Temperature data option calculation

Pietro Rota

Table of contents

::: {.content-visible html} **Last updated:** 2025-05-22 :::

1 Weather Derivatives Temperature Options

Full dependencies list, click to expand list

```
[1] "R version 4.4.3 (2025-02-28 ucrt)"
```

```
[1] "x86_64-w64-mingw32/x64"
```

Functions loaded from main

Functions.1	Functions.2	Functions.3
check_acc	desc_df	find_outliers
normalize	quickplot	remove_outliers
RSS	show_df	smart_round

Packages required to run this file

```
required_packages(file)
```

Required.Packages.1	Required.Packages.2	Required.Packages.3
caret	changepoint	dplyr
DT	e1071	fBasics
forecast	gganimate	ggplot2
gt	gtExtras	knitr
leaflet	lubridate	MASS
nlme	nlstools	NMOF
PerformanceAnalytics	plotly	quantmod
reshape2	rugarch	splines
stats4	tibble	tidyr
tidyverse	timeDate	timeSeries
TTR	xts	zoo

Functions defined in this document

Required.Functions.1	Required.Functions.2	Required.Functions.3
apply_convolution	sin_component	create_spline_plot
create_poly_plot	fourier_series	create_fourier_plot
create_changepoint_plot	T_model	dT_model
spline_fit	euler_step	monte_carlo_temp
model_formula		

```
time of creation
```

```
Warning in as.POSIXlt.POSIXct(x, tz): unknown timezone '
'
```

```
[1] "2025-05-06 15:32:44 GMT"
```

```
LAST MODIFICATION
```

```
Warning in as.POSIXlt.POSIXct(x, tz): unknown timezone '
'
```

```
[1] "2025-05-22 15:11:26 GMT"
```

```
Last Access
```

```
Warning in as.POSIXlt.POSIXct(x, tz): unknown timezone '
'
```

```
[1] "2025-05-22 15:11:26 GMT"
```

References:

1. Weather Futures and Options (CME Group Inc, 2021) [LINK]
2. Managing Climate Risk with CME Group Weather Futures and Options (Dominic Sutton-Vermeule, 20-Jan 2021) [LINK]
3. MANAGING CLIMATE RISK IN THE U.S. FINANCIAL SYSTEM, ISBN: 978-0-578-74841-2 (2020) [LINK]

2 Climate data download

Much of the weather market is dominated by temperature derivatives, aimed to protect the holder from unexpected temperature prints. The underlying used to trade these contracts is either Heating Degree Days (HDD) or Cooling Degree Days (CDD). For a specific day n, these can be defined as:

The most popular temperature instruments traded are options, whose payoff function depends on a cumulative sum over a longer period, usually an entire season:

$HDDn = \max(T_{ref} - T_n, 0)$

$CDDn = \max(T_n - T_{ref}, 0)$

Table 4: Calls protect you from extreme circumstances, meanwhile puts protect you from mild climates

Option Type	Protection Against	Exercise When	Payout	Example
HDD call	Overly cold winters	HDD > K	$\alpha\cdot$ (HDD - K)	Farmers
HDD **put**	Overly warm winters	HDD < K	$\alpha\cdot$ (K - HDD)	Ski resorts
CDD call	Overly hot summers	CDD > K	$\alpha\cdot$ (CDD - K)	Utilities
CDD **put**	Overly cold summers	CDD < K	$\alpha\cdot$ (K - CDD)	Beaches

NASA Prediction Of Worldwide Energy Resources (POWER) | Data Access Viewer (DAV) [LINK]

This dataset is from the Agroclimatology community, which is crucial for ensuring reproducibility in climate research.

This study utilizes temperature data obtained from the MERRA-2 (Modern-Era Retrospective analysis for Research and Applications, Version 2) dataset, which provides detailed meteorological measurements at a spatial resolution of 0.5 x 0.625 degrees latitude/longitude.

The specific region analyzed, with an average elevation of 288.19 meters, is located at a latitude of 45.5330, and longitude of 9.1911.

```
library(leaflet, quietly = TRUE, warn.conflicts = FALSE)
map_data <- data.frame(
  name = "Location",
  lat = 28.3688,
  lon = -81.5614
)
# Create a leaflet map
if (knitr::is_html_output()) {
  leaflet(map_data) %>%
    addTiles() %>% # Add default OpenStreetMap tiles
    addMarkers(~lon, ~lat, label = ~name, ) %>%
    addCircleMarkers(~lon, ~lat, radius = 10, color = "red", fillOpacity = 0.8) %>%
    setView(lng = map_data$lon, lat = map_data$lat, zoom = 12)
} else {
  print("interactive map of disneyworld in the html version")
}
```

```
[1] "interactive map of disneyworld in the html version"
```

It is important to note that the dataset uses a value of -999 to denote missing data, which I've removed and replaced with the mean of the overall dataset. Either when a parameter cannot be computed or falls outside the range of available source data

Parameter(s):

- T2M_MAX = Temperature at 2 Meters Max for the day (C)
- T2M_MIN = Temperature at 2 Meters Min for the day (C)
- T2M_AVG = $\frac{T_{MIN}+T_{MAX}}{2}$ Temperature at 2 Meters Average for the day (C)

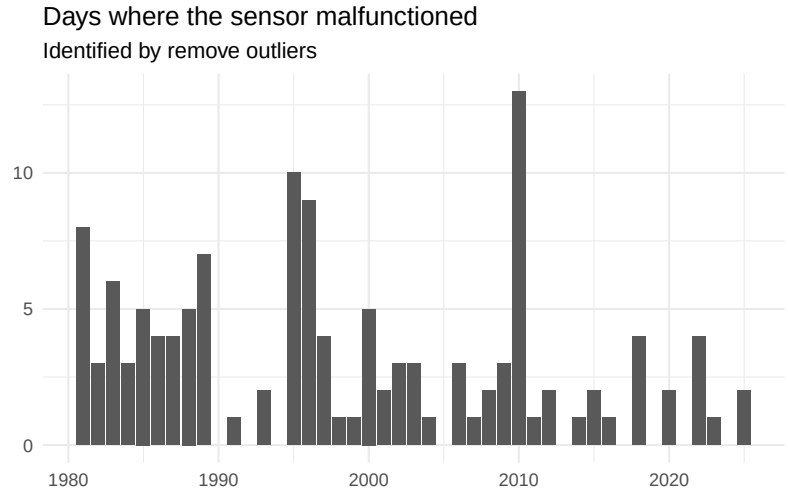
Temperature at 2 meters refers to the air temperature measured 2 meters (approximately 6.5 feet) above the ground. This is the standard height for temperature measurements used in meteorology because it minimizes the effects of ground heating or cooling, providing a more accurate reflection of the ambient air temperature experienced by humans and relevant to various economic activities.

Why? Temperature at 2 meters is commonly used as the basis for weather derivatives. This is because:

- **Standardization**: Temperature at 2 meters is a standardized and widely available data point, making it reliable for use in financial contracts.
- **Relevance**: Many weather-related risks, such as heating degree days (HDD) and cooling degree days (CDD), are calculated based on temperatures at this height. These metrics are commonly used in weather derivatives to hedge against risks related to heating and cooling needs.
- **Accuracy**: Because it represents the temperature most relevant to human activities and energy consumption, it is directly applicable for creating and settling weather derivatives that are based on temperature-related indices.

```
ORIGINAL_DATASET <- read.csv("DISNEY DATA.csv", skip = 10)

DATASET <- ORIGINAL_DATASET %>%
  mutate(T_MAX=remove_outliers(T2M_MAX, fill = "NA") %>% na.approx) %>%
  mutate(T_MIN=remove_outliers(T2M_MIN, fill = "NA") %>% na.approx) %>%
  mutate(DAY = as.Date(ORIGINAL_DATASET$DOY - 1, origin = paste0(ORIGINAL_DATASET$YEAR, "-01-0
  mutate(Month=month(DAY)) %>%
  dplyr::select(DAY, YEAR, Month, DOY, T_MAX, T_MIN)


DATASET$T_AVG <- apply(DATASET[c("T_MAX", "T_MIN")], 1, mean)

ORIGINAL_DATASET[ifelse(find_outliers(ORIGINAL_DATASET$T2M_MAX)==0,FALSE,TRUE),] %>%
  group_by(YEAR) %>%
  summarise("Days" = length(DOY)) %>%
  ggplot( aes(x = YEAR, y = Days))+
```

```
    geom_bar(stat = "identity") +
    labs(x= NULL, y =NULL, title = "Days where the sensor malfunctioned",
        subtitle = "Identified by remove outliers")
```




```
if (knitr::is_html_output()) {
DATASET %>% select(DAY, T_MAX, T_MIN, T_AVG) %>%
  smart_round(digits = 3) %>%
  datatable() %>%
  formatStyle("T_MAX",
  background = styleColorBar(range(DATASET$T_MAX), "indianred3"),
  backgroundSize = "100% 80%",
  backgroundRepeat = "no-repeat"
  )
  } else {
  print("interactive table of the data in the html version")
}
```

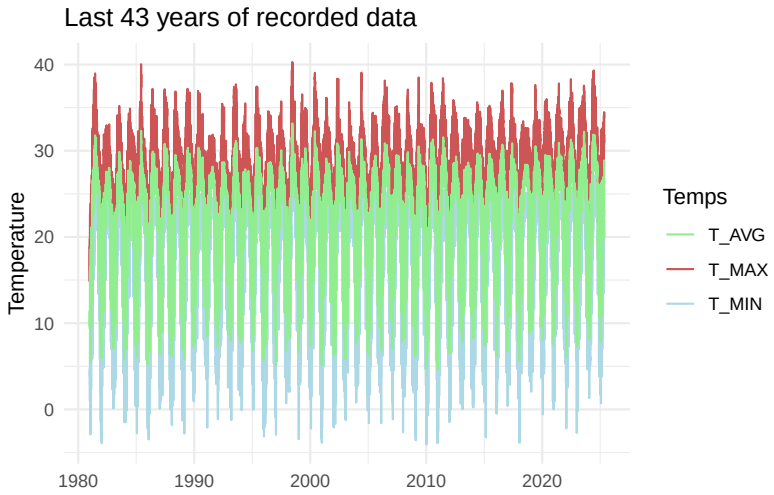
```
[1] "interactive table of the data in the html version"
```

2.1 Initial data visualization

```
cleandataset <- DATASET %>%
  select(T_MAX, T_MIN, T_AVG) %>%
  xts(order.by = DATASET$DAY)

desc_df(cleandataset)
```

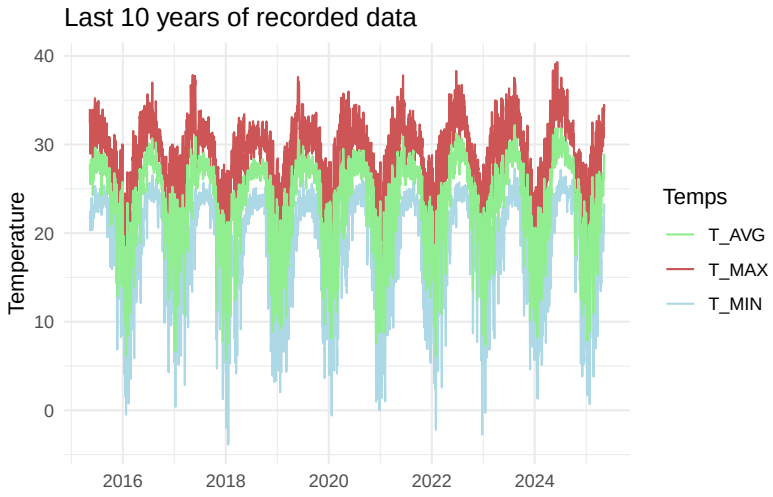
	n	mean	sd	median	trimmed	min	max	range	skew	kurtosis	se	%NA	Q.1%	Q.25%	Q.75%	Q.99%
T_MAX	16128	27.57	4.56	28.42	27.94	12.58	40.32	27.74	-0.19	-0.38	0.04	0	14.32	24.48	30.96	36.64
T_MIN	16128	17.56	4.09	18.88	18.29	0.00	27.32	27.32	-1.39	0.03	0.03	0	0.77	10.82	22.87	25.36
T_AVG	16128	22.56	4.08	23.65	23.14	8.96	33.17	24.21	-0.50	0.07	0.03	0	8.28	17.52	26.74	31.23

```
plot1 <- cleandataset %>%
  as.data.frame() %>%
  mutate(year = index(cleandataset)) %>%
  ggplot(aes(x=year))+
  geom_line(aes(y=T_MAX, color = "T_MAX"))+
  geom_line(aes(y=T_MIN, color = "T_MIN"))+
  geom_line(aes(y=T_AVG, color = "T_AVG"))+
  labs(title = "Last 43 years of recorded data", y="Temperature", x=NULL)+
  scale_color_manual(name = "Temps",
                     values = c(T_MAX = "indianred3",T_MIN = "lightblue",T_AVG = "lightgreen"))

plot1
```



However this is not that useful so let me zoom in on the last third of the database (approximately 2014 onward)

```
NDAYS <- nrow(cleandataset)
lookback <- 365*10

tail(cleandataset, lookback) %>%
  as.data.frame() %>%
  mutate(year = index(tail(cleandataset, lookback))) %>%
  ggplot(aes(x = year))+
  geom_line(aes(y=T_MAX, color = "T_MAX"))+
  geom_line(aes(y=T_MIN, color = "T_MIN"))+
  geom_line(aes(y=T_AVG, color = "T_AVG"))+
  labs(title = "Last 10 years of recorded data", y="Temperature", x=NULL)+
  scale_color_manual(name = "Temps",
                     values = c(T_MAX = "indianred3",T_MIN = "lightblue",T_AVG = "lightgreen"))
```



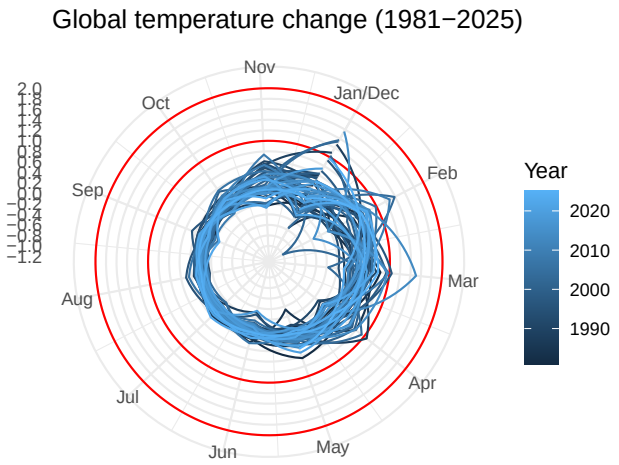

```
library(gganimate)

months365 <- c()  # initialize empty vector

for (i in month.abb) {
  months365 <- c(months365, i, rep(" ", 4))
}
monthplot <- months365[-52][-47][-40][-33][-22][-13][-8]

plot_data <- cleandataset %>%
  as.data.frame() %>%
  mutate(
    date = index(cleandataset),
    year = year(date),
    month = month(date),
    week = week(date)
  ) %>%
  group_by(year, week) %>%
  summarise(T_AVG = mean(T_AVG), .groups = 'drop') %>%
  arrange(year, week)

if (knitr::is_html_output()) {
funny_plot <- ggplot(plot_data, aes(x = factor(week), y = T_AVG, group = year, color = T_AVG))
  geom_line(linewidth = 1.2, show.legend = FALSE) +
  coord_polar() +
```

```
  scale_x_discrete(breaks = 1:53, labels = monthplot) +
  scale_color_gradient(low = "blue", high = "red") +
  labs(subtitle = "Year: {frame_time}", x = NULL, y = NULL) +
  transition_reveal(year)

animate(funny_plot, fps = 30, duration = 15, end_pause =120)
} else {
print("animated plot in html")
}
```

```
[1] "animated plot in html"
```

```
library(tidyverse)

pivot_df <- DATASET %>%
  mutate(t_diff = c(NA, diff(T_AVG))) %>%
  select(Month, YEAR, t_diff) %>%
  pivot_wider(names_from = YEAR, values_from = t_diff, values_fn = median) %>%
  mutate(Month = month.abb) %>%
  melt(id.vars = "Month", variable.name = "Year") %>%
  arrange(Year) %>%
  drop_na() %>%
  mutate(Year = as.integer(Year)+1980)

last_dec <- pivot_df %>%
  filter(Month == "Dec") %>%
  mutate(Year = Year - 1,
  Month = "last_Dec")

next_jan <- pivot_df %>%
  filter(Month == "Jan") %>%
  mutate(Year = Year + 1,
  Month = "Next_jan")

t_data <- bind_rows(pivot_df, next_jan) %>%
  mutate(Month = factor(Month, levels = c(month.abb, "next_Jan")),
         Month_number = as.numeric(Month)) %>% drop_na()

annotation <- t_data %>%
  slice_max(Year-1) %>%
```

```
  slice_max(Month_number)

temp_lines <- tibble(
  x = 12,
  y = c(1.5, 2.0),
  labels = c("1.5\u00B0C", "2.0\u00B0C")
)

month_labels <- tibble(
  x = 1:12,
  labels = month.abb,
  y = 2.7
)

t_data %>%
  ggplot(aes(x=Month_number, y=value, group=Year, color=Year)) +
  geom_hline(yintercept = c(1, 2.0), color="red") +
  geom_line() +
  scale_x_continuous(breaks=1:12,
                     labels=month.abb, expand = c(0,0),
                     sec.axis = dup_axis(name = NULL, labels=NULL)) +
  scale_y_continuous(breaks = seq(-2, 2, 0.2),
                     limits = c(-2, 2.7), expand = c(0, -0.7),
                     sec.axis = dup_axis(name = NULL, labels=NULL)) +
  coord_polar(start = 2*pi/12) +
  labs(x = NULL,
       y = NULL,
       title = "Global temperature change (1981-2025)")
```



2.2 Seasonal analysis

Now i can look at the distribution of my database, first i need to distinguish the data between summer periods and winter periods by taking the day of the year and splitting it so that

Winter ~ October-March

Summer ~ April-September

```
DATASET_seas <- DATASET %>%
  group_by(Month < 4|Month>9) %>%
  rename("Season"="Month < 4 | Month > 9") %>%
  mutate(Season = ifelse(Season, "Winter", "Summer"))

DATASET_seas[-1] %>%
  xts(order.by = DATASET$DAY)%>%
  show_df() %>%
  gt() %>%
  tab_header(title = "Temperature Overview") %>%
  opt_stylize(style = 5, add_row_striping = TRUE) %>%
  cols_align(align = "center") %>%
  sub_missing(columns = everything(), missing_text = " ")
```

```
Warning: There was 1 warning in `mutate()`.
```

Temperature Overview

DATE	YEAR	Month	DOY	T_MAX	T_MIN	T_AVG	Season
4018	1981	1	1	18.96000	3.7600000	11.360000	Winter
4019	1981	1	2	14.78000	4.7100000	9.745000	Winter
4020	1981	1	3	17.66000	1.0000000	9.330000	Winter
4021	1981	1	4	19.79000	2.9600000	11.375000	Winter
4022	1981	1	5	14.88000	4.4500000	9.665000	Winter
20212	2025	5	124	31.01000	21.9100000	26.460000	Summer
20213	2025	5	125	32.31000	20.0000000	26.155000	Summer
20214	2025	5	126	34.18000	22.2700000	28.225000	Summer
20215	2025	5	127	34.48000	23.2600000	28.870000	Summer
20216	2025	5	128	33.77000	22.3900000	28.080000	Summer

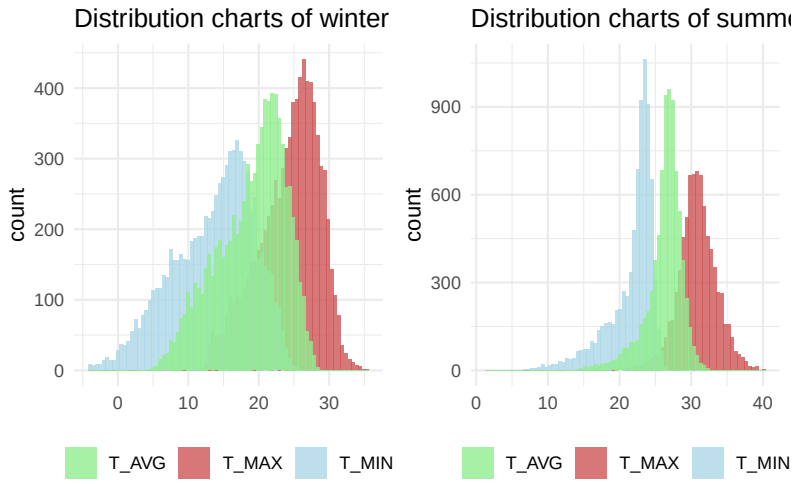
```
i In argument: `across(where(is.numeric), round, digits = digits)`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))
```

```
winter_dataset <- DATASET_seas[DATASET_seas$Season=="Winter",]
summer_dataset <- DATASET_seas[DATASET_seas$Season=="Summer",]

cat(paste0("The difference in number of rows is approximately: ", round(nrow(summer_dataset) /
```

```
The difference in number of rows is approximately: -0.23%, or -38 days
```

```
grid.arrange(ncol=2,
ggplot(winter_dataset)+
  geom_histogram(aes(x = T_MAX, fill = "T_MAX"), alpha=0.8, bins = 80)+
  geom_histogram(aes(x = T_MIN, fill = "T_MIN"), alpha=0.8, bins = 80)+
```

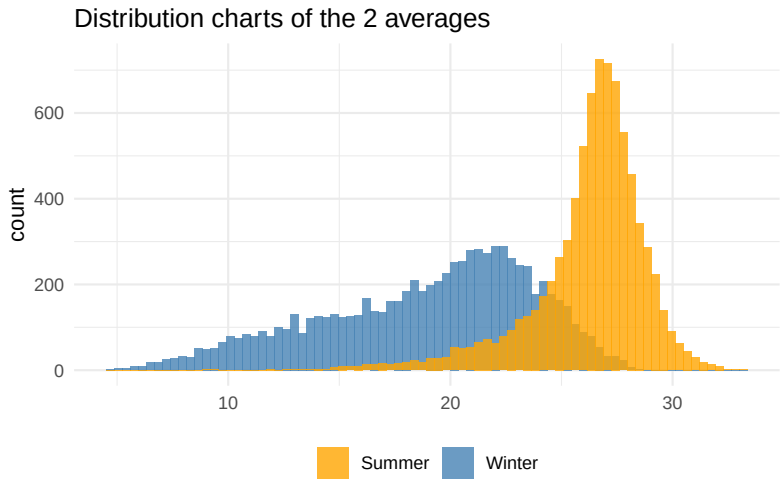
```
  geom_histogram(aes(x = T_AVG, fill = "T_AVG"), alpha=0.8, bins = 80)+
  labs(title = "Distribution charts of winter",x=NULL)+
  scale_fill_manual(name = NULL,
                    values = c(T_MAX = "indianred3",T_MIN = "lightblue",T_AVG = "lightgreen"))+
  theme(legend.position = "bottom")

,

ggplot(summer_dataset)+
  geom_histogram(aes(x = T_MAX, fill = "T_MAX"), alpha=0.8, bins = 80)+
  geom_histogram(aes(x = T_MIN, fill = "T_MIN"), alpha=0.8, bins = 80)+
  geom_histogram(aes(x = T_AVG, fill = "T_AVG"), alpha=0.8, bins = 80)+
  labs(title = "Distribution charts of summer",x=NULL)+
  scale_fill_manual(name = NULL,
                    values = c(T_MAX = "indianred3",T_MIN = "lightblue",T_AVG = "lightgreen"))+
  theme(legend.position = "bottom")

)
```



Its with this choice of season im still able to keep the data very balanced. clear to see that there

Lastly I'm able to plot just the two average temperatures of winter and of summer for the average temperatures

```
ggplot()+
  geom_histogram(data = winter_dataset, aes(x = T_AVG, fill = "Winter"), alpha=0.8, bins = 80)+
  geom_histogram(data = summer_dataset, aes(x = T_AVG, fill = "Summer"), alpha=0.8, bins = 80)+
  labs(title = "Distribution charts of the 2 averages",x=NULL)+
    scale_fill_manual(name = NULL,
                      values = c(Winter="steelblue", Summer="orange"))+
  theme(legend.position = "bottom")
```



Just to understand the records of my dataset i wanted to see what months on record had the hottest and coldest days across all dataset

```
DATASET_seas %>%
  group_by(Month) %>%
  summarise(T_min_min = min(T_MIN),
            T_min_max = max(T_MIN),
            T_max_min = min(T_MAX),
            T_max_max = max(T_MAX)) %>%
  round(3) %>%
  mutate(Month = month.abb) %>%
  gt() %>%
  opt_stylize(5) %>%
  tab_header("Data exploration", "what is the highest and lowest in my database") %>%
```

Data exploration

what is the highest and lowest in my database

	T_MIN		T_MAX	
Month	Min	Max	Min	Max
Jan	-4.08	20.90	12.59	29.78
Feb	-3.58	20.24	12.62	32.49
Mar	-1.26	22.43	13.21	35.29
Apr	1.63	23.82	14.38	36.70
May	9.48	25.50	21.34	39.12
Jun	13.06	26.49	24.94	40.30
Jul	19.55	27.13	26.31	38.58
Aug	19.61	27.31	25.72	37.71
Sep	13.40	26.03	23.38	35.13
Oct	2.68	25.56	15.78	34.12
Nov	1.66	24.50	12.88	32.13
Dec	-3.90	22.12	12.61	30.43

```
  cols_label(T_min_min = "Min",
             T_min_max = "Max",
             T_max_min = "Min",
             T_max_max = "Max") %>%
  tab_spanner(label = "T_MIN", columns = 2:3) %>%
  tab_spanner(label = "T_MAX", columns = 4:5)
```

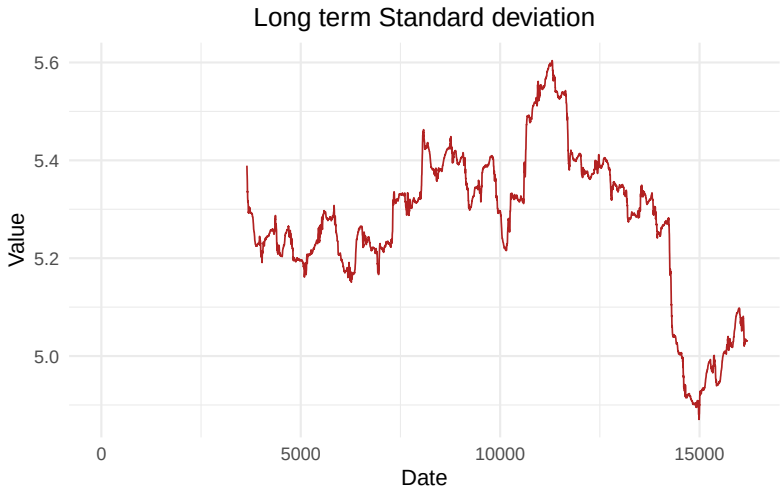
2.3 Long term trends

```
SMA(DATASET$T_AVG, lookback) %>% quickplot(show_legend = F, title = "Long term trend")
```

```
Warning: Removed 3649 rows containing missing values or values outside the scale range
(`geom_line()`).
```


```
runSD(DATASET$T_AVG, lookback) %>% quickplot(show_legend = F, title = "Long term Standard devia
```

```
Warning: Removed 3649 rows containing missing values or values outside the scale range
(`geom_line()`).
```



3 Seasonal decomposition

The first step in pricing temperature options is to decompose the time series into distinct components that each represent different underlying patterns. This process is essential for understanding the structure of the data and for developing predictive models. In basic temrs it can be expressed as the sum of three main components:$y_t = T_t + S_t + e_t$, Here, T_t is the trend component that captures the long-term movement, S_t represents the seasonal variations (periodic patterns), and e_t refers to the residuals or noise in the data.

which in our case becomes

$$\overline{T_t} = T_{trend} + T_{seasonal} + Residuals$$

The approach was to develop each component indipendently

$$T_{trend} = a + b \cdot t$$

:

The trend component could be as simple as a straight line or a moving average

This form captures the cyclical nature of temperature changes throughout the year, like daily or annual temperature fluctuations.

$$T_{seasonal} = \alpha \cdot \sin(\omega \cdot t + \theta)$$

- α represents the amplitude
- ω is the angular frequency (related to the period of the cycle)
- θ is the phase shift

And lastly the residuals are just:

$$Residuals = T_t - T_{trend} + T_{seasonal}$$

Recent research has suggested that partial Fourier decomposition can be effective in modeling temperature data, particularly by omitting the cosine component. This approach simplifies the seasonal model while still accurately representing the cyclical nature of temperature changes. The sinusoidal term is often sufficient to describe seasonal temperature variations, making the model both interpretable and computationally efficient.

Before this I used a denoising procedure using Gaussian Convolution Filter as it also enhances the effectiveness of the Fourier transformation by reducing noise, allowing you to identify the dominant frequencies or cycles in your temperature data more accurately.

- **Gaussian Convolution Filter**:
  - A **convolution** is a mathematical operation used to blend or smooth data. It involves applying a kernel (a small array of weights) across the data to compute a weighted average. The Gaussian kernel is a specific type of convolution filter that assigns weights based on the Gaussian (normal) distribution, which is a bell-shaped curve.
  - The **Gaussian kernel** is widely used for smoothing because it gives more weight to the central points (closer to the middle) while gradually reducing the weights for points further away. This is mathematically expressed as:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

- **Moving Window**: The Gaussian kernel slides across each point in the time series. For each position, it computes a weighted sum where the weights are defined by the Gaussian kernel.
- **Local Averaging**: The computed value represents a smoothed average of the points within the window's range. Points closer to the center of the window contribute more heavily to the average than those farther away, as per the Gaussian weights.

- **Symmetric Filtering**: using 2 sides makes this operation symmetric, applying the kernel both forward and backward across the data, ensuring the filtering effect is centered.
- **Conclusion and data inspection**: in visual terms the denoised data should look smoother with slightly lower peaks

```
temps <- DATASET
apply_convolution <- function(x, kernel) {
  # Use filter() from stats package to apply convolution
  filtered <- stats::filter(x, kernel, sides = 2)  # Use sides = 2 for symmetric filter
  return(filtered)
}

# DENOISED - convolution with a window of 90 days
kernel <- dnorm(-3:3)
data.frame("Gaussian_Kernel" = round(kernel, 10))
```

Gaussian_Kernel
0.0044318
0.0539910
0.2419707
0.3989423
0.2419707
0.0539910
0.0044318

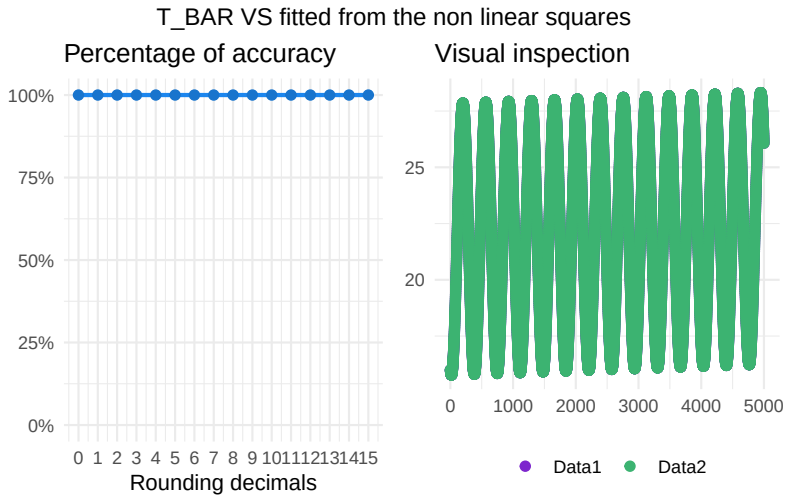
```
temps$Denoised <- apply_convolution(temps$T_AVG, kernel)
temps$Denoised <- na.fill(temps$Denoised, mean(temps$Denoised, na.rm = TRUE))

temps$Trend <- SMA(temps$Denoised, n = lookback)

library(nlme, quietly = T, warn.conflicts = F)

# Define the model
sin_component <- function(t, a, b, alpha, theta) {
  omega <- 2 * pi / 365.25
```

```
  a + b * t + alpha * sin(omega * t + theta)
}
omega <- 2 * pi / 365.25

# Fit model using non linear squares
temps$NUM_DAY <- 1:nrow(temps)

fit <- nls(Denoised ~ sin_component(NUM_DAY, a, b, alpha, theta),
           data = temps,
           start = list(a = 1, b = 0, alpha = 1, theta = 0))


# Get coefficients and confidence intervals for the model
params <- coef(fit)
confint_fit <- suppressMessages(confint(fit))

temps$SEAS <- params["alpha"] * sin(omega * temps$NUM_DAY + params["theta"])
temps$TREND <- params["a"] + params["b"] * temps$NUM_DAY
temps$BAR <- temps$TREND + temps$SEAS
temps$RESID <-  temps$T_AVG - temps$TREND - temps$SEAS
```

just to be certain i looked at if the residuals and the fitted values matched the $\overline{T}$ and *Residuals* by checking percentage of same values as the rounding decimals increase The logic is that if the data sets are anything alike (like 2 different models results for the same dataset) you should see how far in the rounding are the data sets similar

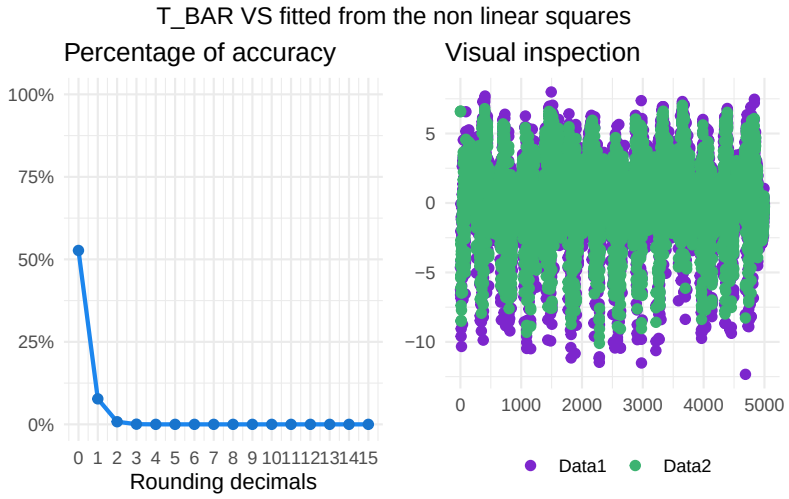
```
check_acc(temps$BAR, fitted(fit),15, title = "T_BAR VS fitted from the non linear squares")
```

```
Scale for y is already present.
Adding another scale for y, which will replace the existing scale.
```




```
check_acc(temps$RESID, residuals(fit),15, title = "T_BAR VS fitted from the non linear squares"
```

```
Scale for y is already present.
Adding another scale for y, which will replace the existing scale.
```



3.1 Model performance

for assessing my model performance I'm looking at the values and confidence intervals and looking at the Residual sum of

squares and the mean absolute error

- The **RSS function** calculates the Residual Sum of Squares, which is a measure of the discrepancy between the observed data and the model's predictions. A lower RSS indicates a better fit of the model to the data, meaning that the model's predictions are closer to the actual observed values.
- The **MAE function** is the average of the absolute differences between the observed and predicted values. It gives an idea of how far, on average, the predictions are from the actual values, without considering the direction of the error. A lower MAE indicates that the model's predictions are generally close to the observed data.

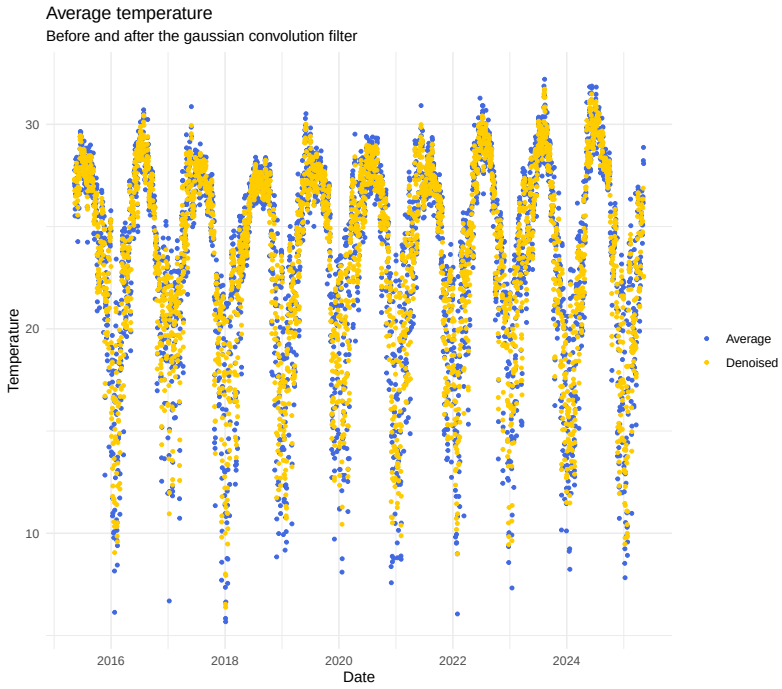
```
a :  21.801  CI ~normally [ 21.718 , 21.883 ]
b :  0  CI ~normally [ 0 , 0 ]
alpha :  -6.034  CI ~normally [ -6.092 , -5.975 ]
theta :  -5.008  CI ~normally [ -5.018 , -4.998 ]
```

```
  RSS model sine curve: 157805.5
```

```
  MAE model fit: 2.35
```

3.2 Visualization of results

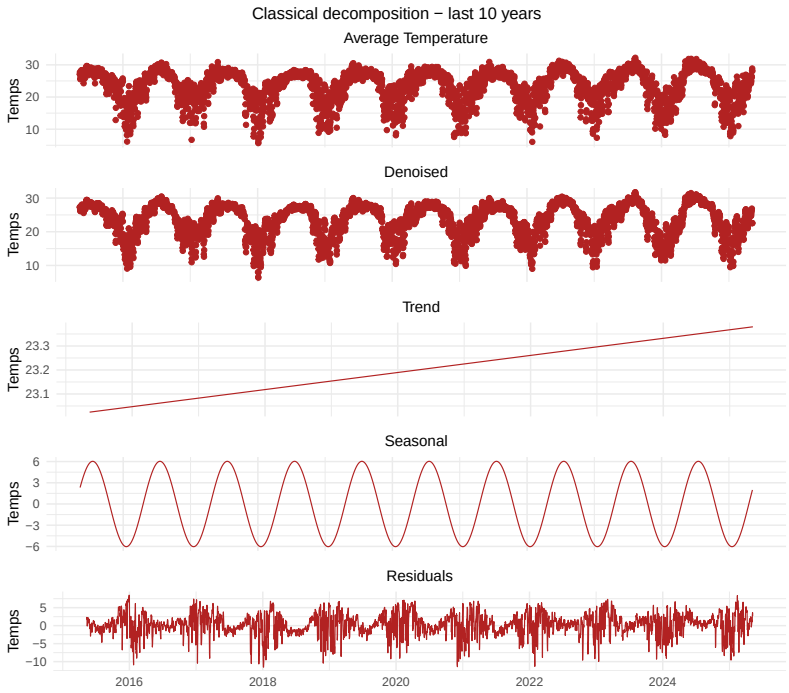
```
# plot denoised
ggplot(tail(temps, lookback), aes(x=DAY))+
  geom_point(aes(y = T_AVG, color = "Average"), size = 1)+
  geom_point(aes(y = Denoised, color = "Denoised"), size = 1)+
  scale_color_manual(name=NULL, values = c(Average = "royalblue", Denoised = "#ffcc00"))+
  labs(title = "Average temperature", x = "Date", y = "Temperature",
      subtitle = "Before and after the gaussian convolution filter")
```



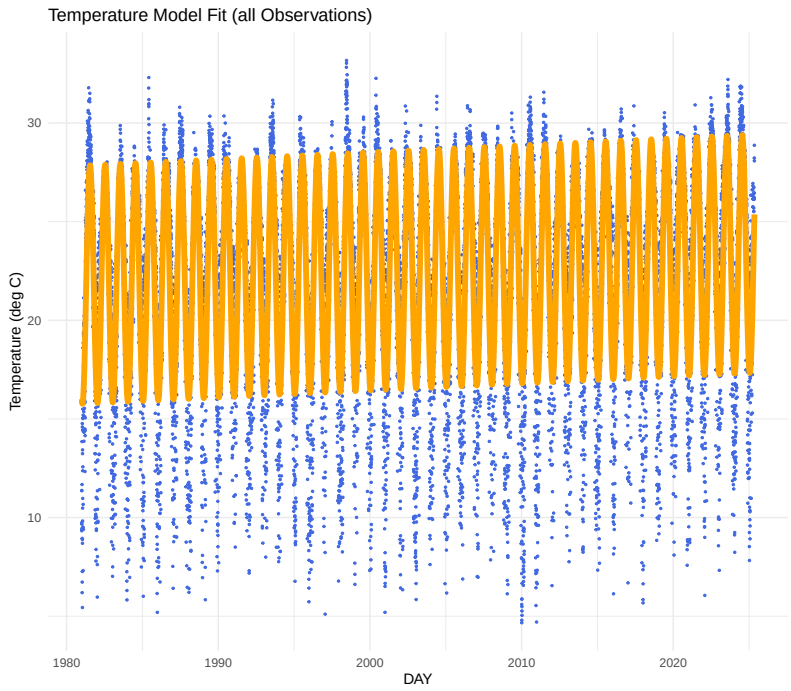

```
temps_xts <- temps %>%
  select(T_AVG, Denoised, TREND, SEAS, RESID) %>%
  xts(order.by = temps$DAY) %>%
  tail(lookback)

# Plot seasonal decomposition from Avg to residuals
grid.arrange(nrow=5, top = paste0("Classical decomposition - last ",  lookback/365, " years"),

  temps_xts$T_AVG %>%
    quickplot(subtitle = "Average Temperature", show_legend = F, xlab = NULL, ylab = "Temps",
    type = geom_point, show_x = F),

  temps_xts$Denoised %>%
    quickplot(subtitle = "Denoised", show_legend = F, xlab = NULL, ylab = "Temps",
    type = geom_point, show_x = F),

  temps_xts$TREND %>%
    quickplot(subtitle = "Trend", show_legend = F, xlab = NULL, ylab = "Temps" , show_x = F),

  temps_xts$SEAS %>%
```

```
    quickplot(subtitle = "Seasonal", show_legend = F, xlab = NULL, ylab = "Temps" , show_x = F

  temps_xts$RESID %>%
    quickplot(subtitle = "Residuals", show_legend = F, xlab = NULL, ylab = "Temps"))
```

```
Warning in type(linewidth = linewidth, alpha = alpha): Ignoring unknown parameters: `linewidth`
Ignoring unknown parameters: `linewidth`
```




```
# Plot original vs. fitted data
ggplot(temps, aes(x = DAY)) +
  geom_point(aes(y = T_AVG), color = 'royalblue', size = 0.5) +
  geom_line(aes(y = BAR), color = 'orange', linewidth=2) +
  labs(title = "Temperature Model Fit (all Observations)", y = "Temperature (deg C)")
```

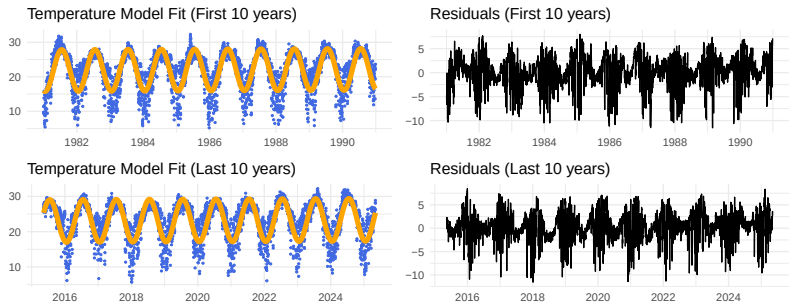


3.2.1 Check for possible model degradation

Checking for model degradation across time by looking at the first and last 10 years to see if there is a meaningful shift in where the sine curve and the average temperature

```
grid.arrange(nrow = 2, ncol = 2,
ggplot(temps %>% head(lookback), aes(x = DAY)) +
  geom_point(aes(y = T_AVG), color = 'royalblue', size = 0.5) +
  geom_line(aes(y = BAR), color = 'orange', linewidth=2) +
  labs(title = paste0("Temperature Model Fit (First ", lookback/365, " years)"),x=NULL, y = NUL

ggplot(temps %>% head(lookback), aes(x = DAY)) +
  geom_line(aes(y = RESID), color = 'black', linewidth=0.5) +
  labs(title = paste0("Residuals (First ", lookback/365, " years)"),x=NULL, y = NULL),

ggplot(temps %>% tail(lookback), aes(x = DAY)) +
  geom_point(aes(y = T_AVG), color = 'royalblue', size = 0.5) +
  geom_line(aes(y = BAR), color = 'orange', linewidth=2) +
  labs(title = paste0("Temperature Model Fit (Last ", lookback/365, " years)"), x=NULL, y = NUL
```

```
ggplot(temps %>% tail(lookback), aes(x = DAY)) +
  geom_line(aes(y = RESID), color = 'black', linewidth=0.5) +
  labs(title = paste0("Residuals (Last ", lookback/365, " years)"),x=NULL, y = NULL)
)
```



3.3 Residuals analysis and diagnostics

1. **Autocorrelation Function (ACF) Plot of Residuals**:

   The ACF plot checks for any correlation between residuals at different lags. If residuals are uncorrelated (i.e., resemble white noise), all values should fall within the confidence bands, indicating no significant autocorrelation.

2. **Partial Autocorrelation Function (PACF) Plot of Residuals**:

   The PACF plot displays the partial correlation of residuals with their own lagged values, considering the effect of any intermediate lags. Like the ACF plot, if the model fits well, the PACF values should also fall within the confidence bands, showing no significant partial autocorrelations.

3. **Normality Check using QQ Plot**:

   The QQ plot assesses whether the residuals are normally distributed. Points should lie along the reference line; significant deviations from this line suggest that the residu-

als do not follow a normal distribution, which could imply model misspecification or the presence of outliers.

4. **Histogram of Residuals with Normal Curve**:

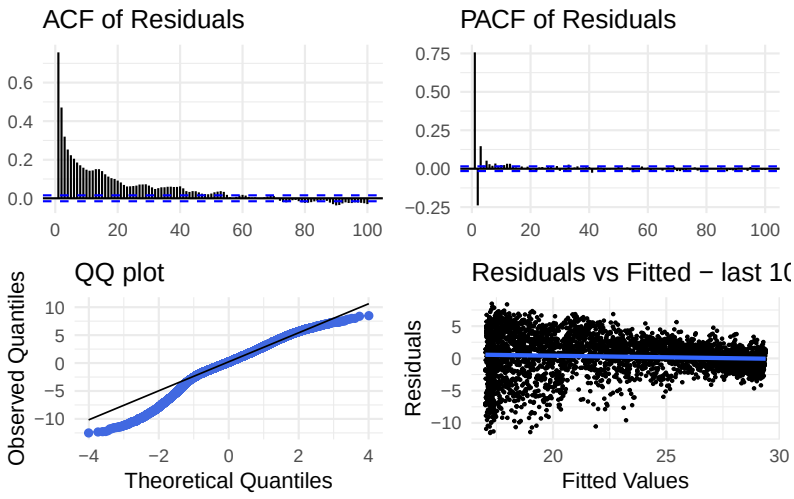
   The histogram visualizes the distribution of residuals, and the overlaid normal curve helps assess normality. If the histogram closely follows the bell-shaped curve, it suggests normal residuals. The vertical line representing kurtosis indicates whether the residuals are more or less peaked than a normal distribution.

5. **Residuals vs. Fitted Values Plot**:

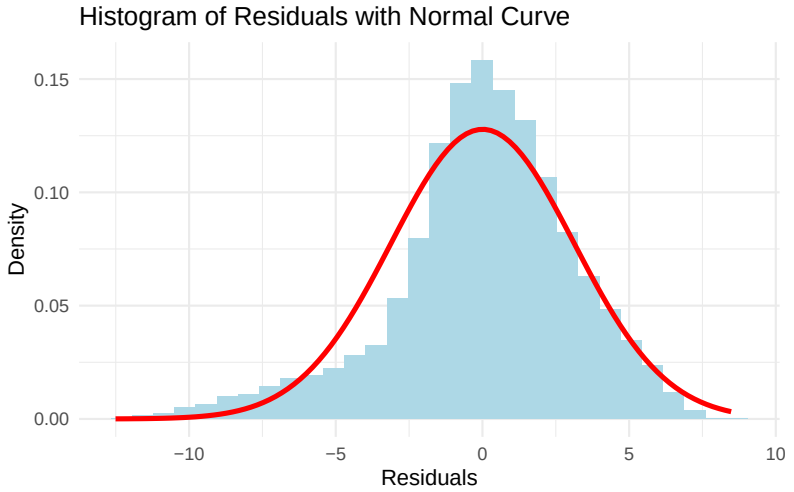
   The plot checks for patterns in the residuals against fitted values. Ideally, the residuals should be randomly scattered around zero, indicating homoscedasticity (constant variance). Any visible pattern or trend suggests issues like non-linearity, heteroscedasticity, or omitted variables.

```
grid.arrange(nrow = 2,
# ACF
ggAcf(temps$RESID, lag.max = 100)+
  labs(title = "ACF of Residuals", x = NULL, y = NULL),

# PACF
ggPacf(temps$RESID, lag.max = 100)+
  labs(title = "PACF of Residuals", x = NULL, y = NULL),

## Check normality of residuals using QQ plot
ggplot(temps, aes(sample = RESID)) +
  stat_qq(color="royalblue")+
  stat_qq_line(color = "black", linewidth = 0.4)+
  labs(title = "QQ plot", x="Theoretical Quantiles", y= "Observed Quantiles"),

## Check for heteroskedasticity or any pattern in residuals
ggplot(temps %>% tail(lookback), aes(x=BAR, y = RESID))+
  geom_point(size = 0.4)+
  geom_smooth(method = "lm")+
  labs(title = paste0("Residuals vs Fitted - last ",  lookback/365, " years"), x= "Fitted Value
```

```
`geom_smooth()` using formula = 'y ~ x'
```




```
# Histogram with bell curve and kurtosis
ggplot(temps, aes(x = RESID)) +
  geom_histogram(aes(y = after_stat(density)), fill = "lightblue", bins = 30) +
  stat_function(fun = dnorm, args = list(mean = mean(temps$RESID), sd = sd(temps$RESID)),
                color = "red", linewidth = 1.2) +
  # geom_vline(xintercept = skewness(temps$RESID)[1], linetype = "dashed", linewidth=1, color =
  labs(title = "Histogram of Residuals with Normal Curve", x = "Residuals", y = "Density",
       # subtitle = "Vertical line is kurtosis"
       )
```



4 Ornstein-Uhlenbeck (OU) process

Temperature dynamics using a mean-reverting stochastic process like the Ornstein-Uhlenbeck (OU) process, incorporate the seasonal adjustment into the drift rate to ensure that the expected value of the temperature follows the seasonal mean temperature.

The Ornstein-Uhlenbeck (OU) process is a type of stochastic (random) process that is often used in physics, finance, and other fields to model systems that exhibit some form of "mean-reverting" behavior. Let's start from the basics to understand what this means.

A stochastic process is a collection of random variables representing the evolution of a system over time. In simpler terms, it describes how something changes when randomness is involved. For example, the price of a stock or the position of a particle under the influence of random forces can be modeled as a stochastic process.

$$dX_t = \kappa(\mu - X_t)\ d_t + \sigma \cdot dW_t$$

- X_t is the value of the process at time t.
- μ is the long-term mean or equilibrium level toward which the process reverts.
- $\theta > 0$ is the rate of mean reversion, determining how fast the process reverts to the mean P.
- $\sigma > 0$ is the volatility or the intensity of the random fluctuations.
- dW_t is an infinitesimal increment of a Wiener process (Brownian motion).

Mean Reversion Term: $\kappa(\mu - X_t)\ d_t$: Representing the tendency of the process to revert to the mean μ. If $X_t$is above the mean μ, this term will be negative (pulling it down toward the mean). If X_t is below μ, the term will be positive (pushing it up toward the mean).

The speed of this pull or push is determined by κ. A higher κ means the process will revert to the mean faster.

Random Fluctuation Term $\sigma \cdot dW_t$: represents the random noise or shock. This is a source of randomness, and it causes the process to deviate randomly over time.

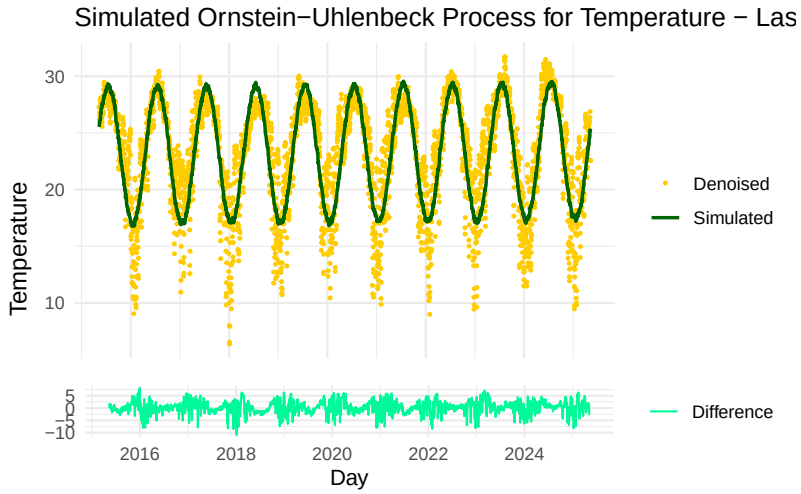
The size of the noise is controlled by σ, which is the volatility parameter.

```
temps_OU <- temps
# Define parameters for the OU process
kappa <- 1-arima(temps_OU$RESID, order = c(1,0,0))$coef[1]  # Mean-reversion rate
sigma <- 0.1                                               # Volatility of the process
dt <- 1                                                    # Time step (daily data)

cat("Kappa is estimated as:", round(kappa,4))
```

```
Kappa is estimated as: 0.2431
```

```
# Initialize variables for simulation
n <- nrow(temps_OU)                        # Number of time points
T_simulated <- numeric(n)                  # Simulated temperature vector
T_simulated[1] <- temps_OU$Denoised[1]   # Set initial temperature to the first observed value

# Simulate the seasonal mean as a time-varying mean (trend + seasonal component)
T_bar <- temps_OU$BAR

# Simulate the modified OU process
for (i in 2:n) {
  # Rate of change of the seasonal mean
  dT_bar_dt <- (T_bar[i] - T_bar[i - 1]) / dt

  # Brownian motion increment
  dWt <- rnorm(1, mean = 0, sd = sqrt(dt))

  T_simulated[i] <- T_simulated[i - 1] +
                    (dT_bar_dt + kappa * (T_bar[i] - T_simulated[i - 1])) * dt +
                    sigma * dWt
}

temps_OU$OU <- T_simulated
```

```
DATE <- tail(temps$DAY, lookback)

grid.arrange(layout_matrix = matrix(data = c(1,1,1,2), ncol = 1), left = "Temperature",
temps_OU %>%
  select(Denoised, OU, BAR) %>%
  tail(lookback) %>%
  ggplot(aes(x = DATE)) +
  geom_point(aes(y = Denoised, color = "Denoised"), size = 0.5) +
  geom_line(aes(y = OU, color = "Simulated"), linewidth = 0.8) +
  scale_color_manual(name = NULL, values = c(Denoised = "#ffcc00", Simulated = "darkgreen")) +
  labs(title = "Simulated Ornstein-Uhlenbeck Process for Temperature - Last 10 years", x = NULL
  theme(axis)+
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
,
temps_OU %>%
  select(Denoised, OU, BAR) %>%
  tail(lookback) %>%
  ggplot(aes(x = DATE)) +
  geom_line(aes(y = Denoised-OU, color = "Difference"), linewidth = 0.5) +
  scale_color_manual(name = NULL, values = c(Difference = "mediumspringgreen")) +
  labs(x = "Day", y = NULL)
)
```



5 Modeling volatility

5.1 Polynomial Regression:

Fit polynomial curves to model the relationship between day-of-year and volatility.

```
# Create a dataframe with temperature and date components
temp_t <- data.frame(
  Date = temps$DAY,  # Assuming the index is a proper Date object
  T = temps$Denoised,
  day = yday(temps$DAY),
  month = month(temps$DAY)
)

# 2. Calculate monthly volatility statistics
vol1 <- temps %>%
  group_by(YEAR, Month) %>%
  summarise(
    mean_temp = mean(Denoised, na.rm = TRUE),
    std_temp = sd(Denoised, na.rm = TRUE)
  ) %>%
  ungroup()
```

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

```
# 5. Volatility Analysis
    vol <- temps %>%
      select(DAY, T_AVG) %>%
      mutate(day = yday(DAY),
             month = month(DAY)) %>%
      group_by(day) %>%
      summarise(mean = mean(T_AVG, na.rm = TRUE),
                std = sd(T_AVG, na.rm = TRUE))
```

5.2 B-splines

Flexible non-parametric method that fits piecewise ials (splines) at different intervals (knots).

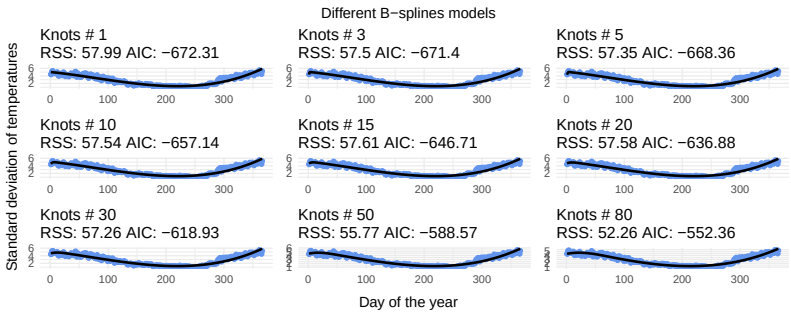
Avoids overfitting by controlling the number of knots.

AIC (Akaike Information Criterion): Helps select the best model by balancing goodness-of-fit and complexity (lower AIC preferred).

The residual sum of squares (RSS) measures the level of variance in the error term, or residuals, of a regression model. The smaller the residual sum of squares, the better your model fits your data; the greater the residual sum of squares, the poorer your model fits your data.

```
library(splines)
x <- 1:366
y <- vol$std

# Define the number of knots
knots <- c(1, 3, 5, 10, 15, 20, 30, 50, 80)
x <- 1:366
# Function to create a spline model and plot
create_spline_plot <- function(knots, x, y) {
  # Fit the spline model
  spline_model <- lm(y ~ bs(x, knots = knots))
  yfit <- predict(spline_model, data.frame(x = x))

  # Calculate Residual Sum of Squares (RSS)
  rss <- sum((y - yfit)^2)
  aic <- length(x) * log(rss / length(x)) + 2 * knots

  # Create the plot
  plots <- ggplot() +
    geom_point(aes(x, y), color = 'cornflowerblue', size = 1.5) +
    geom_line(aes(x, yfit), color = 'black', linewidth = 1) +
    ggtitle(paste("Knots #", knots, "\nRSS:", round(rss, 2), "AIC:", round(aic,2))) +
    labs(x = NULL, y = NULL)
    theme_classic()+
    theme(plot.title = element_text(size = 10, face = "bold"))

  return(plots)
}

# Generate all the plots
```

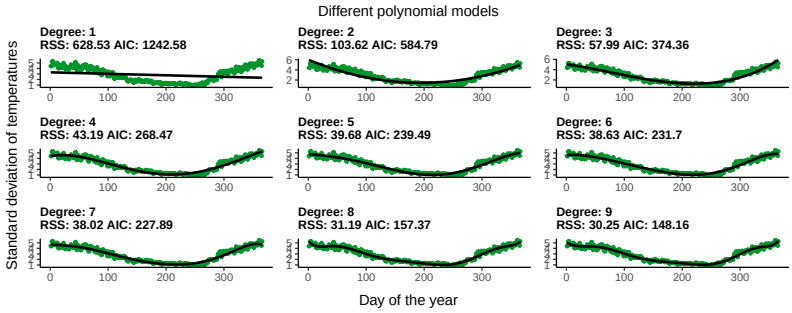
```r
plots <- lapply(knots, create_spline_plot, x = x, y = y)

# Arrange and display the plots in a 2x3 grid
grid.arrange(grobs = plots, nrow = 3, ncol = 3, left = "Standard deviation of temperatures", bo
```



5.3 Polynomial models to fit volatility

```r
# Function to create polynomial model and plot
x <- 1:366
create_poly_plot <- function(degree, x, y) {
  # Fit polynomial model
  model <- lm(y ~ poly(x, degree, raw = TRUE))
  yfit <- predict(model, data.frame(x = x))

  # Calculate metrics
  rss <- sum(resid(model)^2)
  aic <- AIC(model)

  # Create plot
  ggplot() +
    geom_point(aes(x, y), color = '#00962B', size = 1) +
    geom_line(aes(x, yfit), color = 'black', linewidth = 1) +
    ggtitle(paste("Degree:", degree, "\nRSS:", round(rss, 2), "AIC:", round(aic, 2))) +
    labs(y = NULL, x = NULL)+
    theme_classic()+
    theme(plot.title = element_text(size = 10, face = "bold"))
  }
```

```r
# Generate plots for degrees 1 through 9
degrees <- 1:9
plots <- lapply(degrees, create_poly_plot, x = x, y = y)

# Arrange in 3x3 grid
grid.arrange(grobs = plots, nrow = 3, ncol = 3, left = "Standard deviation of temperatures", bo
```



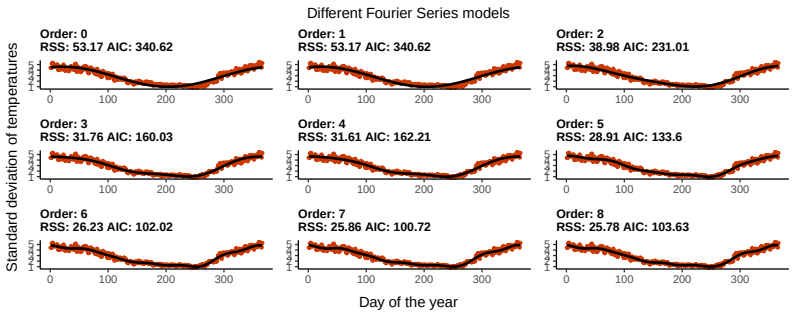
5.4 Fourier transforms

```r
fourier_series <- function(x, n_terms, period = 365.25) {
  omega <- 2 * pi / period
  terms <- list(a0 = 1)

  for (i in 1:n_terms) {
    terms[[paste0("a", i)]] <- cos(i * omega * x)
    terms[[paste0("b", i)]] <- sin(i * omega * x)
  }
  return(terms)
}

fourier_series(1:366, 3) %>%
  data.frame("actual" = normalize(y, 1)) %>%
  quickplot(
    title = "Fourier transformation different components with actual volatility",
    subtitle = "Order 3 so 7 components",
    xlab = "Day of the year"
  )
```

Fourier transformation different components with actual volatility


```
create_fourier_plot <- function(n_terms, x, y) {
  # Fit Fourier series
  fourier_terms <- fourier_series(1:366, n_terms = n_terms)
  model_data <- data.frame(y = y, fourier_terms)

  fourier_fit <- lm(y ~ ., data = model_data)
  fourier <- predict(fourier_fit, newdata = model_data)

  # Calculate metrics
  rss <- sum(resid(fourier_fit)^2)
  aic <- AIC(fourier_fit)

  # Create plot
  ggplot(data = data.frame(), aes(x = x)) +
    geom_point(aes(y = y), color = 'orangered3', size = 1) +
    geom_line(aes(y = fourier), color = 'black', linewidth = 1) +
    ggtitle(paste("Order:", n_terms, "\nRSS:", round(rss, 2), "AIC:", round(aic, 2))) +
    labs(y = NULL, x = NULL) +
    theme_classic() +
    theme(plot.title = element_text(size = 10, face = "bold"))
}

# Generate plots for degrees 1 through 9
order <- c(0:8)
```

```
plots <- lapply(order, create_fourier_plot, x = x, y = y)

# Arrange in 3x3 grid
grid.arrange(grobs = plots, nrow = 3, ncol = 3,
  left = "Standard deviation of temperatures",
  bottom = "Day of the year",
  top = "Different Fourier Series models")
```



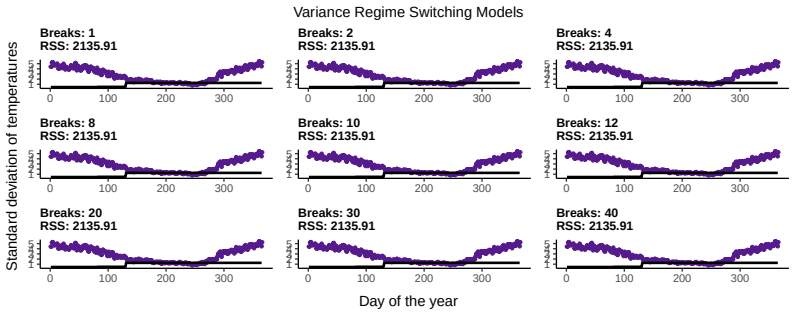
5.5 Regime switch

```
library(changepoint, quietly = T, warn.conflicts = F)
```

```
Successfully loaded changepoint package version 2.3
 WARNING: From v.2.3 the default method in cpt.* functions has changed from AMOC to PELT.
 See NEWS for details of all changes.
```

```
create_changepoint_plot <- function(n_breaks, x, y) {
  # Detect changepoints in variance with fixed number of breaks
  cp <- cpt.var(y, method = "PELT", Q = n_breaks)

  # Get regime-wise standard deviation
  regime_sd <- rep(NA, length(y))
  segs <- c(1, cpts(cp), length(y))
  for (i in 1:(length(segs) - 1)) {
    idx <- segs[i]:(segs[i + 1] - 1)
    regime_sd[idx] <- sd(y[idx])
```

```
  }

  # RSS (within segment variance sum)
  rss <- sum((y - na.omit(regime_sd))^2)

  # Build plot
  data_plot <- data.frame(x = x, y = y, sd_fit = regime_sd)
  ggplot(data_plot, aes(x = x)) +
    geom_point(aes(y = y), color = 'purple4', size = 1) +
    geom_line(aes(y = sd_fit), color = 'black', linewidth = 1) +
    ggtitle(paste("Breaks:", n_breaks, "\nRSS:", round(rss, 2))) +
    theme_classic() +
    labs(y = NULL, x = NULL) +
    theme(plot.title = element_text(size = 10, face = "bold"))
}
# Create 9 plots for rhe breaks
breaks <- c(1, 2, 4, 8, 10, 12, 20, 30, 40)
plots <- suppressWarnings(lapply(breaks, create_changepoint_plot, x = x, y = y))

suppressWarnings(grid.arrange(grobs = plots, nrow = 3, ncol = 3,
  left = "Standard deviation of temperatures",
  bottom = "Day of the year",
  top = "Variance Regime Switching Models"))
```




```
# 3. Print long-term volatility metrics
cat("Trend or long term volatility is easy: ~", round(mean(vol1$std_temp, na.rm = TRUE), 3), "
```

```
Trend or long term volatility is easy: ~ 2.024
```

```r
cat("Gamma is:", round(sd(vol1$std_temp, na.rm = TRUE), 3), "\n")
```

```
Gamma is: 1.125
```

```r
# 4. Fit AR(1) model for mean reversion rate
ar_model <- arima(vol1$std_temp, order = c(1, 0, 0), include.mean = FALSE)
coef <- ar_model$coef
residuals <- ar_model$residuals

cat("Rate of mean reversion of volatility process is:", round(coef["ar1"], 3), "\n")
```

```
Rate of mean reversion of volatility process is: 0.917
```

```r
summary(ar_model)
```

```
Call:
arima(x = vol1$std_temp, order = c(1, 0, 0), include.mean = FALSE)

Coefficients:
         ar1
      0.9173
s.e.  0.0172

sigma^2 estimated as 0.8632:  log likelihood = -718,  aic = 1440

Training set error measures:
                     ME      RMSE       MAE       MPE    MAPE      MASE
Training set 0.1660121 0.9290675 0.6942639 -4.485931 39.3609 0.9684978
                    ACF1
Training set -0.1142199
```

$$dT_t = \left(\frac{dT_{bar}(t)}{dt} + \kappa(T_{bar}(t) - T_t) \right) dt + \sigma(t)\, dW_t$$

6 Montecarlo simulations

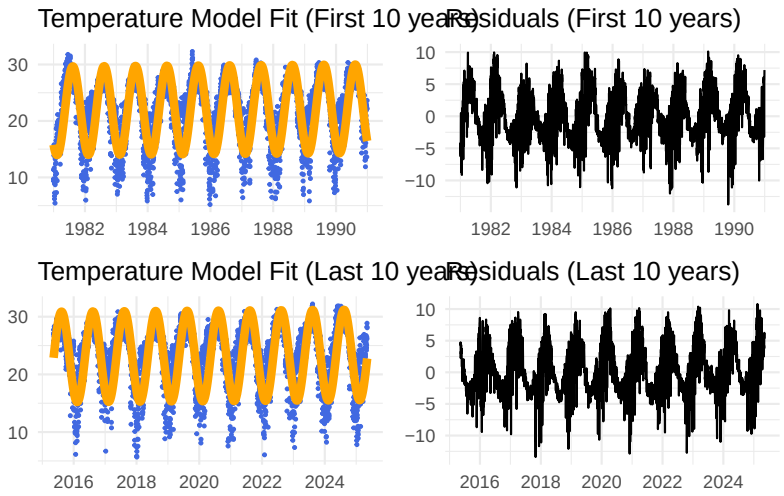
```
a <- params[1]
b <- params[2]
theta <- atan2(params[3], params[4])
alpha <- sqrt(params[3]^2 + params[4]^2)
kappa <- as.double(kappa)

data.frame(a = a, b = b, theta = theta, alpha = alpha, kappa = kappa)
```

	a	b	theta	alpha	kappa
a	21.80081	9.75e-05	-2.263556	7.841268	0.2430516

```
# 1. Temperature Model Functions
T_model <- function(x, a, b, alpha, theta) {
  omega <- 2 * pi / 365.25
  a + b * x + alpha * sin(omega * x + theta)
}
dT_model <- function(x, a, b, alpha, theta) {
  omega <- 2 * pi / 365.25
  b + alpha * omega * cos(omega * x + theta)
}

# 2. Prepare Data
if (inherits(temps$DAY, "Date")) {
  first_ord <- as.numeric(temps$DAY[1])
  temp_t$ordinal <- as.numeric(temps$DAY)
} else {
  temp_t$Date <- as.Date(temps$DAY)
  first_ord <- as.numeric(temps$DAY[1])
  temp_t$ordinal <- as.numeric(temps$DAY)
}

# 3. Apply Model with Given Parameters
Tbar_params <- list(a = a, b = b, alpha = alpha, theta = theta)

temps$model_fit <- T_model(
  temp_t$ordinal - first_ord,
```

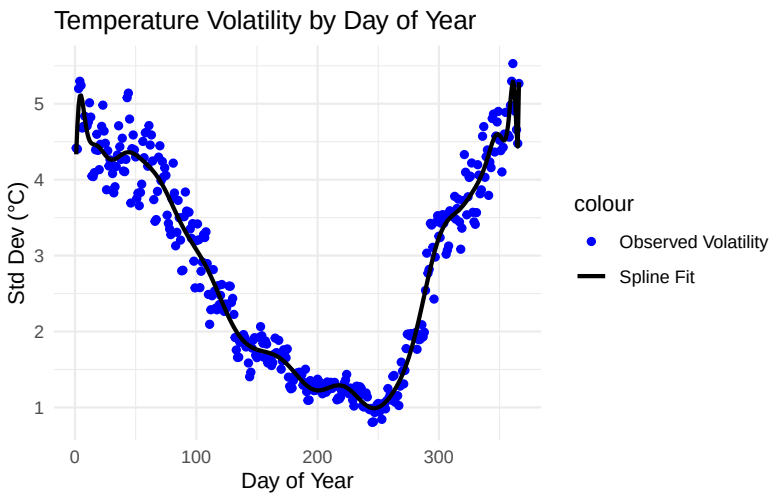
```
  Tbar_params$a,
  Tbar_params$b,
  Tbar_params$alpha,
  Tbar_params$theta
)

grid.arrange(
  nrow = 2,
  ncol = 2,
  ggplot(temps %>% head(lookback), aes(x = DAY)) +
    geom_point(aes(y = T_AVG), color = 'royalblue', size = 0.5) +
    geom_line(aes(y = model_fit), color = 'orange', linewidth = 2) +
    labs(
      title = paste0(
        "Temperature Model Fit (First ",
        lookback / 365,
        " years)"
      ),
      x = NULL,
      y = NULL
    ),

  ggplot(temps %>% head(lookback), aes(x = DAY)) +
    geom_line(aes(y = T_AVG - model_fit), color = 'black', linewidth = 0.5) +
    labs(
      title = paste0("Residuals (First ", lookback / 365, " years)"),
      x = NULL,
      y = NULL
    ),

  ggplot(temps %>% tail(lookback), aes(x = DAY)) +
    geom_point(aes(y = T_AVG), color = 'royalblue', size = 0.5) +
    geom_line(aes(y = model_fit), color = 'orange', linewidth = 2) +
    labs(
      title = paste0("Temperature Model Fit (Last ", lookback / 365, " years)"),
      x = NULL,
      y = NULL
    ),

  ggplot(temps %>% tail(lookback), aes(x = DAY)) +
```

```
    geom_line(aes(y = T_AVG - model_fit), color = 'black', linewidth = 0.5) +
    labs(
      title = paste0("Residuals (Last ", lookback / 365, " years)"),
      x = NULL,
      y = NULL
    )
)
```




```
# 6. Spline Fit for Volatility
spline_fit <- function(knots, x, y) {
  x_new <- seq(0, 1, length.out = knots + 2)[2:(knots + 1)]
  knots_pos <- quantile(x, probs = x_new)
  bspline <- lm(y ~ bs(x, knots = knots_pos, degree = 15))
  predict(bspline, newdata = data.frame(x = x))
}

volatility <- spline_fit(15, vol$day, vol$std)

# Plot Volatility
ggplot(vol, aes(x = day)) +
  geom_point(aes(y = std, color = "Observed Volatility")) +
  geom_line(aes(y = volatility, color = "Spline Fit"), linewidth = 1) +
  scale_color_manual(
    values = c("Observed Volatility" = "blue", "Spline Fit" = "black")
```

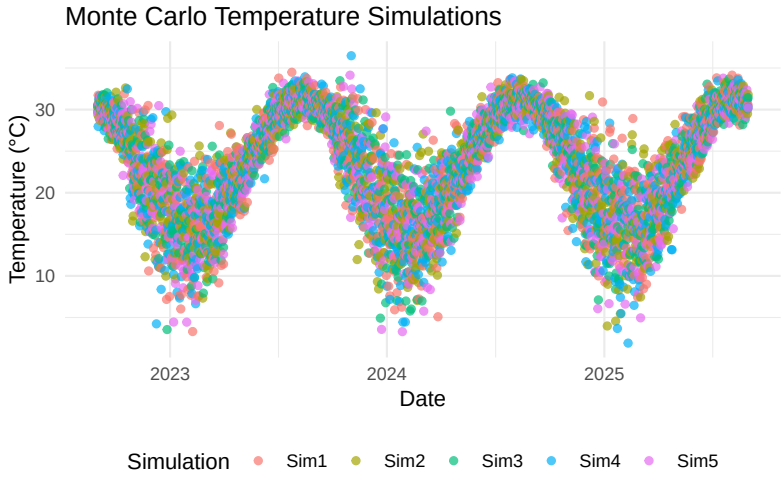
```
  ) +
  labs(
    title = "Temperature Volatility by Day of Year",
    y = "Std Dev (°C)",
    x = "Day of Year"
  ) +
  theme_minimal()
```




```
# 7. Monte Carlo Simulation Functions
euler_step <- function(row, kappa, M) {
  T_i <- ifelse(is.na(row$Tbar_shift), row$Tbar, row$Tbar_shift)
  T_det <- T_i + row$dTbar
  T_mrev <- kappa * (row$Tbar - T_i)
  sigma <- row$vol * rnorm(M)
  T_det + T_mrev + sigma
}

monte_carlo_temp <- function(trading_dates, Tbar_params, vol_model,
                             first_ord, M = 1, kappa = 0.2430516) {
  # Convert dates to numeric if needed
  if (inherits(trading_dates, "Date")) {
    trading_numeric <- as.numeric(trading_dates)
  } else {
```

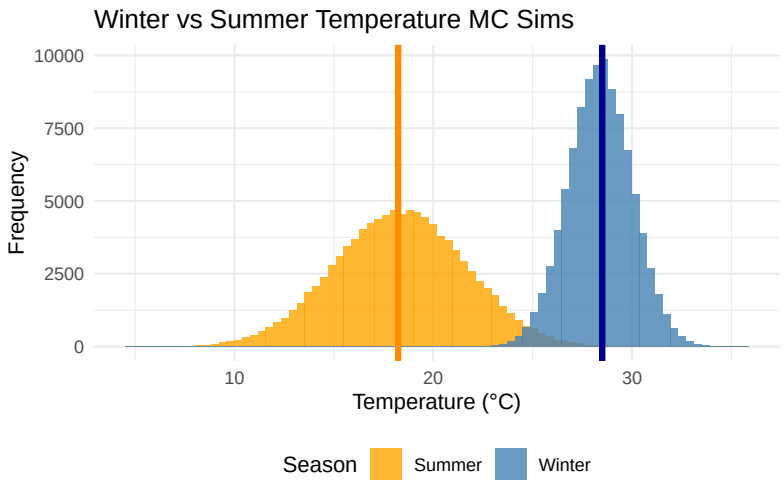
```
    trading_dates <- as.Date(trading_dates)
    trading_numeric <- as.numeric(trading_dates)
  }

  # Calculate Tbar and dTbar
  x_vals <- trading_numeric - first_ord
  Tbars <- T_model(x_vals, Tbar_params$a, Tbar_params$b,
                   Tbar_params$alpha, Tbar_params$theta)

  dTbars <- dT_model(x_vals, Tbar_params$a, Tbar_params$b,
                   Tbar_params$alpha, Tbar_params$theta)

  # Create simulation dataframe
  mc_temps <- data.frame(
    Date = trading_dates,
    Tbar = Tbars,
    dTbar = dTbars,
    day = yday(trading_dates),
    vol = vol_model[yday(trading_dates)] # Directly add volatility
  )

  # Add lagged Tbar
  mc_temps$Tbar_shift <- dplyr::lag(mc_temps$Tbar)

  # Run simulations - modified apply call
  simulations <- sapply(1:nrow(mc_temps), function(i) {
    row <- mc_temps[i, ]
    euler_step(row, kappa, M)
  })

  # Transpose and format results
  simulations <- t(simulations)
  colnames(simulations) <- paste0("Sim", 1:M)

  list(
    mc_temps = mc_temps,
    mc_sims = cbind(Date = trading_dates, as.data.frame(simulations))
  )
}
```

```r
# 8. Run Simulation
trading_dates <- seq(as.Date("2022-09-01"), as.Date("2025-08-31"), by = "day")
sim_results <- monte_carlo_temp(
  trading_dates,
  Tbar_params,
  volatility,
  first_ord,
  M = 5
)

# 9. Plot Simulation Results
sim_results$mc_sims %>%
  pivot_longer(-Date, names_to = "Simulation", values_to = "Temperature") %>%
  ggplot(aes(x = Date, y = Temperature, color = Simulation)) +
  geom_point(alpha = 0.7) +
  labs(title = "Monte Carlo Temperature Simulations", y = "Temperature (°C)") +
  theme_minimal() +
  theme(legend.position = "bottom")
```




```r
# Set number of simulations
no_sims <- 100000

# Define winter and summer dates (Southern Hemisphere)
trading_dates_winter <- as.Date("2025-10-01")
```

```r
trading_dates_summer <- as.Date("2025-04-01")

# Run simulations
sim_results_winter <- monte_carlo_temp(trading_dates_winter, Tbar_params, volatility, first_or
sim_results_summer <- monte_carlo_temp(trading_dates_summer, Tbar_params, volatility, first_or



# Extract results
mc_sims_winter <- sim_results_winter$mc_sims %>% select(-Date)
mc_sims_summer <- sim_results_summer$mc_sims %>% select(-Date)

Tbar_summer <- sim_results_summer$mc_temps$Tbar[1]
Tbar_winter <- sim_results_winter$mc_temps$Tbar[1]

# Create combined data frame for plotting
plot_data <- bind_rows(
  data.frame(Temperature = unlist(mc_sims_summer), Season = "Summer"),
  data.frame(Temperature = unlist(mc_sims_winter), Season = "Winter")
)

# Create the plot
ggplot(plot_data, aes(x = Temperature, fill = Season)) +
  geom_histogram(position = "identity", alpha = 0.8, bins = 80) +
  geom_vline(aes(xintercept = Tbar_winter), color = "darkblue",
             linewidth = 1.5, linetype = "solid") +
  geom_vline(aes(xintercept = Tbar_summer), color = "darkorange",
             linewidth = 1.5, linetype = "solid") +
  scale_fill_manual(values = c(Winter="steelblue", Summer="orange")) +
  labs(title = "Winter vs Summer Temperature MC Sims",
       x = "Temperature (°C)",
       y = "Frequency") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



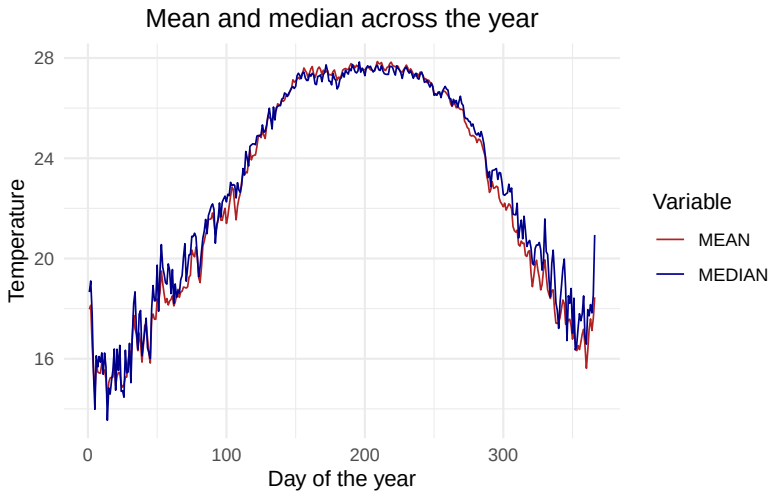
7 Sidequests

7.1 Sidequest: overlap of the dataset

One of the points that i would need for later is the standard deviation across the days of year, a pivot table is used for this step as with this I'm able to line up all the dataset month to month, by doing this im also able to analize seasonal patterns and look at the seasonality at a glance

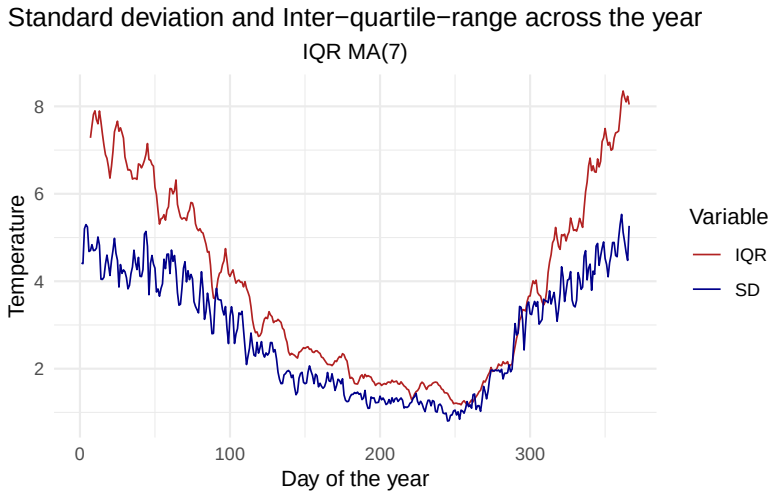
```
pivot_df <- DATASET %>%
  select(DOY, YEAR, T_AVG) %>%
  pivot_wider(names_from = YEAR, values_from = T_AVG)


MAX_pivot_df <- DATASET %>%
  select(DOY, YEAR, T_MAX) %>%
  pivot_wider(names_from = YEAR, values_from = T_MAX)

MIN_pivot_df <- DATASET %>%
  select(DOY, YEAR, T_MIN) %>%
  pivot_wider(names_from = YEAR, values_from = T_MIN)
```

```
MEAN <- apply(pivot_df[-1], 1, mean, na.rm=TRUE)
MEDIAN <- apply(pivot_df[-1], 1, median, na.rm=TRUE)
IQR <- apply(pivot_df[-1], 1, IQR, na.rm=TRUE)
SD <- apply(pivot_df[-1], 1, sd, na.rm=TRUE)

data.frame(MEAN = MEAN,
           MEDIAN = MEDIAN) %>%
  quickplot(title = "Mean and median across the year", xlab = "Day of the year", ylab = "Tempe
```

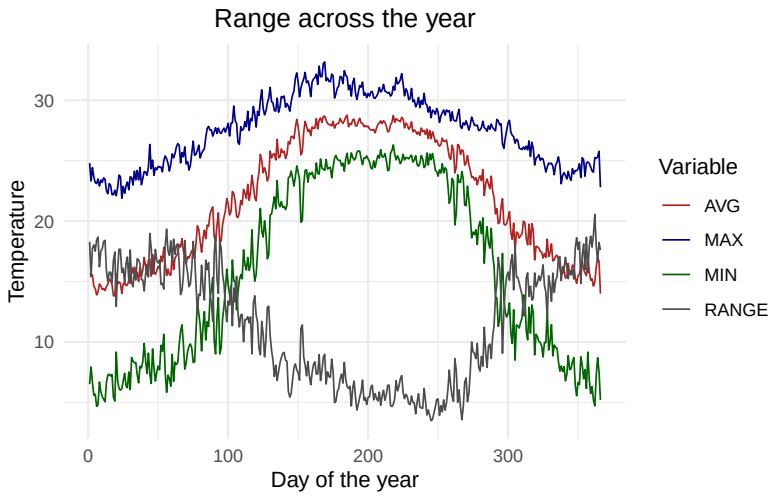



```
data.frame(IQR = SMA(IQR, n = 7),
           SD = SD) %>%
  quickplot(title = "Standard deviation and Inter-quartile-range across the year",
            subtitle = "IQR MA(7)", xlab = "Day of the year", ylab = "Temperature")
```

```
Warning: Removed 6 rows containing missing values or values outside the scale range
(`geom_line()`).
```



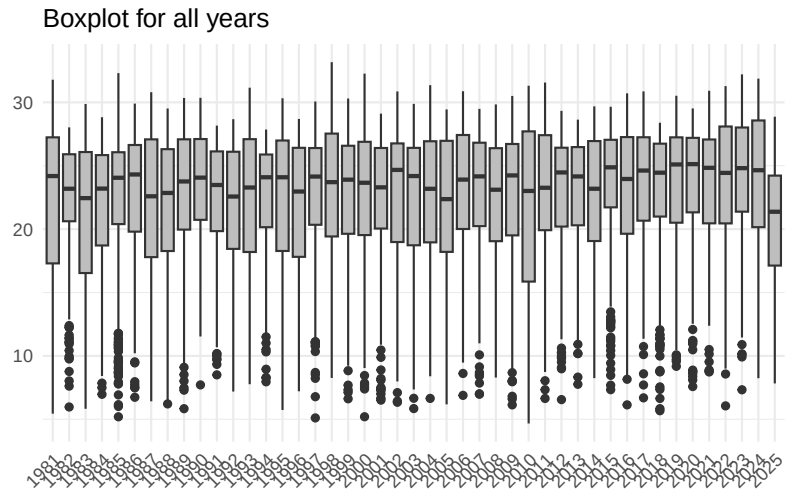

```
data.frame(MAX = apply(pivot_df[-1], 1, max, na.rm=TRUE),
           MIN = apply(pivot_df[-1], 1, min, na.rm=TRUE)) %>%
  mutate(AVG = (MAX + MIN)/2) %>%
  mutate(RANGE = MAX - MIN) %>%
  quickplot(title = "Range across the year", show_legend = T, xlab = "Day of the year", ylab =
```



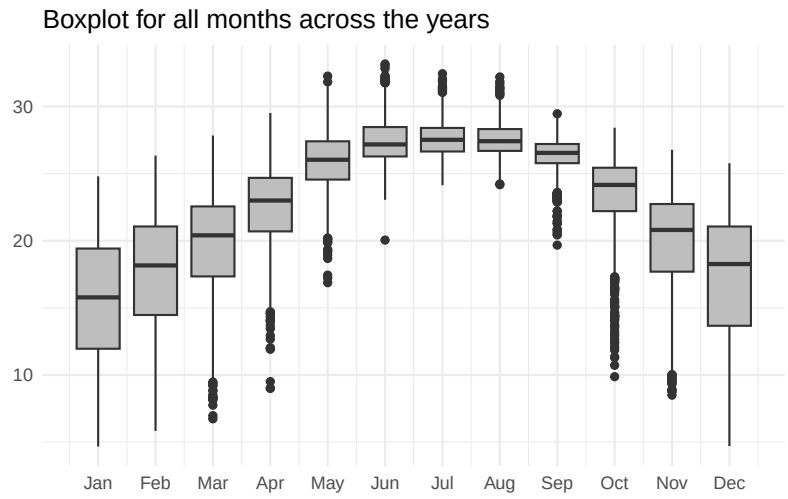

```
melt(pivot_df[-1], id.vars = NULL) %>%
  ggplot(aes(x = variable, y = value)) +
```

```
    geom_boxplot(fill = "gray") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = "Boxplot for all years", x = NULL, y = NULL)
```

```
Warning: Removed 271 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



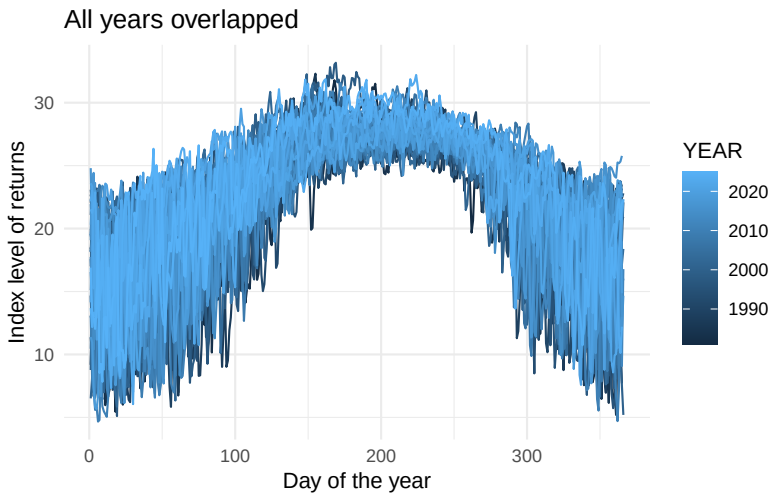

```
DATASET %>%
  select(Month, T_AVG) %>%
  group_by(Month) %>%
  ggplot(aes(x = Month, y = T_AVG, group = Month)) +
    geom_boxplot(fill = "gray") +
    scale_x_continuous(breaks = seq(1, 12, by = 1), labels = month.abb) +
    labs(title = "Boxplot for all months across the years", x = NULL, y = NULL)
```



7.1.1 How is this year compared to the rest

this dataset is 43 years long and there have been some really hot points and some really cold points over the years, so i wanted to look at my dataset and create a ribbon of all the max and mins, to see if this year has been so hot when looking at the averages, while this is in no way conclusive evidence it at least gives you a glimpse at how the climate is changing especially when looking at the middle of the year.

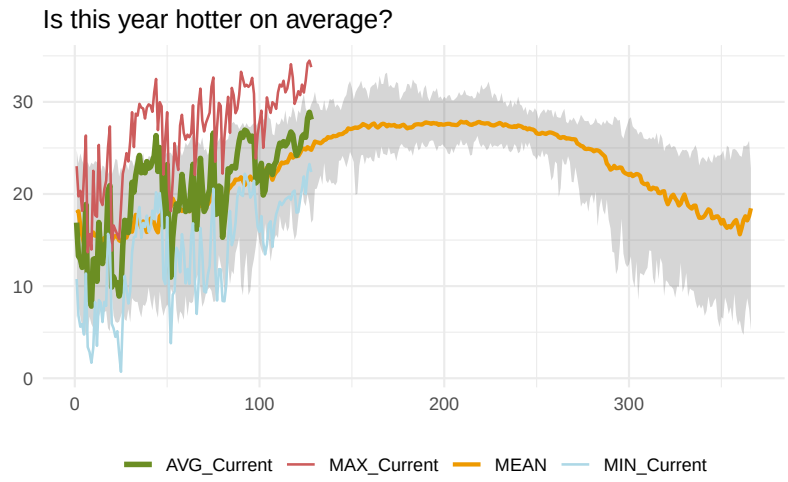
```
ggplot(DATASET, aes(x = DOY, y = T_AVG, group = YEAR, color = YEAR)) +
  geom_line() +
  labs(title = "All years overlapped", x = "Day of the year", y = "Index level of returns")
```




```
THISYEAR <- data.frame(
  c(DATASET[DATASET$YEAR == 2024, ]["T_MAX"]),
  c(DATASET[DATASET$YEAR == 2024, ]["T_AVG"]),
  c(DATASET[DATASET$YEAR == 2024, ]["T_MIN"])
)

cbind(pivot_df,
      "MIN" = apply(pivot_df[-1], 1, min, na.rm = TRUE),
      "MAX" = apply(pivot_df[-1], 1, max, na.rm = TRUE),
      "MEAN" = apply(pivot_df[-1], 1, mean, na.rm = TRUE)) %>%
  round(2) %>%
  ggplot(aes(x = DOY)) +
    geom_ribbon(aes(ymin = MIN, ymax = MAX), alpha = 0.2) +
    geom_line(aes(y = MEAN, color = "MEAN"), linewidth = 1) +
    geom_line(aes(y = pivot_df$`2025`, color = "AVG_Current"), linewidth = 1.3) +
    geom_line(aes(y = MAX_pivot_df$`2025`, color = "MAX_Current"), linewidth = 0.6, alpha = 1)
    geom_line(aes(y = MIN_pivot_df$`2025`, color = "MIN_Current"), linewidth = 0.6, alpha = 1)
  scale_color_manual(name = NULL, values = c(MEAN = "orange2", AVG_Current = "olivedrab",
                                             MIN_Current = "lightblue", MAX_Current = "indi
  labs(title = "Is this year hotter on average?", y = NULL, x = NULL)+
  theme(legend.position = "bottom")
```

```
Warning: Removed 238 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 238 rows containing missing values or values outside the scale range
```

```
(`geom_line()`).
Removed 238 rows containing missing values or values outside the scale range
(`geom_line()`).
```



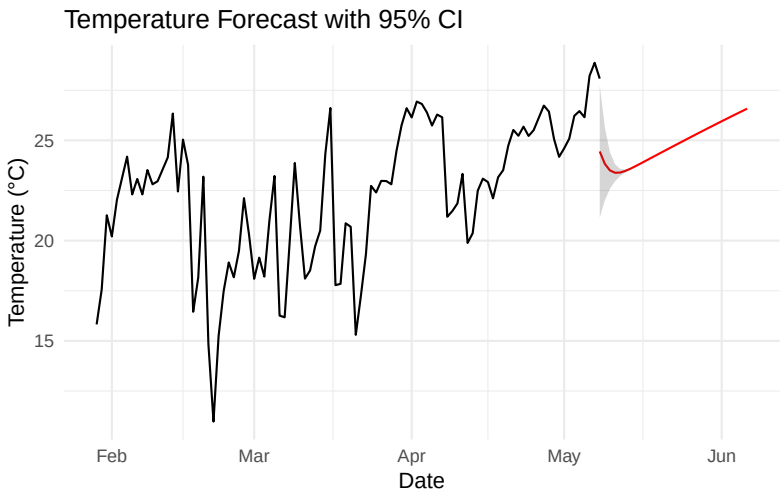
7.2 Sidequest: predicting temperatures in the future

Right now on 11-5-2025 i want to do an experiment and actually see if at least in some way i can predict the weather using this model that i've built right now i want to predict the temperature one month from now and

```
model_formula <- function(t, a, b, a1, b1) {
  omega <- 2 * pi / 365.25
  a + b * t + a1 * cos(omega * t) + b1 * sin(omega * t)
}

forecast_N <- 30
new_t <- max(temps$NUM_DAY) + forecast_N  # forecast_N days after last observation
predicted_T <- model_formula(new_t, params["a"], params["b"], params["alpha"], params["theta"])
cat("predicted temperature for", as.character(max(temps$DAY)+forecast_N), "=", predicted_T)
```

```
predicted temperature for 2025-06-07 = 26.82483
```

```r
arima_model <- Arima(temps$RESID, order = c(2, 0, 0), include.mean = FALSE)
future_residuals <- forecast(arima_model, h = forecast_N)  # forecast_N steps ahead

first_date <- min(temps$DAY)
future_dates <- seq(max(temps$DAY), by = "day", length.out = forecast_N)
future_t <- as.numeric(difftime(future_dates, first_date, units = "days"))

# Deterministic part
deterministic_part <- model_formula(future_t, params["a"], params["b"], params["alpha"], params

# Stochastic part (residuals)
stochastic_part <- future_residuals$mean

# Final prediction
future_T <- deterministic_part + stochastic_part

forecast_df <- data.frame(
  Date = future_dates,
  Temperature = future_T,
  Lower = future_T - 1.96 * future_residuals$mean,  # 95% CI
  Upper = future_T + 1.96 * future_residuals$mean
)

ggplot() +
  geom_line(data = tail(temps, 100), aes(x = DAY, y = T_AVG)) +
  geom_line(data = forecast_df, aes(x = Date, y = Temperature), color = "red") +
  geom_ribbon(data = forecast_df, aes(x = Date, ymin = Lower, ymax = Upper), alpha = 0.2) +
  labs(title = "Temperature Forecast with 95% CI", y = "Temperature (°C)", x = "Date")
```



The residuals represent short-term deviations from the deterministic trend/seasonality, becoming irrelevant after 15 days, which means that in the long run you will get a temperature entirely determined by the deterministic trend, over time, these deviations are expected to fade back to zero (the mean).

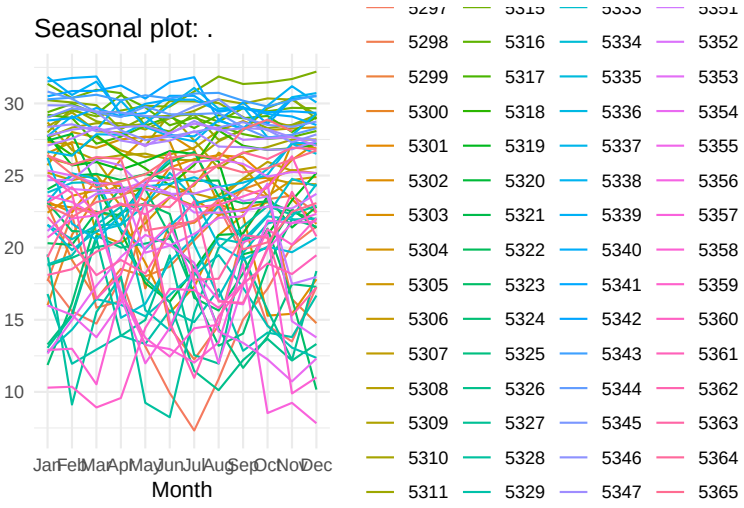
Forecast implications:

for forecasting purposes it means a couple of things: that in the short-term my Residuals adjust for recent "shocks" (e.g., a heatwave/cold snap), but in the long-term the forecast relies entirely on the deterministic component (trend + seasonality).

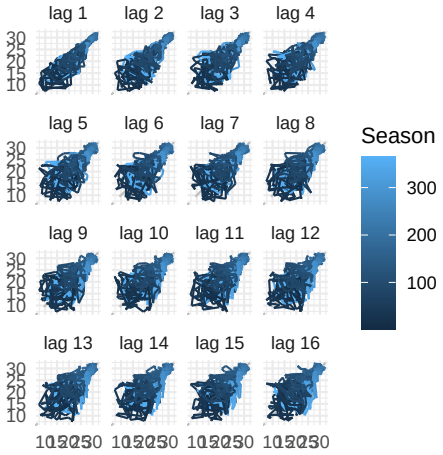
This model doesn't really account for Volatility clustering, like how temperature tends to have time-varying volatility (e.g., summer vs. winter variability). which is the part of the OU that we still need to model

7.3 forecast seasonal plots

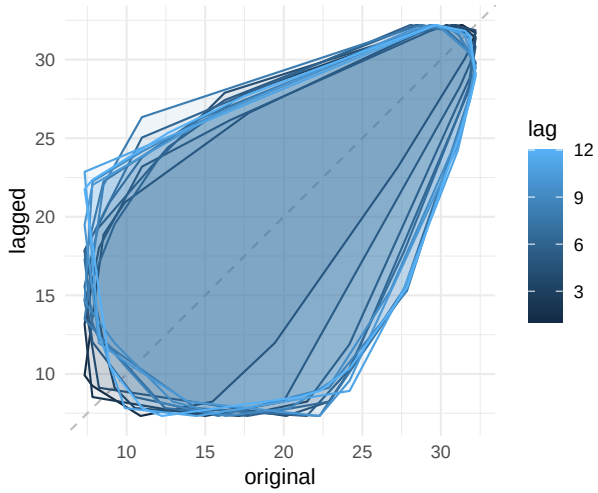
```
ts(data = temps$T_AVG, frequency = 12, start = temps$DAY[1]) %>% tail(365*2+ last(temps$DOY)) %
```




```
ts(data = temps$T_AVG, frequency = 365, start = temps$DAY[1]) %>% tail(365*2+ last(temps$DOY))
```

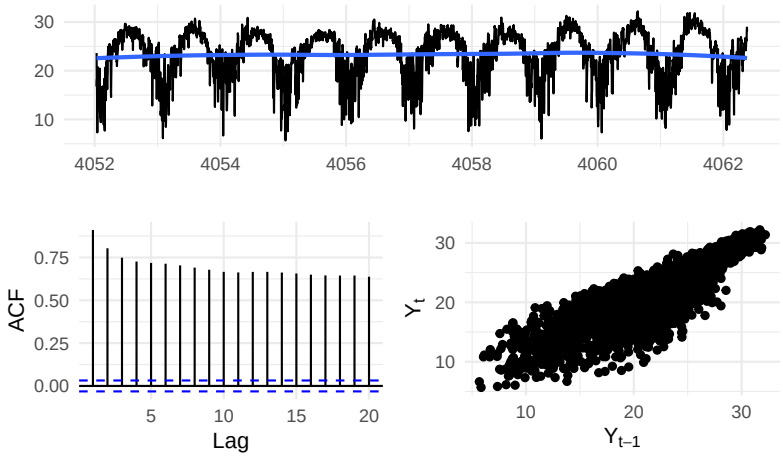



```
ts(data = temps$T_AVG, frequency = 365.25, start = temps$DAY[1]) %>% tail(365*2+ last(temps$DOY
```




```
ts(data = temps$T_AVG, frequency = 365, start = temps$DAY[1]) %>% tail(365*10+ last(temps$DOY)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

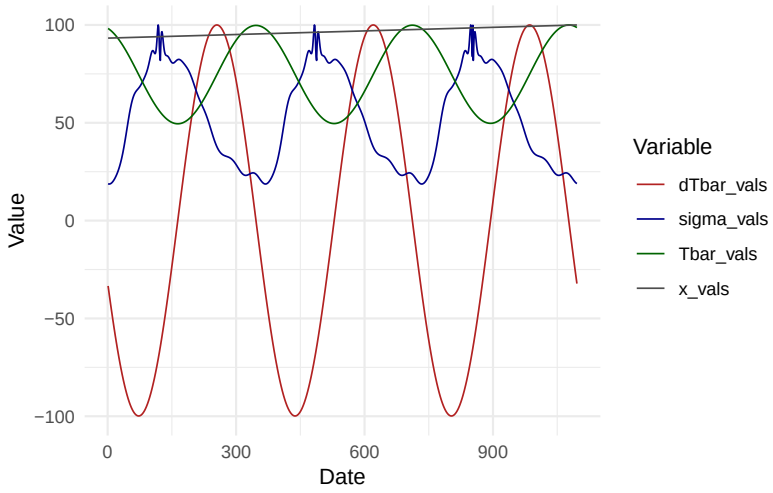



```
# Assuming trading_dates and sim_length already defined
x_vals <- as.numeric(trading_dates) - first_ord
Tbar_vals <- T_model(x_vals, a, b, alpha, theta)
dTbar_vals <- dT_model(x_vals, a, b, alpha, theta)
```

```
sigma_vals <- volatility[yday(trading_dates)]

data.frame(
  x_vals = x_vals,
  Tbar_vals = Tbar_vals,
  dTbar_vals = dTbar_vals,
  sigma_vals = sigma_vals
) %>% show_df()
```

DATE	x_vals	Tbar_vals	dTbar_vals	sigma_vals
1	15218	30.67273	-0.0450815	0.9961240
2	15219	30.62656	-0.0472611	0.9922418
3	15220	30.57822	-0.0494267	0.9902199
4	15221	30.52771	-0.0515776	0.9900876
5	15222	30.47507	-0.0537132	0.9918538
NA	NA	NA	NA	NA
1092	16309	30.96880	-0.0345569	1.0409072
1093	16310	30.93313	-0.0367942	1.0289168
1094	16311	30.89522	-0.0390206	1.0183175
1095	16312	30.85509	-0.0412354	1.0092453
1096	16313	30.81275	-0.0434380	1.0018160

```
data.frame(
  x_vals = x_vals,
  Tbar_vals = Tbar_vals,
  dTbar_vals = dTbar_vals,
  sigma_vals = sigma_vals
) %>% apply(2, normalize) %>% quickplot()
```


100
50
0
−50
−100
Value
0
300
600
900
Date
Variable
dTbar_vals
sigma_vals
Tbar_vals
x_vals