose(precip_ts)$trend,
             type = "all",
             title = "Seasonal Plot - precipitation (Detrend)")
plot6 <- ts_surface(precip_ts)


seasonally_adjusted_ts <- precip_ts %>% stl(s.window='periodic') %>% seasadj()

arima_ts <- auto.arima(seasonally_adjusted_ts)


plot7 <- autoplot(cpt.meanvar(seasonally_adjusted_ts), cpt.colour = 'purple', cpt.linetype = 'solid')


print("Fit arima model to min and max.")

plot8 <- arima_ts %>% forecast(h=2400) %>% autoplot()


saveWidget(plot1, "plot1.html")
saveWidget(plot2, "plot2.html")
saveWidget(plot3, "plot3.html")
saveWidget(plot4, "plot4.html")
saveWidget(plot5, "plot5.html")
saveWidget(plot6, "plot6.html")

  print("done!")

  return(list(df = list(df, precip_ts), plots = list(plot1, plot2, plot3, plot4, plot5, plot6, plot7, pl
}

precip_summary <-  cft_time_series.precip(MuscogeeNation, "pr")
```

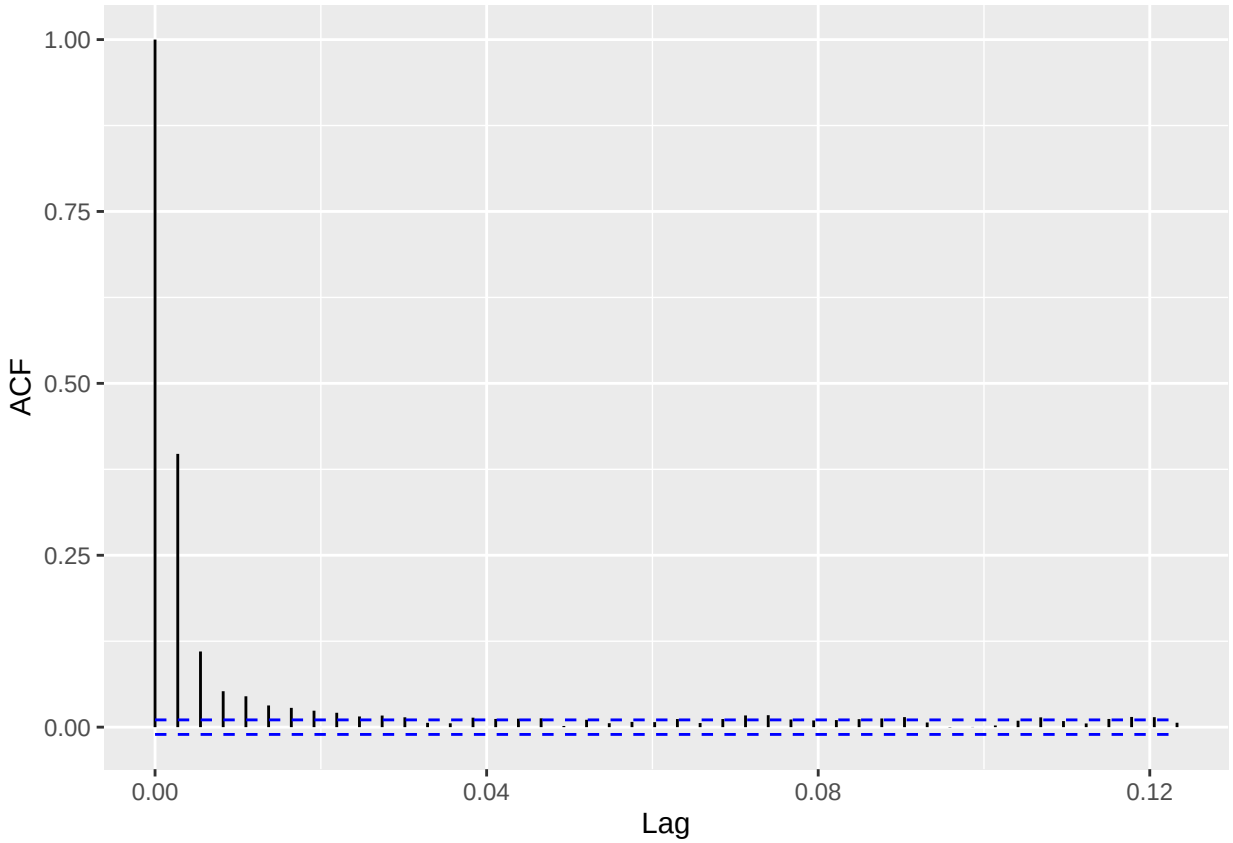
Zero-inflated data

```
## [1] "Combined data with verbose dates."
```

```
## [1] "regroup data"
## [1] "Fit arima model to min and max."
## [1] "done!"
```

```
include_url("plot1.html", height = "400px")
include_url("plot2.html", height = "400px")
include_url("plot3.html", height = "400px")
include_url("plot4.html", height = "400px")
include_url("plot5.html", height = "400px")
include_url("plot6.html", height = "400px")
```

```
autoplot(acf(precip_summary$df[[2]], plot = FALSE))
```



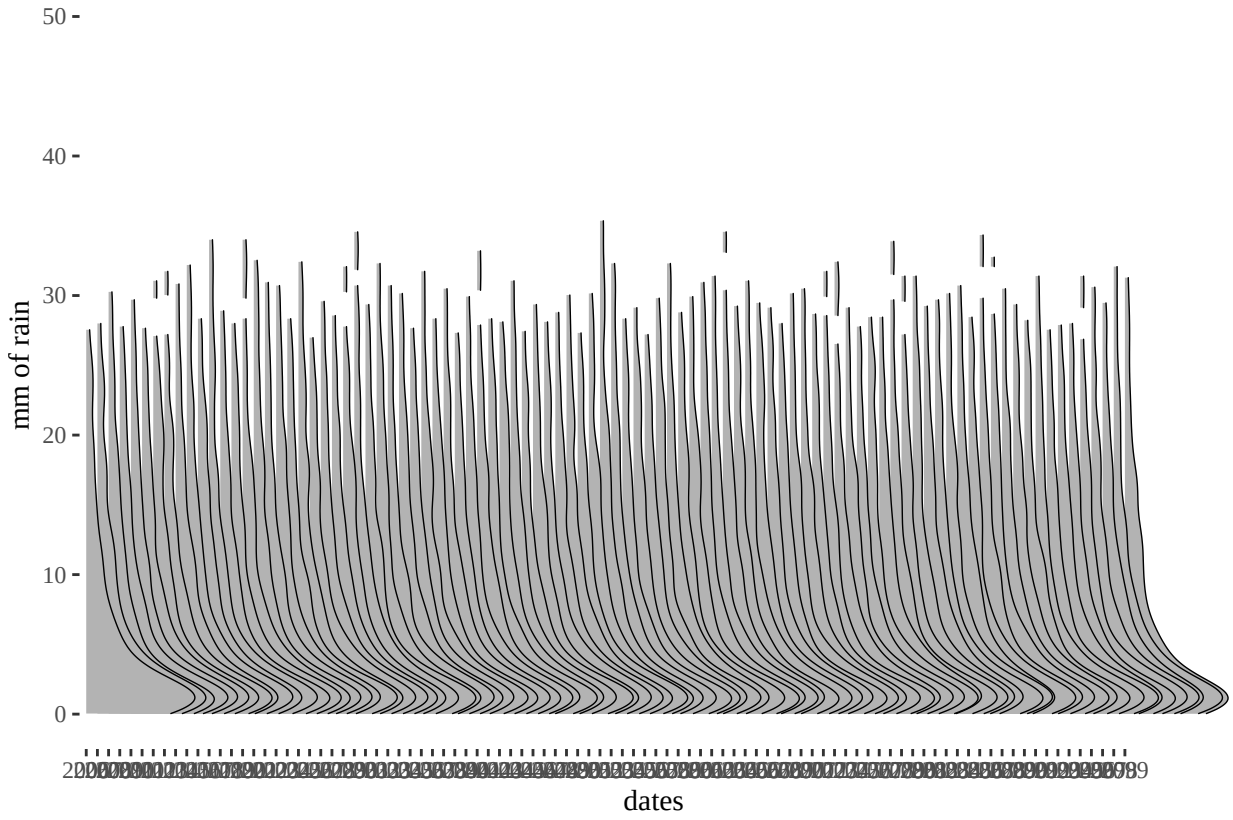
2.78 inches of rain = 70 mm of rain

```
all_rain <- precip_summary$df[[1]] %>%
   filter(value > 0) %>%
filter(str_detect(variable, "pr") )

epic_rain <- precip_summary$df[[1]] %>%
  filter(value >= 100) %>%
filter(str_detect(variable, "pr") )

extreme_rain <- precip_summary$df[[1]] %>%
  filter(value >= 70) %>%
filter(str_detect(variable, "pr") )
```

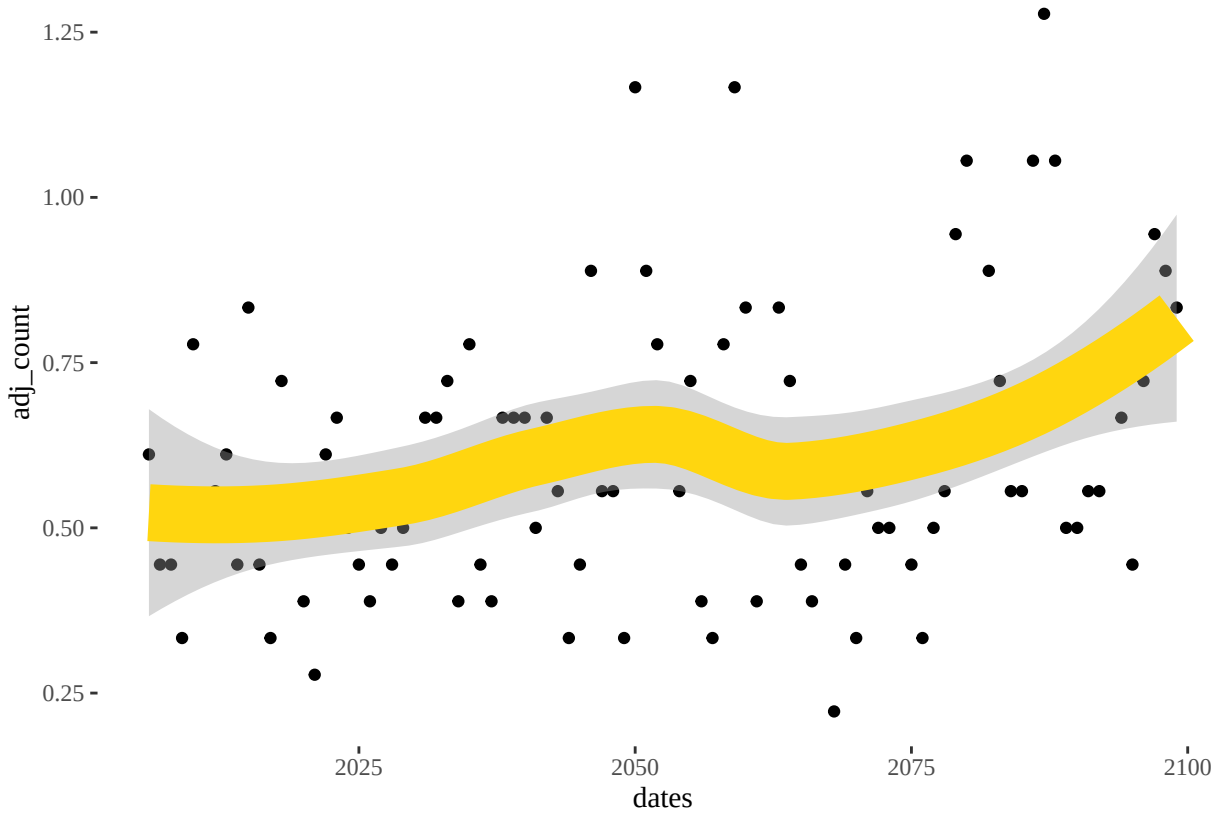
```
no_rain <- precip_summary$df[[1]] %>%
  filter(value == 0) %>%
filter(str_detect(variable, "pr") )
```

```
extreme_rain$dates <- format(extreme_rain$dates, format="%Y")
 extreme_rain$value <- metric_to_inches(extreme_rain$value, unit.from = "mm")
  no_rain$dates <- format(no_rain$dates, format="%Y")
  all_rain$dates <- format(all_rain$dates, format="%Y")
  epic_rain$dates <- format(epic_rain$dates, format="%Y")
  epic_rain$value <- metric_to_inches(epic_rain$value, unit.from = "mm")
```

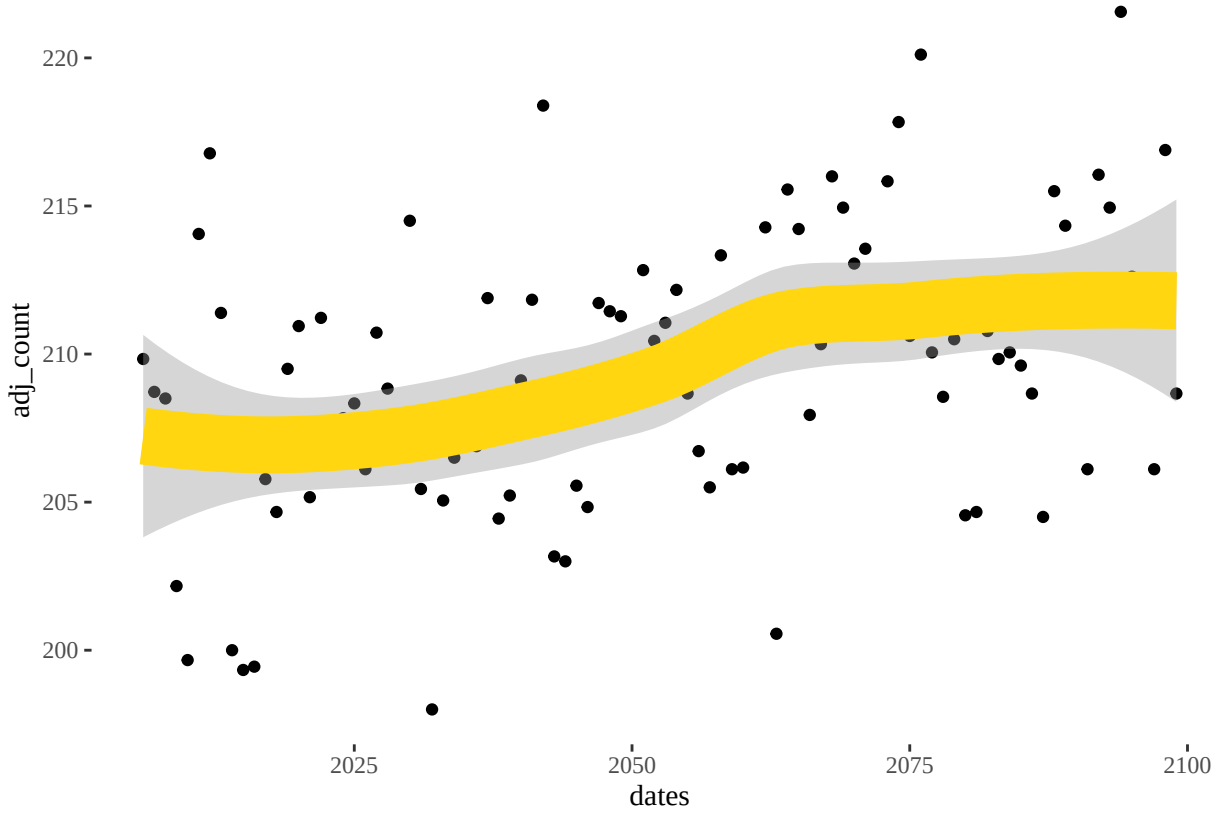
```
extreme_rain_counts <- extreme_rain %>%
 count(dates) %>%
  mutate(adj_count = n/18)

no_rain_counts <- no_rain %>%
  count(dates) %>%
  mutate(adj_count = n/18)

epic_rain_counts <- epic_rain %>%
 count(dates) %>%
  mutate(adj_count = n/18)

ggplot(data=all_rain, aes(x = value, y = dates)) +
   geom_density_ridges(scale = 10, size = 0.25, rel_min_height = 0.03)+
    coord_flip() +
  theme_ridges(grid = FALSE) +
    theme_tufte() + xlim(0,50) + xlab("mm of rain")
```



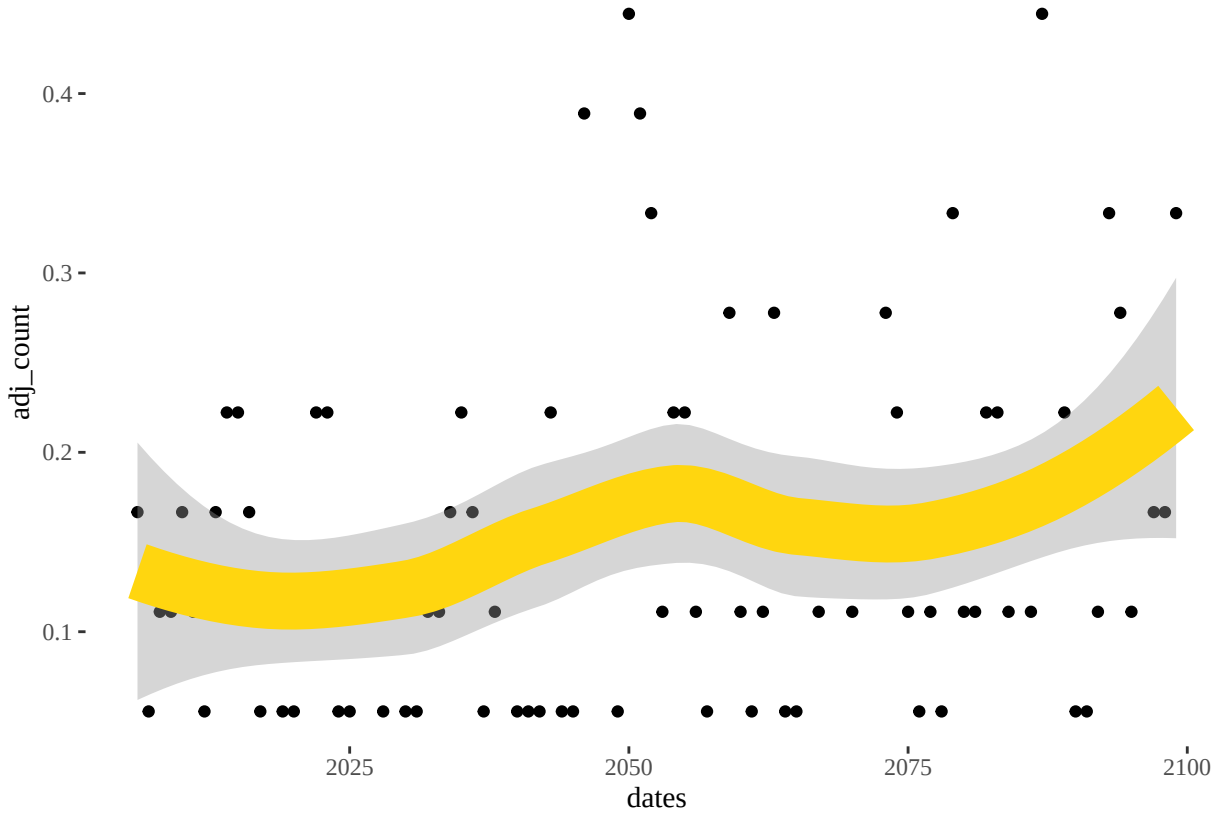

```
no_rain_counts$dates <- as.numeric(no_rain_counts$dates)
extreme_rain_counts$dates <- as.numeric(extreme_rain_counts$dates)
epic_rain_counts$dates <- as.numeric(epic_rain_counts$dates)

   ggplot(data=extreme_rain_counts, aes(y = adj_count, x = dates)) +
   geom_point()+
   theme_tufte()+
   geom_smooth(col=our_yellow, size=10)
```




```
ggplot(data=no_rain_counts, aes(y = adj_count, x = dates)) +
geom_point()+
theme_tufte()+
geom_smooth(col=our_yellow, size=10)
```




```
ggplot(data=epic_rain_counts, aes(y = adj_count, x = dates)) +
geom_point()+
theme_tufte()+
  geom_smooth(col=our_yellow, size=10)
```




```
    ylim(0,0.4)
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits:    0 --  0.4
```