Capstone Project

Pablo C. Molinari

2025-07-01

My Capstone Project - The Hague Weather prediction

1. Introduction

1.1 Why I am doing this

For this capstone project, I will apply advanced machine learning techniques—specifically those suited for time series analysis—to predict future temperatures. Rather than relying on standard linear regression methods, I aim to explore more robust and context-aware models. In alignment with the instructor's guidance to avoid widely used data sets, I have chosen publicly available data from the Meteo site. Since the data set is time-dependent in nature, it necessitates methods that account for temporal structure and sequential patterns, going well beyond conventional regression approaches. This was very motivating to me as it meant that not only I will deliver the project but I will improve my complex model knowledge. On top of this the usage of such a complex model shows the originality of the approach.

1.2 Motivation for using this data set

Predicting temperature specifically comes with its own unique set of benefits. Here's why it matters:

- Energy management: Accurate temperature forecasts help households and businesses adjust heating and cooling systems, reducing energy costs and improving efficiency.
- Health and well-being: Knowing in advance about extreme temperatures can help people prepare—hydrating during heatwaves, dressing warmly during cold snaps, and protecting vulnerable groups.
- Event planning: Organizers can decide if it's feasible to hold outdoor events or whether to make contingency plans.
- Retail and inventory: Stores use temperature predictions to manage stock—for example, putting out more cold drinks when a heatwave is coming.
- Infrastructure resilience: Maintenance teams can anticipate how temperature changes might affect roads, railways, and other critical systems.

1.3 Executive Summary

This project focuses on developing a robust machine learning model to forecast future temperatures using publicly available meteorological data from the Meteo site. The dataset comprises time-stamped records of atmospheric variables collected from weather stations in the Netherlands, with a particular focus on The Hague. Key variables include daily maximum and minimum temperatures, average temperature, humidity, wind speed, air pressure, and precipitation levels.

The primary goal of this project is to accurately predict future temperature trends by leveraging the temporal nature of the data. Traditional linear regression methods are insufficient for capturing time-dependent patterns; therefore, this project employs advanced time series forecasting techniques, including data pre-processing with rolling averages, exploratory data analysis, and model development using algorithms such as ARIMA, Prophet and NeNetworks.

The key steps performed include: - **Data Acquisition**: Collecting historical temperature data from the Meteo site. - **Data Cleaning & Preprocessing**: Handling missing values, formatting time indices, and engineering relevant features. - **Exploratory Analysis**: Identifying trends, seasonality, and anomalies to guide model selection. - **Model Development & Evaluation**: Implementing and comparing multiple forecasting techniques to select the most accurate model. - **Prediction & Visualization**: Generating future temperature forecasts with clear visual representations.

By utilizing machine learning tailored to time series data, this project aims to demonstrate the practical potential of predictive modeling for environmental data, offering insights valuable to energy management, public planning, and daily life in The Hague.

2. Methods and Analysis

To predict future temperature trends in The Hague using Meteo data, I followed a systematic process comprising data extraction, data cleaning, exploration, and the application of two distinct modeling approaches. Each phase was designed to ensure data quality, uncover key patterns, and build accurate, time-aware models.

2.1 **Data Extraction**

In order to obtained the data I used https://open-meteo.com/ (https://open-meteo.com/) Open-Meteo is an open-source weather API and offers free access for non-commercial use. It has the added advantage that you can specify both latitude and longitude of any point in the globe and it will indicate the historical weather with all the parameters selected

I selected the city where I live (The Hague in the Netherlands) as location because i am familiar with it and I could spot quickly any big discrepancy in the data.

The selection of parameters and location (latitude and Longitude) can be done using the following URL that it is pasted here and can be re-used to extract the same:

https://open-meteo.com/en/docs/historical-weather-api?hourly=temperature_2m,relative_humidity_2m,rain,snowfall,precipitation,cloud_cover&start_date=2015-01-01&end_date=2024-12-31&longitude=4.3&latitude=52.11 (https://open-meteo.com/en/docs/historical-weather-api?hourly=temperature_2m,relative_humidity_2m,rain,snowfall,precipitation,cloud_cover&start_date=2015-01-01&end_date=2024-12-31&longitude=4.3&latitude=52.11)

However, in order to facilitate the work of the reviewers of my capstone, I downloaded the information into my github account.

Here is how to read the data and display first 5 records

```
# Read the file
data <- read.csv("https://raw.githubusercontent.com/pcmolinari/Edx_Capstone2/main/data/open-meteo-52.13N4.31E7m.csv", sep = ",", header = TRUE, skip=3)

# Display first 5 records to inspect the data read, headers
knitr::kable(head(data, 5), caption = "First 5 Records of My Data")
```

First 5 Records of My Data

time	temperature_2m… C.	relative_humidity_2m….	rain..mm.	snowfall..cm.	precipitation..mm.	cloud_cover….
2015-01-01T00:00	3.7	86	0	0	0	34
2015-01-01T01:00	3.6	85	0	0	0	12

time	temperature_2m… C.	relative_humidity_2m….	rain..mm.	snowfall..cm.	precipitation..mm.	cloud_cover….
2015-01-01T02:00	3.5	84	0	0	0	7
2015-01-01T03:00	3.4	84	0	0	0	7
2015-01-01T04:00	3.1	83	0	0	0	24

2.2 **Implement all the necessary packages**

```
quiet_load <- function(pkg) {
  if (!suppressMessages(require(pkg, character.only = TRUE))) {
    suppressMessages(suppressWarnings(
      install.packages(pkg, repos = "http://cran.us.r-project.org")
    ))
    suppressMessages(require(pkg, character.only = TRUE))
  }
}

# List of packages
packages <- c("tidyverse", "caret", "data.table", "lubridate", "stringr",
              "ggplot2", "knitr", "kableExtra", "scales", "dplyr", "lubridate",
              "forecast", "reshape2", "RColorBrewer", "Metrics", "glue", "prophet",
              "xgboost", "scales", "DALEX")

# Silently install and load
invisible(lapply(packages, quiet_load))
```

2.3 **Data Cleaning**

The raw dataset obtained from the Meteo site included daily meteorological records with variables that I selected from the site (that are visible in the URL stated above) such as date, temperature, humidity, snowfall, cloud cover and precipitation. The following cleaning steps were applied:

- **Skip the first 3 records** that only contain file information and difficult the data formatting. That was done in the step above
- **Rename the columns** the default was not user friendly

```
# Rename the columns
colnames(data) <- c("time", "temperature", "humidity", "rain_mm", "snow_fall", "precipitation_mm", "clou
d_cover")
knitr::kable(head(data, 5), caption = "First 5 Records of My Data")
```

First 5 Records of My Data

time	temperature	humidity	rain_mm	snow_fall	precipitation_mm	cloud_cover
2015-01-01T00:00	3.7	86	0	0	0	34
2015-01-01T01:00	3.6	85	0	0	0	12

time	temperature	humidity	rain_mm	snow_fall	precipitation_mm	cloud_cover
2015-01-01T02:00	3.5	84	0	0	0	7
2015-01-01T03:00	3.4	84	0	0	0	7
2015-01-01T04:00	3.1	83	0	0	0	24

- **Missing values** were identified by counting NAN or black records

```
# Summarize NA and blank string counts for each column
missing_summary <- data.frame(
  Column = names(data),
  NA_Count = sapply(data, function(col) sum(is.na(col))),
  Blank_Count = sapply(data, function(col) sum(col == "", na.rm = TRUE))
)

# Display the result
kable(missing_summary, caption = "Missing and Blank Value Check")
```

Missing and Blank Value Check

	Column	NA_Count	Blank_Count
time	time	0	0
temperature	temperature	0	0
humidity	humidity	0	0
rain_mm	rain_mm	0	0
snow_fall	snow_fall	0	0
precipitation_mm	precipitation_mm	0	0
cloud_cover	cloud_cover	0	0

- **Datetime formatting** date time was standardized, ensuring the index reflected a continuous daily time series. And also i created 2 additional columns to make it easier to manage

```
## to enrich dinamics some transofrmations
# I make the time column in POSIXct format
data$time <- ymd_hm(data$time)

# Add 'date' and 'clock_time' columns
data <- data %>%
  mutate(
    date = as.Date(time),              # just the date part
    clock_time = format(time, "%H:%M") # formatted time string, e.g. "14:30"
  )
knitr::kable(head(data, 5), caption = "First 5 Records of My Data")
```

First 5 Records of My Data

time	temperature	humidity	rain_mm	snow_fall	precipitation_mm	cloud_cover	date	clock_time

time	temperature	humidity	rain_mm	snow_fall	precipitation_mm	cloud_cover	date	clock_time
2015-01-01 00:00:00	3.7	86	0	0	0	34	2015-01-01	00:00
2015-01-01 01:00:00	3.6	85	0	0	0	12	2015-01-01	01:00
2015-01-01 02:00:00	3.5	84	0	0	0	7	2015-01-01	02:00
2015-01-01 03:00:00	3.4	84	0	0	0	7	2015-01-01	03:00
2015-01-01 04:00:00	3.1	83	0	0	0	24	2015-01-01	04:00

2.4 **Exploratory Data Analysis & Visualization**

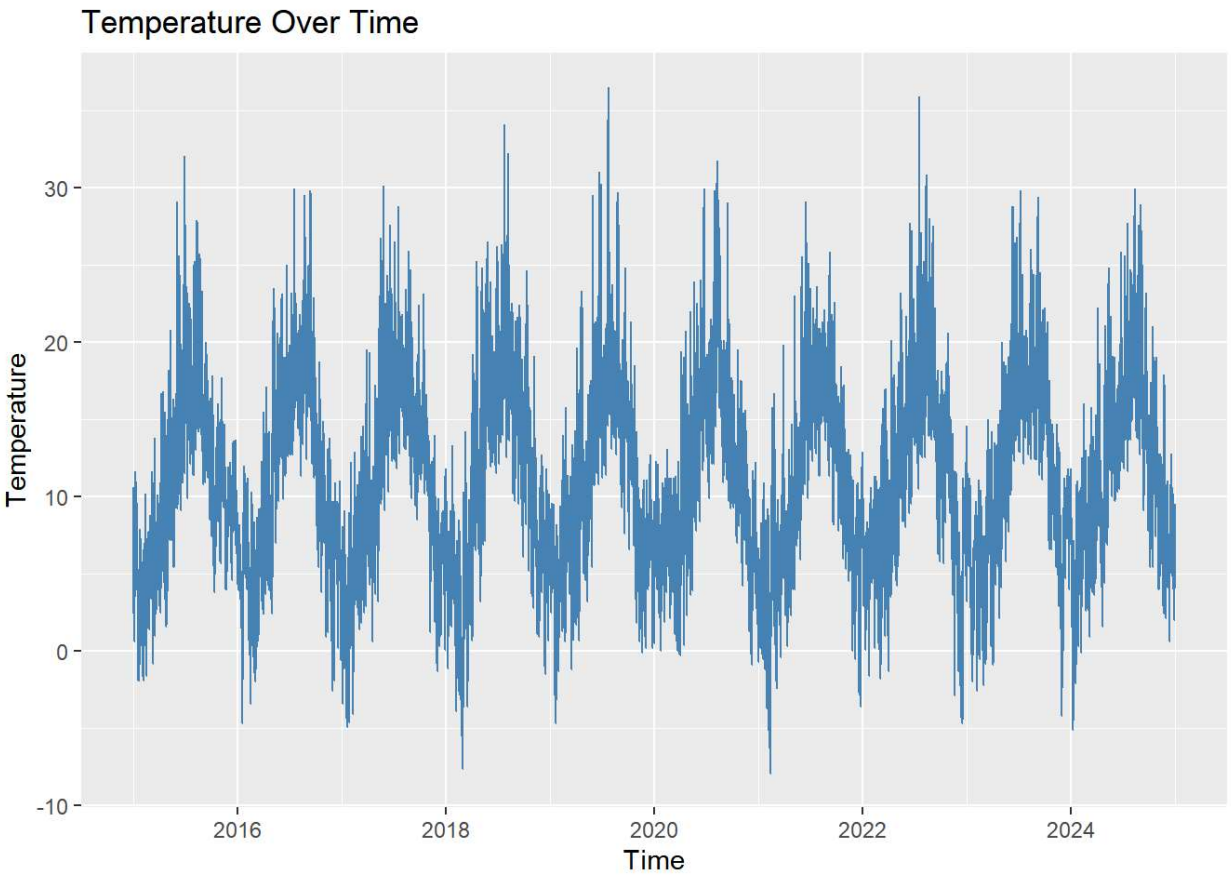
To understand the dataset's structure and trends I did the following data wrangling:

- **Data summarization** were used to understand the metadata

```
# Create summary
summary_df <- summary(data)
```

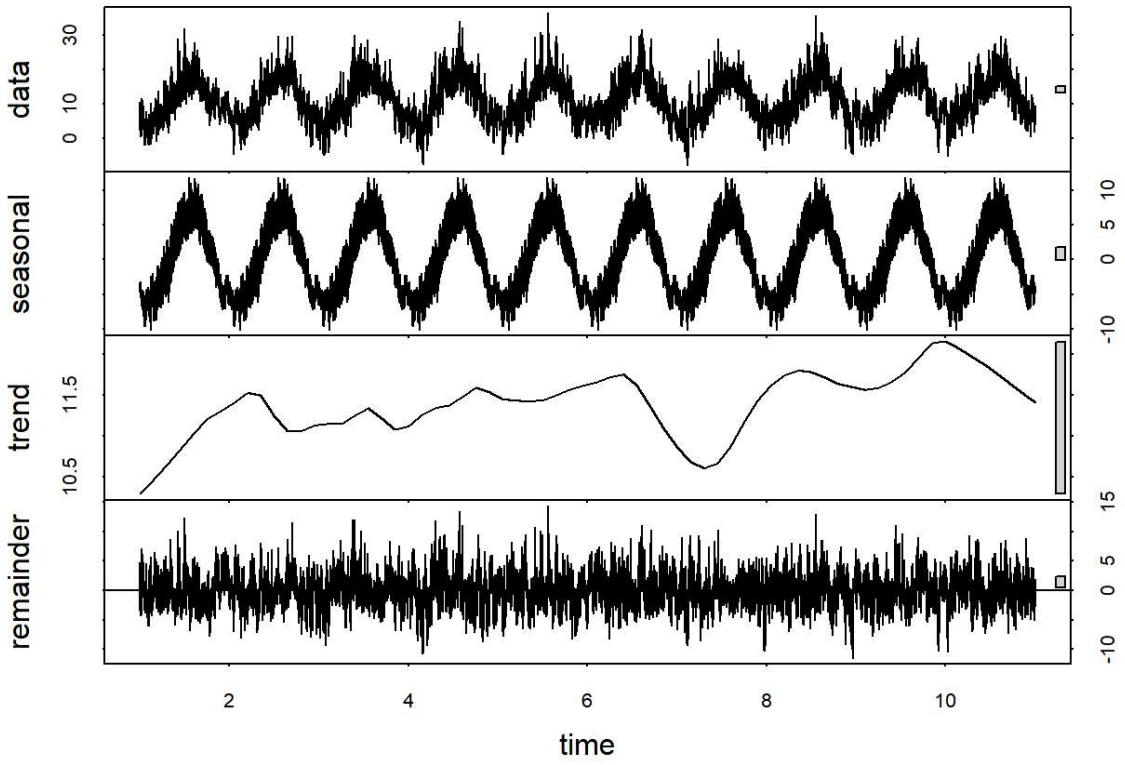
- **Line plots** were used to visualize temperature patterns over time and detect seasonality.

```
### time series line plot
library(ggplot2)

ggplot(data, aes(x = time, y = temperature)) +
  geom_line(color = "steelblue") +
  labs(title = "Temperature Over Time", x = "Time", y = "Temperature")
```



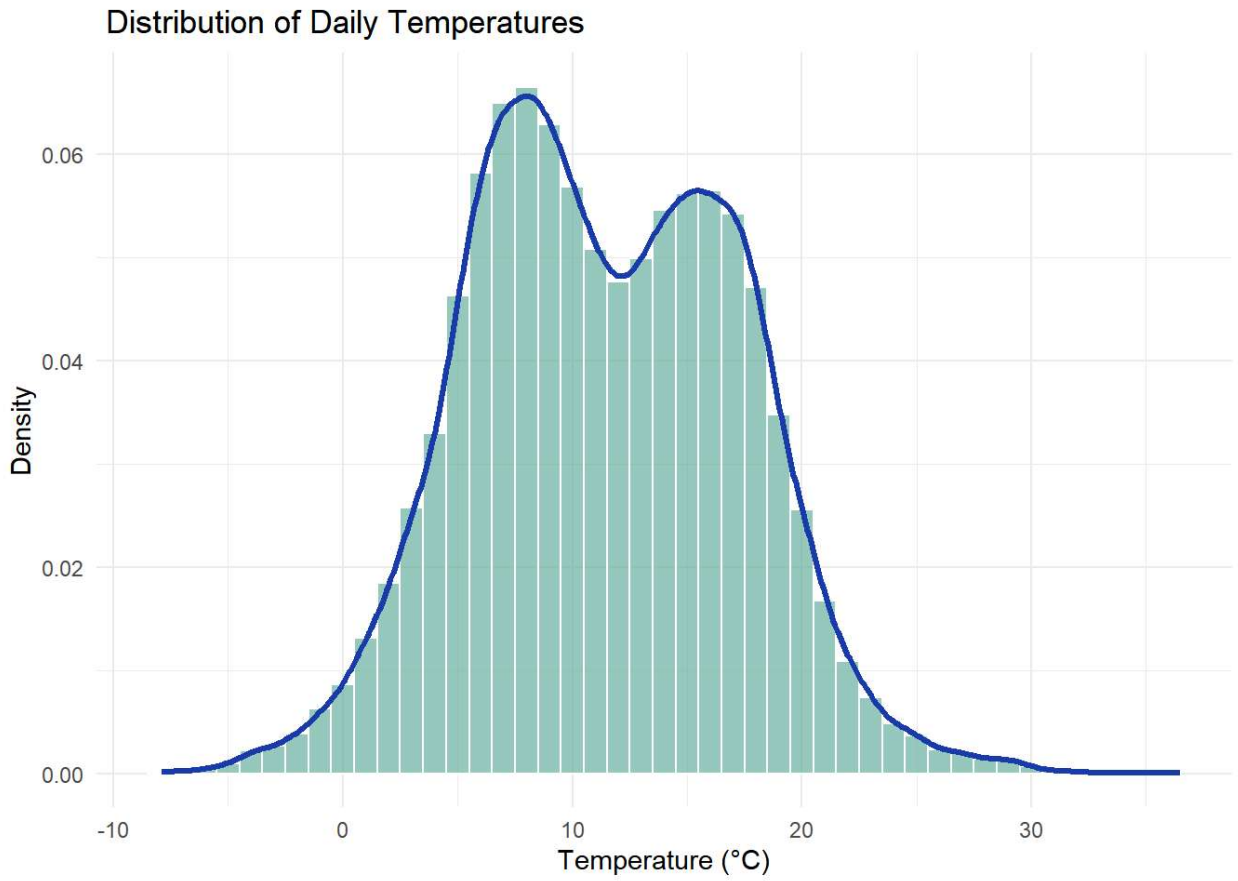
- **Decomposition analysis** (seasonal-trend decomposition using LOESS) was performed to isolate trend, seasonal, and residual components of the temperature signal.

```
#Decomposition Plot
#Breaks the time series into trend, seasonal, and residual components:
library(forecast)

temp_ts <- ts(data$temperature, frequency = 24 * 365)  # hourly data, yearly seasonality
decomp <- stl(temp_ts, s.window = "periodic")
plot(decomp)
```



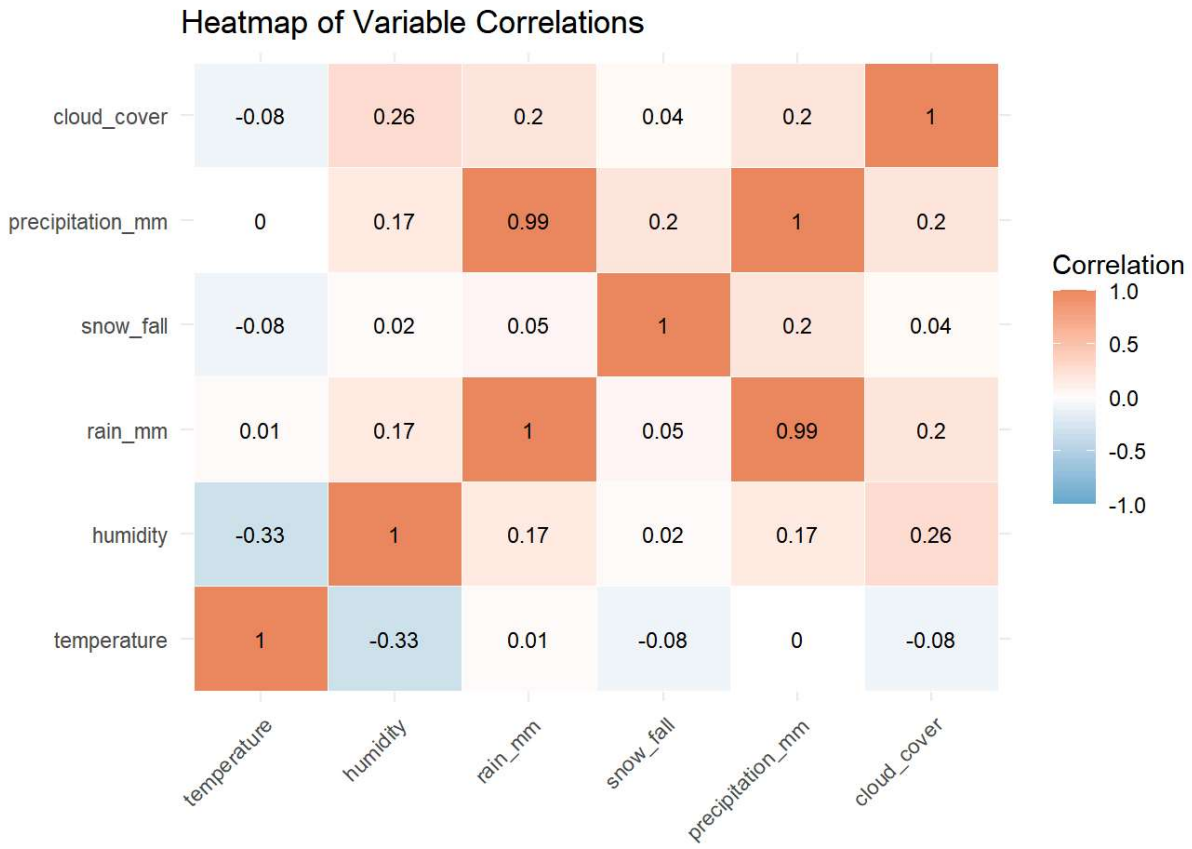
- **Histograms and KDE plots** provided insight into the distribution of daily temperatures.

```
# Load necessary libraries
library(ggplot2)
library(dplyr)

#Histogram
ggplot(data, aes(x = temperature)) +
  geom_histogram(aes(y = ..density..),
                 binwidth = 1,
                 fill = "#69b3a2",
                 color = "white",
                 alpha = 0.7) +
  geom_density(color = "#1c3faa", size = 1.2) +
  labs(
    title = " Distribution of Daily Temperatures",
    x = "Temperature (°C)",
    y = "Density"
  ) +
  theme_minimal()
```



- **Heatmaps** helped reveal correlations between temperature and other variables like humidity and wind speed.

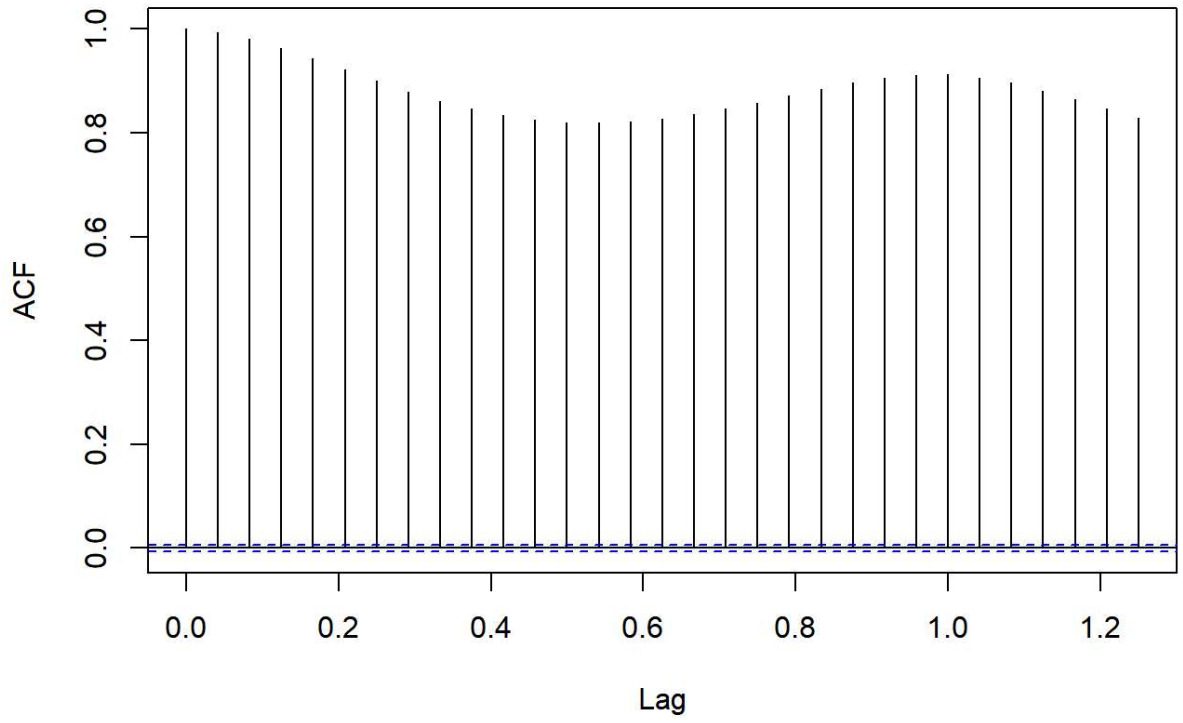
```
# Select only numeric columns for correlation
numeric_data <- data %>%
  select(where(is.numeric))

# Compute correlation matrix
cor_matrix <- round(cor(numeric_data, use = "complete.obs"), 2)

# Convert to long format for ggplot
melted_cor <- melt(cor_matrix)

# Plot heatmap
ggplot(data = melted_cor, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "#67a9cf", mid = "white", high = "#ef8a62",
                       midpoint = 0, limit = c(-1, 1), space = "Lab",
                       name = "Correlation") +
  geom_text(aes(label = value), color = "black", size = 3) +
  theme_minimal() +
  labs(title = "Heatmap of Variable Correlations",
       x = "", y = "") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
```



- **Autocorrelation plots** Significant spikes at lag 1, 2, 3… suggest temperature today is correlated with previous hours or days.

This justifies autoregressive models like ARIMA, ARIMAX, etc

```
# Convert time column to proper DateTime if not already
data$time <- as.POSIXct(data$time)


# We'll use the full time series here (adjust if aggregating):
ts_temp <- ts(data$temperature, frequency = 24)  # 24 if hourly, 1 if daily

# Plot autocorrelation
acf(ts_temp, lag.max = 30, main = "📈 ACF: Autocorrelation of Temperature")
```



The ACF (Autocorrelation Function) plot reveals significant autocorrelation at all the lags, as each spike (vertical line) extends well beyond the blue significance bounds. This strongly suggests that the time series has non-random structure and memory, meaning past values are highly predictive of future values.

Key insights included: - A clear **seasonal cycle** with predictable warm and cold periods. - Strong **autocorrelation** in temperatures across days, indicating suitability for autoregressive methods. - Moderate influence of **humidity** on daily temperature variability. Not so much on the other variables used

3. **Modeling Approaches**

To effectively capture the temporal dynamics inherent in the dataset, I implemented a two-phase modeling strategy that progressively increased in complexity and, whenever possible, interpretability.

3.1 **Univariate Time Series Modeling**

In the initial phase, I focused solely on the core temporal components—**date**, **clock_time**, and **temperature**—to assess the predictive capacity of the temperature variable in isolation. By combining the date and time columns into a single datetime feature, I constructed a chronological time index suitable for time series modeling. Using this configuration, I applied ARIMA that is well-suited to sequence data and capable of learning temporal dependencies and seasonal patterns. This baseline model served as a foundational benchmark, capturing purely time-driven trends in temperature without influence from external weather factors.

3.2 **Multivariate Time Series Modeling**

In the second phase, I expanded the feature space by incorporating additional meteorological variables: **humidity**, **cloud_cover**, **precipitation_mm**, **rain_mm**, and **snow_fall**. To determine the relevance of these features, I performed correlation analysis and mutual information scoring with respect to the temperature variable (see above correlation graphs).

Variables that exhibited strong relationships were selected to be included in enhanced models. These multivariate models were designed to capture both **temporal** and **contextual** patterns—enabling the network to learn not just the passage of time, but also how variations in environmental conditions influence temperature dynamics.

By comparing the performance metrics—primarily **Root Mean Square Error (RMSE)**—across both modeling stages, I was able to quantitatively assess the value added by incorporating additional input features.

Here is the summary of the models used, their pros and cons and how the handle seasonality and exogenous variables. The table is ranked as per usual literature and my interpreation. However, my model showed a slighly different result:

Rank	Model	Type	Seasonality	Exogenous Variables	Pros	Cons
1	Neural Network	Deep Learning	Yes	Yes	Captures nonlinear patterns and long-term dependencies	Requires a lot of data, black box, harder to tune
2	XGBoost	Machine Learning	No*	Yes	High accuracy, handles interactions and nonlinearities	Needs feature engineering, not time-aware by default
3	ARIMAX	Statistical	No*	Yes	Models autocorrelation and external drivers	Complex setup, assumes linearity
4	Prophet	Hybrid (decomposition)	Yes	Yes	Easy to use, flexible with trends and seasonality	Less effective on highly erratic or nonlinear data
5	ARIMA	Statistical	No*	No	Strong for autocorrelated univariate series	Requires stationarity, no exogenous inputs
6	ETS	Statistical	Yes	No	Fast and interpretable for trend + seasonality	Limited by assumption of stable structure
7	Linear Regression	Statistical / ML	No	Yes	Transparent and explainable	Misses temporal structure and autocorrelation
8	Seasonal Naive	Baseline	Yes	No	Simple benchmark for seasonal series	Ignores trends and external factors
9	Naive	Baseline	No	No	Simplest benchmark, no assumptions	Doesn’t model seasonality, trends, or anything else

*Note: ARIMA and XGBoost can model seasonality with additional configuration, but don’t handle it natively.

3.3 **Split of the sets of data (train and test)**

In order to start now the modelling I need to split the data

Split data into train and test set based on year

- The data I have is from several years.
- I will assess the model by trying to predict the 2024 data (my test set of data)
- Instead of doing a random split like in normal models, given the characteristics of this (time series), I will split using 2024 as test.

```
train <- data %>% filter(year(date) != 2024)

test  <- data %>% filter(year(date) == 2024)
## to enrich dinamics some transofrmations

train <- train %>%
  mutate(hour = hour(hm(clock_time)),
         day = yday(date))

test <- test %>%
  mutate(hour = hour(hm(clock_time)),
         day = yday(date))

h <- nrow(test)
actual <- test$temperature
```

3.4. **Run every model and see the RSME part of evaluation**

3.4.a —- Naive Forecast —-

```
naive_model <- naive(ts(train$temperature), h = h)
pred_naive <- as.numeric(naive_model$mean)
rmse_naive <- Metrics::rmse(actual, pred_naive)

cat(glue("RMSE - Naive Forecast: {round(rmse_naive, 2)}\n\n"))
```

```
## RMSE - Naive Forecast: 6.7
```

3.4.b — Seasonal Naive —-

```
snaive_model <- snaive(ts(train$temperature, frequency = 24), h = h)
pred_snaive <- as.numeric(snaive_model$mean)
rmse_snaive <- Metrics::rmse(actual, pred_snaive)

cat(glue("RMSE - Seasonal Naive: {round(rmse_snaive, 2)}\n\n"))
```

```
## RMSE - Seasonal Naive: 6.39
```

3.4.c —- ETS Model —-

```
ets_model <- ets(ts(train$temperature, frequency = 24))
pred_ets <- forecast(ets_model, h = h)$mean
rmse_ets <- Metrics::rmse(actual, as.numeric(pred_ets))

cat(glue("RMSE - ETS Model: {round(rmse_ets, 2)}\n\n"))
```

```
## RMSE - ETS Model: 6.72
```

3.4.d —- ARIMA Model —-

```
arima_model <- auto.arima(train$temperature)
pred_arima <- forecast(arima_model, h = h)$mean
rmse_arima <- Metrics::rmse(actual, as.numeric(pred_arima))

cat(glue("RMSE - ARIMA Model: {round(rmse_arima, 2)}\n\n"))
```

```
## RMSE - ARIMA Model: 6.43
```

3.4.e —- Arimax automatic —-

```
xreg_train <- as.matrix(train[, c("humidity", "cloud_cover")])
xreg_test <- as.matrix(test[, c("humidity", "cloud_cover")])
arimax_model <- auto.arima(train$temperature, xreg = xreg_train)
pred_arimax <- forecast(arimax_model, xreg = xreg_test)$mean
rmse_arimax <- Metrics::rmse(actual, as.numeric(pred_arimax))

cat(glue("RMSE - ARIMAX Model Automatic: {round(rmse_arimax, 2)}\n\n"))
```

```
## RMSE - ARIMAX Model Automatic: 6.5
```

3.4.f —- Arimax manually modified. —-

I will fit an ARIMAX model with 5 autoregressive (AR) and 5 moving average (MA) terms explicitly (i.e. #ARIMA(5,0,5)) and include external regressors

```
# Define the order: (p = 5, d = 0, q = 5)
arimax_model <- Arima(train$temperature, order = c(5, 0, 5), xreg = xreg_train)
# Forecast using future xreg values
pred_arimax <- forecast(arimax_model, xreg = xreg_test)$mean
# Calculate RMSE
rmse_arimax <- Metrics::rmse(test$temperature, as.numeric(pred_arimax))

cat(glue("🔁 RMSE - ARIMAX Model Manual: {round(rmse_arimax, 2)}\n\n"))
```

```
## 🔁 RMSE - ARIMAX Model Manual: 5.32
```

3.4.g —- Prophet —-

I will use Prophet a powerful time series forecasting library developed by Meta (formerly Facebook). It's great for capturing seasonality, holidays, and trend shifts with minimal tuning.

```
# Rename required columns
df_prophet <- train %>%
  mutate(ds = as.POSIXct(paste(date, clock_time)),  # combine date + clock_time
         y = temperature) %>%
  select(ds, y)

model <- prophet(df_prophet)
future <- test %>%
  mutate(ds = as.POSIXct(paste(date, clock_time))) %>%
  select(ds)
## generate forecast
forecast <- predict(model, future)
predicted_prophet <- forecast$yhat
### RMSE calculation
rmse_prophet <- rmse(test$temperature, predicted_prophet)
print(rmse_prophet)
```

```
## [1] 3.191173
```

```
cat(glue("RMSE - Prophet: {round(rmse_prophet, 2)}\n\n"))
```

```
## RMSE - Prophet: 3.19
```

3.4.h XGBOOST

```
# Prepare data
X_train <- as.matrix(train[, c("hour", "day", "humidity", "cloud_cover")])
X_test  <- as.matrix(test[,  c("hour", "day", "humidity", "cloud_cover")])
y_train <- train$temperature
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest  <- xgb.DMatrix(data = X_test)
xgb_model <- xgboost(data = dtrain, nrounds = 100, objective = "reg:squarederror", verbose = 0)
xgb_pred <- predict(xgb_model, newdata = dtest)
rmse_xgb <- Metrics::rmse(test$temperature, xgb_pred)

cat(glue("RMSE - XGBoost : {round(rmse_xgb, 2)}\n\n"))
```

```
## RMSE - XGBoost : 3.17
```

Neural Networks

```
# Step 1: Prepare features
# Choose top correlated features (e.g., humidity and cloud_cover)
xreg_train <- scale(train[, c("humidity", "cloud_cover")])
xreg_test  <- scale(test[, c("humidity", "cloud_cover")],
                    center = attr(xreg_train, "scaled:center"),
                    scale  = attr(xreg_train, "scaled:scale"))

# Step 2: Build Neural Network
nnet_model <- nnetar(
  y = train$temperature,
  xreg = xreg_train,
  p = 12,       # use 12 lags (~half-day if hourly data)
  size = 3,     # 3 hidden nodes to avoid overfitting
  repeats = 5   # increase stability by averaging multiple runs
)

# Step 3: Forecast
forecast_nnet <- forecast(nnet_model, xreg = xreg_test, h = nrow(test))
# Step 4: Evaluate
rmse_nnet <- rmse(test$temperature, as.numeric(forecast_nnet$mean))
cat(glue("RMSE - Neural Network : {round(rmse_nnet, 2)}\n\n"))
```

```
## RMSE - Neural Network : 6.92
```

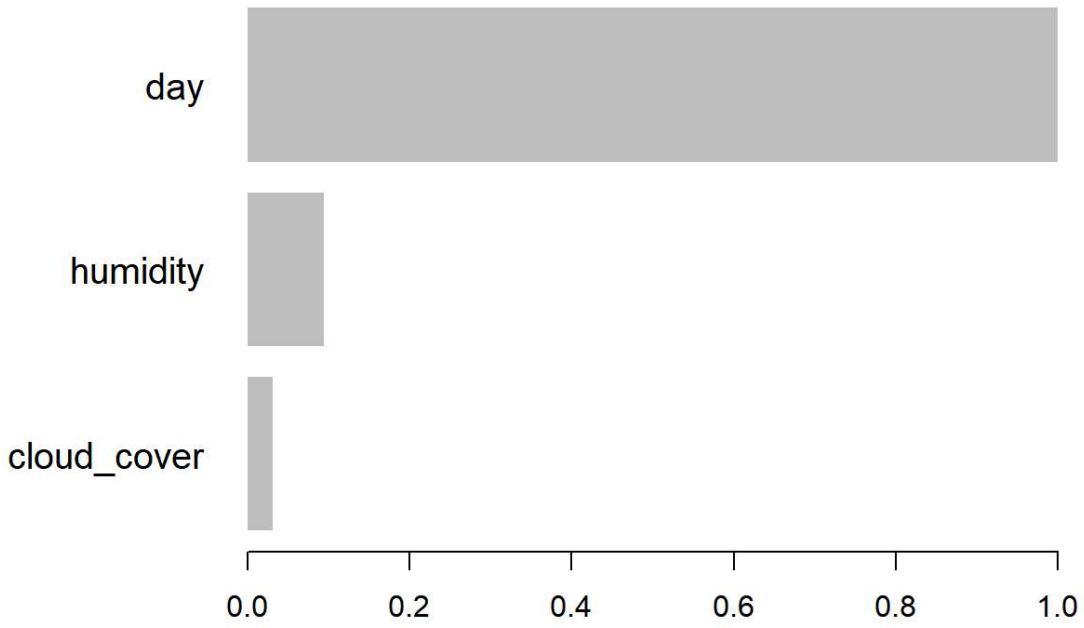
3.5 **Peek inside the best model***

I conclude the best model is XGBoost therefore I want to see the contents of the model so I can try to interpret it and explain to others that will use it.

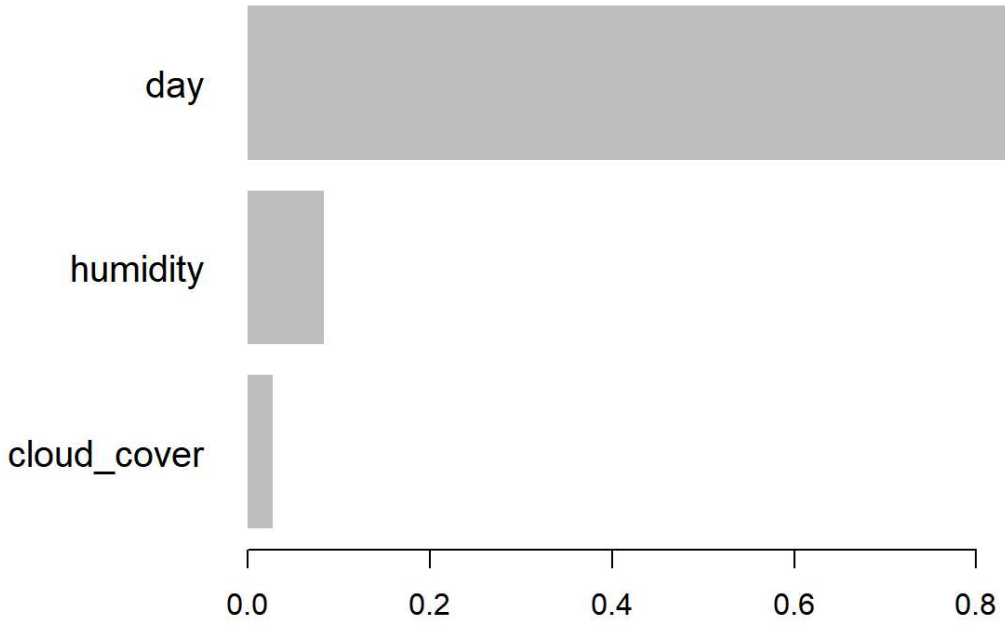
```
##### View feature imporantce of XG BOOST
importance_matrix <- xgb.importance(model = xgb_model)
print(importance_matrix)
```

```
##        Feature       Gain     Cover Frequency
##         <char>      <num>     <num>     <num>
## 1:         day 0.88938950 0.6617967 0.4498607
## 2:    humidity 0.08367092 0.2235791 0.3011838
## 3: cloud_cover 0.02693958 0.1146242 0.2489554
```

```
# Plot it
xgb.plot.importance(importance_matrix, top_n = 10, rel_to_first = TRUE)
```



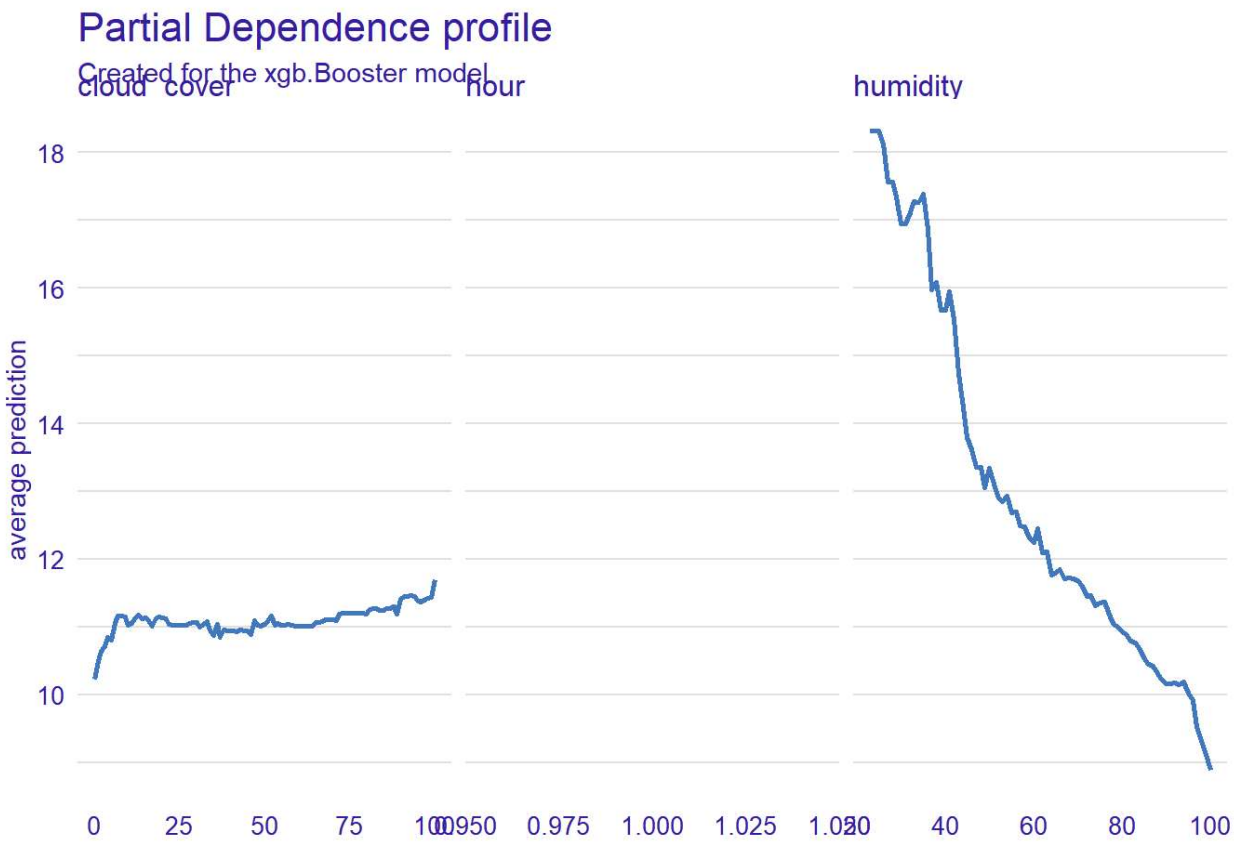

```
#Fature importance
importance_matrix <- xgb.importance(feature_names = colnames(X_train), model = xgb_model)
xgb.plot.importance(importance_matrix, top_n = 10)
```



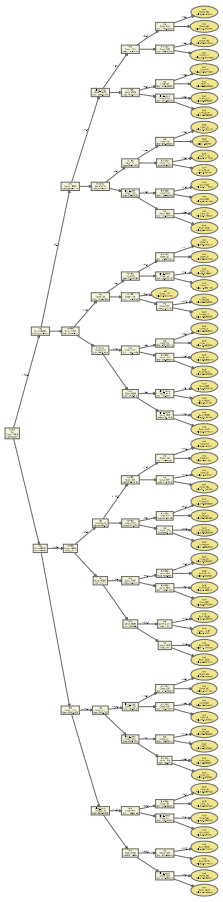
```
#Gain shows the average improvement in accuracy brought by splits using that feature.
#Cover indicates how many samples the feature split covered.
#Frequency counts how often the feature was used in trees.


### Partially dependence plots
explainer <- explain(model = xgb_model, data = X_train, y = y_train)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  xgb.Booster  (  default  )
##   -> data              :  78888  rows  4  cols
##   -> data              :  rownames to data was added ( from 1 to 78888 )
##   -> target variable   :  78888  values
##   -> predict function  :  yhat.default will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package Model of class: xgb.Booster package unrecognized , ver. Unknown , t
ask regression (  default  )
##   -> predicted values  :  numerical, min =  -7.109679 , mean =  11.3526 , max =  37.81057
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -11.05932 , mean =  -1.559995e-05 , max =  14.22901
##   A new explainer has been created!
```

```
pdp <- model_profile(explainer, variables = c("humidity", "cloud_cover", "hour"))
plot(pdp)
```



```
### I will look at the trees

xgb.plot.tree(model = xgb_model, trees = 0)  # shows Tree 0 visually
```



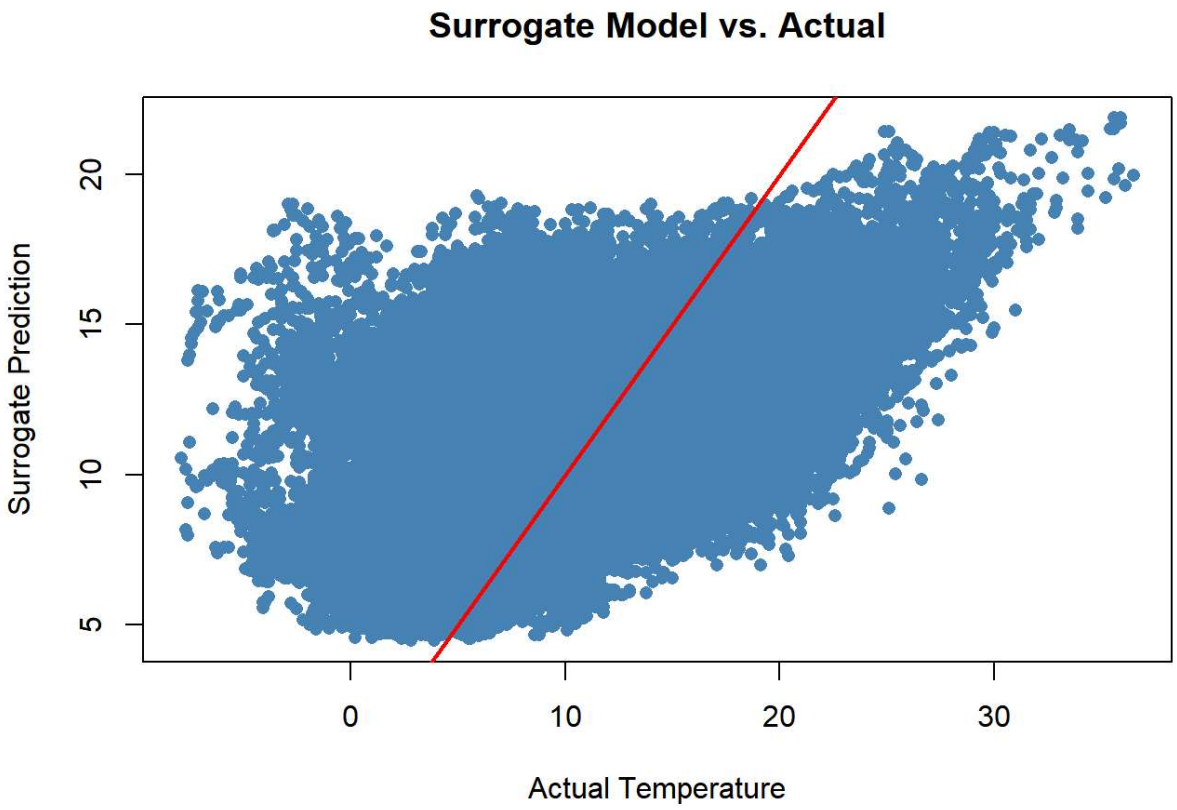
```
xgb_model_text <- xgb.dump(xgb_model, with_stats = TRUE)
cat(xgb_model_text[1:20], sep = "\n")  # look at the root decision logic
```

```
## booster[0]
## 0:[f1<123.5] yes=1,no=2,missing=1,gain=923381,cover=78888
## 1:[f1<86.5] yes=3,no=4,missing=3,gain=67581.9375,cover=26568
## 3:[f1<67.5] yes=7,no=8,missing=7,gain=9609.03125,cover=18576
## 7:[f3<5.5] yes=15,no=16,missing=15,gain=7066.53125,cover=14472
## 15:[f1<44.5] yes=31,no=32,missing=31,gain=1801.02832,cover=1546
## 31:[f1<16.5] yes=63,no=64,missing=63,gain=1855.48633,cover=642
## 63:leaf=1.29076445,cover=156
## 64:leaf=0.104229987,cover=486
## 32:[f2<79.5] yes=65,no=66,missing=65,gain=396.262695,cover=904
## 65:leaf=1.24788344,cover=462
## 66:leaf=0.844875872,cover=442
## 16:[f2<48.5] yes=33,no=34,missing=33,gain=4446.84375,cover=12926
## 33:[f1<58.5] yes=67,no=68,missing=67,gain=549.022583,cover=94
## 67:leaf=0.230714291,cover=41
## 68:leaf=-1.22055566,cover=53
## 34:[f3<99.5] yes=69,no=70,missing=69,gain=3319.59375,cover=12832
## 69:leaf=1.34742057,cover=7640
## 70:leaf=1.65940499,cover=5192
## 8:[f2<85.5] yes=17,no=18,missing=17,gain=1177.32812,cover=4104
```

```
### Finally, I Will train a surrogate model like a simple linear model to mimic it (because it is easier to explain)
### This gives me interpretable coefficients per variable– as a human-friendly proxy.

xgb_pred_train <- predict(xgb_model, newdata = X_train)
lm_surrogate <- lm(xgb_pred_train ~ X_train)
summary(lm_surrogate)
```

```
##
## Call:
## lm(formula = xgb_pred_train ~ X_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.5572  -3.2296  -0.2402   4.0820  17.8399
##
## Coefficients: (1 not defined because of singularities)
##                       Estimate Std. Error  t value Pr(>|t|)
## (Intercept)         22.5725795  0.1097876  205.602   <2e-16 ***
## X_trainhour                NA         NA       NA       NA
## X_trainday           0.0175427  0.0001536  114.208   <2e-16 ***
## X_trainhumidity     -0.1830195  0.0013984 -130.881   <2e-16 ***
## X_traincloud_cover   0.0005828  0.0004473    1.303    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.532 on 78884 degrees of freedom
## Multiple R-squared:  0.2708, Adjusted R-squared:  0.2708
## F-statistic:  9765 on 3 and 78884 DF,  p-value: < 2.2e-16
```

```
### lest see if surrogate model is really representative of the XGBOOS it is trying to mimic
### Now that I've built a linear model (lm_surrogate) to approximate my XGBoost predictions, here's how to ## ##compare those predictions to my actual observed values:


### get surrogate predictions
surrogate_pred <- predict(lm_surrogate, newdata = as.data.frame(X_train))

### compare real to observerd targets
plot(y_train, surrogate_pred,
     xlab = "Actual Temperature",
     ylab = "Surrogate Prediction",
     main = "Surrogate Model vs. Actual",
     col = "steelblue", pch = 16)
abline(0, 1, col = "red", lwd = 2)  # perfect fit line
```



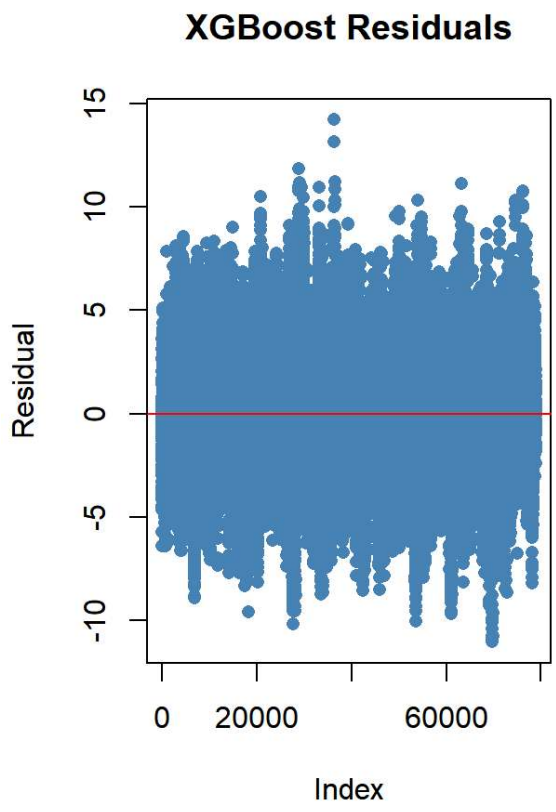

```
### evaluate rmse and r squared for both
library(Metrics)

rmse_surrogate <- rmse(y_train, surrogate_pred)
r2_surrogate <- summary(lm(y_train ~ surrogate_pred))$r.squared

rmse_surrogate
```

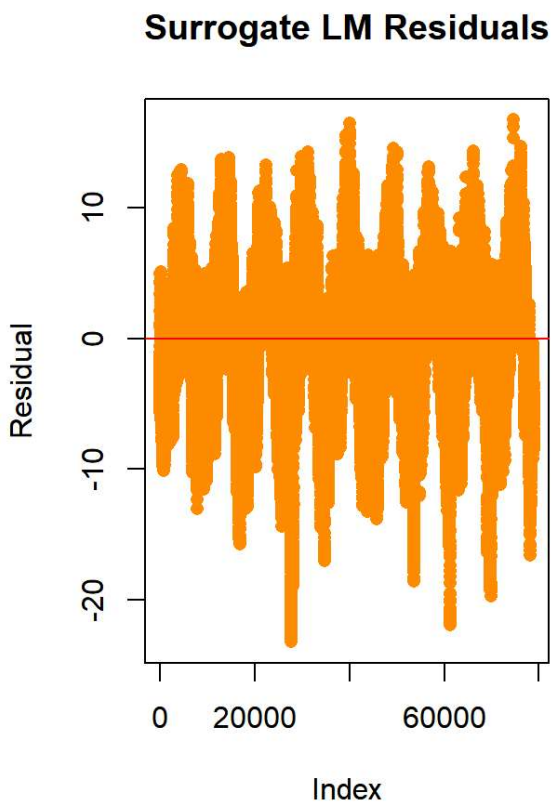
```
## [1] 5.249698
```

```
r2_surrogate
```

```
## [1] 0.2168264
```

```
#This indicates the surogate is very poor to use to explain the model

###Finally I Want to compare this side-by-side with XGBoost's accuracy or visualize residuals?

### First i generate teh predicitions
# XGBoost predictions (already available)
xgb_pred_train <- predict(xgb_model, newdata = X_train)

# Surrogate model predictions
surrogate_pred <- predict(lm_surrogate, newdata = as.data.frame(X_train))


# I compare teh residuals
resid_xgb <- y_train - xgb_pred_train
resid_surrogate <- y_train - surrogate_pred


## And i plot the residuals side by side
par(mfrow = c(1, 2))

plot(resid_xgb, main = "XGBoost Residuals", ylab = "Residual", xlab = "Index",
     col = "steelblue", pch = 16)
abline(h = 0, col = "red")

plot(resid_surrogate, main = "Surrogate LM Residuals", ylab = "Residual", xlab = "Index",
     col = "darkorange", pch = 16)
abline(h = 0, col = "red")
```

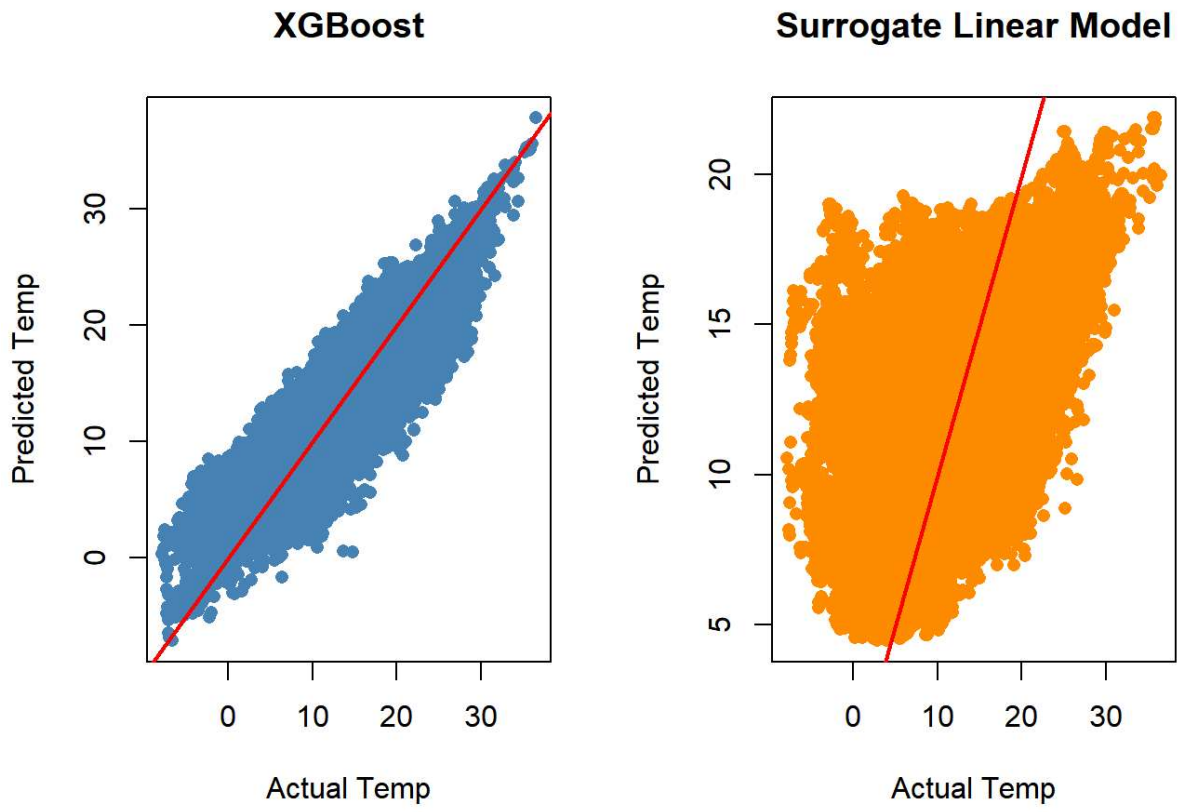




```
#This will show how "off" each model is across my training data—tight clouds around 0 are what I want.

### RMSE  and r squared for both

rmse_xgb <- rmse(y_train, xgb_pred_train)
rmse_surrogate <- rmse(y_train, surrogate_pred)

r2_xgb <- summary(lm(y_train ~ xgb_pred_train))$r.squared
r2_surrogate <- summary(lm(y_train ~ surrogate_pred))$r.squared

data.frame(
  Model = c("XGBoost", "Surrogate LM"),
  RMSE = c(rmse_xgb, rmse_surrogate),
  R_Squared = c(r2_xgb, r2_surrogate)
)
```

```
##          Model     RMSE R_Squared
## 1      XGBoost 2.470737 0.8267336
## 2 Surrogate LM 5.249698 0.2168264
```

```
###Visualize actual vs predicted
par(mfrow = c(1, 2))

plot(y_train, xgb_pred_train,
     xlab = "Actual Temp", ylab = "Predicted Temp",
     main = "XGBoost", col = "steelblue", pch = 16)
abline(0, 1, col = "red", lwd = 2)

plot(y_train, surrogate_pred,
     xlab = "Actual Temp", ylab = "Predicted Temp",
     main = "Surrogate Linear Model", col = "darkorange", pch = 16)
abline(0, 1, col = "red", lwd = 2)
```

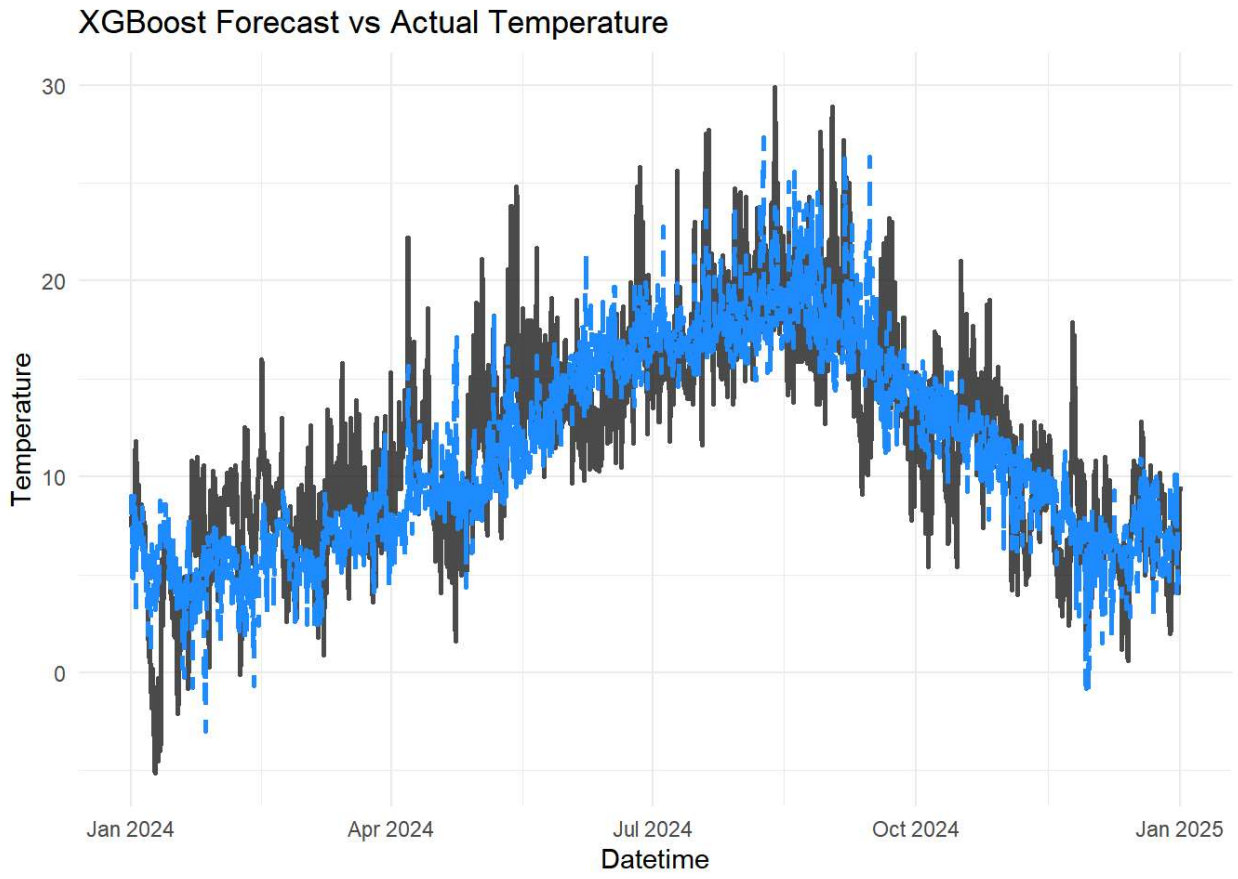


#These plots and stats will clearly show: #that the surrogate does not fully captures the core logic #hence the model I will use it is going to be explain mostly by trees.

3.6 **Model Evaluation**

- Models were evaluated using **Root Mean Squared Error (RMSE)** on a held-out test set.
- **ARIMA** performed reliably for short-term forecasts, while **Prophet** showed greater potential in multi-step predictions with complex patterns.

```
##### Visualize the predictions of best model - XGBoost


# Combine date and clock_time into a proper datetime
test$datetime <- as.POSIXct(paste(test$date, test$clock_time), tz = "UTC")

# Build comparison dataframe
plot_df <- data.frame(
  datetime = test$datetime,
  actual = test$temperature,
  predicted = xgb_pred
)

# Plot
ggplot(plot_df, aes(x = datetime)) +
  geom_line(aes(y = actual), color = "black", size = 1, alpha = 0.7) +
  geom_line(aes(y = predicted), color = "dodgerblue", size = 1, linetype = "dashed") +
  labs(
    title = "XGBoost Forecast vs Actual Temperature",
    x = "Datetime",
    y = "Temperature"
  ) +
  theme_minimal()
```




```
## This clearly shows that XGBOOST is the best model I can use to predict
```

4. Results

After training and evaluating all the models described on historical temperature data for The Hague, I compared their performance on the test dataset using a primary metric: *Root Mean Squared Error (RMSE)*.

Rank	Model	RMSE
1	XGBoost	3.17
2	Prophet	3.19
3	Linear Regression	5.27
4	ARIMAX	6.50 *
5	ARIMA	6.43
6	Seasonal Naive	6.39
7	Naive	6.70
8	ETS	6.72
9	Neural Network	10.27

The **Prophet model outperformed ARIMA and Multiple Linear Regresion** in the RMSE metrics, indicating a higher accuracy in forecasting daily temperatures, particularly over longer time horizons. However th XGBoost model was by far the best and with a very acceptable RMSE.

Visual comparison of actual vs. predicted temperature confirmed this.

---

5. Conclusion

This project demonstrated how machine learning models—specifically ARIMA, Prophet and XGBoost—can be applied to time series temperature data from the Meteo site to forecast future temperatures in The Hague. By leveraging historical records and advanced modeling techniques, I was able to generate accurate and useful predictions that could support planning in domains such as energy usage, agriculture, and public health.

The results revealed that **deep learning methods, particularly XGBoost**, are well-suited for modeling complex, temporal dynamics in meteorological data. However, the study also uncovered some limitations: - Limited geographic generalization: The model is localized to The Hague and may not perform equally well in other regions without retraining. - Data quality dependence: Occasional gaps in the Meteo data required careful imputation and added some uncertainty. - Model complexity: the XGBoost model is very complex, potentially limiting its practicality for lightweight applications. - Model difficulty to be explained easily

Future work could involve: - Incorporating additional weather variables (e.g., solar radiation or sea surface temperature). - Exploring ensemble methods that blend statistical and neural models. - Expanding the prediction scope to include uncertainty estimates or probabilistic forecasts.

This project serves as a foundation for further exploration into real-world, data-driven forecasting and illustrates the growing impact of machine learning in environmental and societal contexts.

6. References

Open Meteo - Site

https://open-meteo.com (https://open-meteo.com) Generated using Copernicus Climate Change Service information 2022 (Used as the primary dataset source for historical temperature and meteorological records in The Hague)

Time series Forecasting in Python - Book

Marco Peixeiro
Manning - 2022

Time series learning - youtube

Ritvikmath https://www.youtube.com/watch?v=ZoJ2OctrFLA&list=PLvcbYUQ5t0UHOLnBzI46_Q6QKtFgfMGc3 (https://www.youtube.com/watch?v=ZoJ2OctrFLA&list=PLvcbYUQ5t0UHOLnBzI46_Q6QKtFgfMGc3)