Machine Learning Project: Weather Prediction

Multi-Horizon Temperature Prediction for Bern, Switzerland

Authors:

Ephraim Elija Stiefel

Kaloyan Boyanov Anastasov

Rupert Sebastian Hunt

GSEM, University of Geneva

S403011 Machine Learning

Machine Learning Project

Fall Semester 2025

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations

Abbreviation	*Full Form*
1SE	One-Standard-Error
ARIMA	Autoregressive Integrated Moving Average
CV	Cross Validation
NWP	Numerical Weather Prediction
MAE	Mean Absolute Error
EDA	Explanatory Data Analysis
OLS	Ordinary Least Squares
QFE	Atmospheric pressure at barometric altitude
QFF	Atmospheric pressure reduced to sea level
RF	Random Forest
RSS	Residual Sum of Squares
SE	Standard Error

1 Introduction

In the 18th and 19th centuries, when mercury or aneroid barometers were expensive and not widely available, and weather forecasts on television or even on cell phones were unimaginable, people in Switzerland had to resort to other methods to predict the weather. They used a “Wetterfrosch” (German: weather frog) to make their predictions. They filled a glass or bottle with some water and soil and placed a European tree frog on top. A small ladder or branch was placed next to it. Then it was up to the frog: if it climbed up the ladder, it was said to predict fair and sunny weather. If it stayed at the bottom, it was said that there would be rainy, bad weather. This practice was widespread in German-speaking Switzerland and Central Europe. Nowadays this humorous piece of folklore to smile about, has been replaced by weather forecasts algorithms, such as the one this project aims to implement. This chapter provides an overview of the project and outlines the structure of the report.

1.1 Problem Statement

The goal of this project is to develop a machine learning model to predict a single meteorological variable, namely the air temperature at 2 m above ground (tre200h0) for a single target location, Bern, for multiple time horizons. The time horizons are 12h, 24h and 48h into the future.
The input data ‘train.csv’, which is available for analysis and model training, is explained in detail in Chapter 2: Data Set Description. The exogenous variables in this dataset capture meteorological data from 10 different weather stations in Switzerland, as well as temporal indicators that specify the time of day and season of the recording. The project runs parallel to a Kaggle competition, which uses a separate dataset, ‘test.csv’, also explained in Chapter 2.
In the weather forecasting industry, short-term weather forecasts are based on a combination of different methods. The industry standard approach is to use numerical weather prediction (NWP) models, which use supercomputers to solve a large number of mathematical equations of physical laws that affect the atmosphere. NWP is often supplemented by statistical methods such as time series analysis of historical data. However, in this project time series models such as autoregressive integrated moving average (ARIMA) are not applicable because the dataset does not contain a timestamp. The dataset is not order chronologically and temporal dependencies cannot be modeled. As an alternative, the project employs regression-based approaches to predict future temperatures. Thus, a variety of linear models such as Linear Regression, Ridge and Lasso are implemented, as well as non-parametric models such as K-Nearest Neighbors, Random Forest, and Boosting. To evaluate the predictive power of each model in this project, the Mean Absolute Error (MAE) metric is used.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|Y_i - \widehat{Y_\iota}|$$

The average MAE is then calculated for the final model. The following section explains the structure used to solve this problem.

1.2 Report Structure

To solve the multi-horizon forecasting problem discussed in the previous section, this project takes a structured approach. Following this introductory chapter, the available datasets are examined in Chapter

2. Next, Chapter 3 uses exploratory data analysis (EDA) to identify revealing patterns and insights in the data, both for parametric and non-parametric models. Topics discussed include, among others, missing values, outliers, and scaling. Further down in the report, Chapter 4 explores different models that are used for the given problem. The different models are described and the necessary steps for their training and testing in Python, such a hyperparameter tuning are discussed. Chapter 5 uses the MAE evaluation criteria to compare and interpret the performance of the various predictions models. The final verdict will be given in Chapter 6, which compares the different forecasting models and recommends the final model to use for this prediction task. Chapter 7 concludes the report by summarizing the main findings and outlining future directions.

2 Data Set Description

This chapter describes the available datasets. It takes a closer look at the data source, the characteristics, and the structure of both the 'train.csv' file, which is used to train and test the model, and the 'test.csv' file, which is available for the Kaggle competition.
As mentioned in Chapter 1, the project uses a dataset of hourly meteorological measurements collected from ten different weather stations spread across various climate regions of Switzerland. The 'train.csv' is used to fit and test the machine learning models. This dataset contains 92 columns and 7579 rows. 89 of these columns in the dataset represent input features while the remaining columns include the target variables. Feature names follow a structured naming. For example, the feature 'fkl010h0_ANT' consists of a prefix 'fkl010' which stands for the measured meteorological variable in this case "wind speed", followed by a suffix 'ANT' which denotes the weather station where the variable is measured. The complete list of the prefixes and their associated meteorological variables is listed and explained in table 1, while table 2 lists all the suffixes and their associated weather station's location.

PREFIX	VARIABLE	UNITS
fkl010h0	Wind speed scalar	Hourly mean in m/s
fkl010h3	Gust peak	Hourly maximum in m/s
gre000h0	Global radiation	Hourly mean in W/m²
pp0qffh0	Atmospheric pressure reduced to sea level (QFF)	Hourly mean in hPa
prestah0	Atmospheric pressure at barometric altitude (QFE)	Hourly mean in hPa
rre150h0	Precipitation	Hourly total in mm
sre000h0	Sunshine duration	Hourly total in minutes
tre200h0	Air temperature	Hourly mean in °C 2 m above ground
ure200h0	Relative air humidity	Hourly mean in percentage 2 m above ground

Table 1: Variable prefixes with their respective meteorological meanings and units.

There are nine different meteorological variables. For each meteorological variable there are 10 stations across Switzerland taking measures. For each combination of a weather station and meteorological variable a separate input feature is defined, resulting in multiple features per variable. An exception is atmospheric pressure reduced to sea level (QFF) variable, for which there are only 5 weather stations taking measurements, the stations are: BAS, GVE, INT, LUG and SIO. Therefore, there are fewer input features compared to other meteorological variables. Using multiple stations is helpful for predicting the temperature in Bern, as local temperature can be influenced by regional weather systems such as wind conditions and pressure changes beyond the target location.

SUFFIX	CITY / STATION	CANTON
ANT	Andermatt	Uri (UR)
BAS	Basel	Basel-Stadt (BS)
DAV	Davos	Graubünden (GR)
DOL	Dole (France, near Jura)	(France)
GVE	Geneva	Geneva (GE)
INT	Interlaken	Bern (BE)
LUG	Lugano	Ticino (TI)
SIO	Sion	Valais (VS)
STG	St. Gallen	St. Gallen (SG)
ZER	Zermatt	Valais (VS)

Table 2: Variable suffixes with the respective name of the city/station and its location.

In addition to these meteorological variables, the following table summarizes the variables relating to the temperature in Bern and the temporal characteristics used for the forecast. These include the current temperature, the 24-hour lag temperature, and the three target temperatures of +12h, +24h, and 48h. The dataset includes temporal characteristics such as the hour of the day and the season, which provide temporal context for each observation.

VARIABLE	MEANING	UNITS
tre200h0	Air temperature in Bern at the current time	Hourly mean in °C 2 m above ground
tre200h0_lag24h	Bern temperature 24 hours earlier	Hourly mean in °C 2 m above ground
target_tre200h0_plus12h	Temperature in Bern 12 hours ahead	Hourly mean in °C 2 m above ground
target_tre200h0_plus24h	Temperature in Bern 24 hours ahead	Hourly mean in °C 2 m above ground
target_tre200h0_plus48h	Temperature in Bern 48 hours ahead	Hourly mean in °C 2 m above ground
hour	Hour of the day (0–23)	-
season	Season label (Winter, Spring, Summer, Autumn)	-

Table 3: Temperature variables for Bern and temporal characteristics hour and season.

The file 'test.csv' used for the Kaggle competition has the same structure, with the difference that it does not contain target temperatures, as these are predicted by the models trained in Chapter 4. When this report refers to the dataset without specifically mentioning the file 'test.csv', it is referring to the file 'train.csv'.

3 Exploratory Data Analysis and Data Preprocessing

The goal of this chapter is to understand in more detail how the different variables of the dataset behave. Do the variables display highly skewed distributions, or do they follow a normal distribution? Are the within group correlations of the variables high or low? Are there any missing values or outliers that could affect the models? Answering these questions helps to decide which feature engineering and preprocessing steps need to be applied on the dataset before fitting the parametric models. While the sections 3.1 to 3.4 examine the steps of EDA and Data Preprocessing for parametric models and KNN, section 3.5 looks at how the data needs to be preprocessed for non-parametric models such as Bagging, Random Forests and Boosting.

3.1 Correlation within Meteorological Variables across Stations

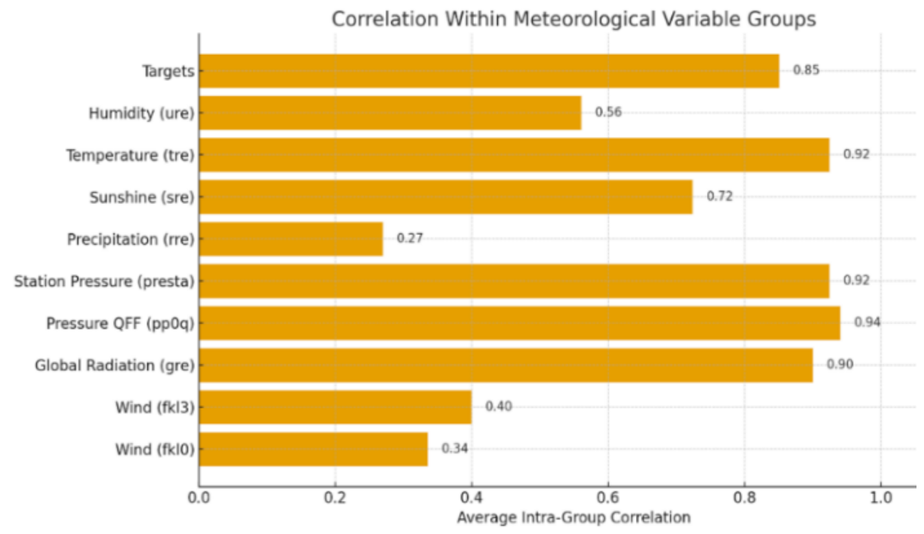


Figure 1: Within correlation measurement for each meteorological variable.

If there is high correlation between the 10 stations for a given meteorological variable, this indicates the weather stations across Switzerland observe similar weather conditions and they move in sync. Feature engineering is applied to highly correlated weather variables by creating two new variables: mean and standard deviation. The mean feature calculates the mean across the 10 weather stations for a given meteorological variable, while the standard deviation feature represents the average variability across the stations. These two new variables are now sufficient to represent the weather condition for a given meteorological variable across Switzerland. Consequently, the 10 standalone station-specific input features for this given meteorological variable are dropped. This helps to reduce the number of redundant features from the total number of features used by the models leading to more stable, precise, and accurate models. It is not efficient to keep 10 features that share the same signal between each other, as this would create a problem of multicollinearity. This is further explored in the parts below.

3.2 Missing Values

Only 1.98% of the observations, i.e., rows in the dataset, contain missing values. Since the missing values are few and not concentrated in any specific station or sensor. As this represents only a very small part of the total number of observations, these rows are removed from the dataset.

3.3 Outliers

Outliers are values far outside the normal range, often caused by sensor glitches or irregular spikes. They are important to detect because models like Linear Regression and ridge regression are very sensitive to extreme points. Tree-based models are less affected as they can handle outliers better.
When studying the distribution of all meteorological variables, based on the 1.5 IQR rule, four variables had more than 5% of outliers: 'sre000h0' (sunshine duration), 'rre150h0' (precipitation), 'gre000h0' (global radiation), 'fkl010h0' (wind speed), and 'fkl010h3' (gust peak). These variables naturally show strong fluctuations due to rapid changes in weather conditions such as sudden rainfall, wind gusts, or short periods of intense sunlight, which explains why they contain many extreme values. Since these outliers

reflect real atmospheric events rather than measurement errors, they are not automatically removed. However, because linear models are sensitive to extreme observations each case should be analyzed individually to understand if they represent too unrealistic and rare events.

3.4 Variables Analysis

In this part of Chapter 3, the analysis of various variables in the data set will be considered in more detail. While the project's notebook analyzes all the variables, the report discusses only the most important findings for a selection of variables for the sake of brevity.

3.4.1 Wind Speed and Gust Peak Analysis

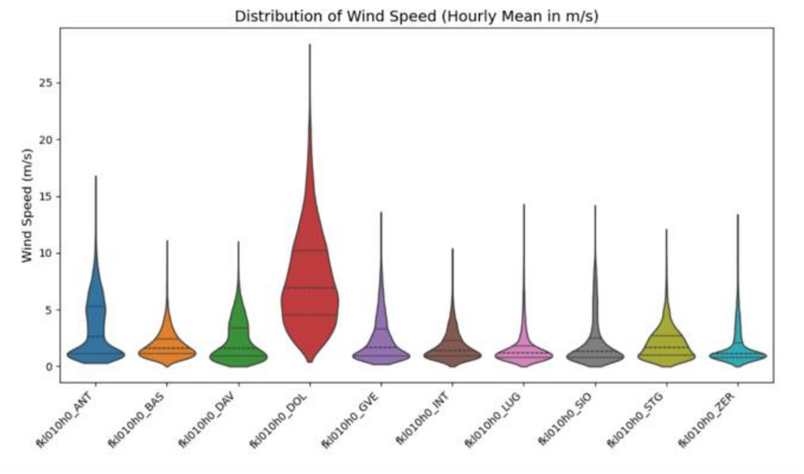


Figure 2: Violin plot – wind speed features 'fkl010h0'

The graph shows that wind speed remains mostly low around 0 with occasional sharp peaks caused by short period gusts. These peaks represent rare and extreme gust events which are not representative of typical hourly behavior. Fkl is also one of the weather meteorological variables with more than 5% of outliers present in the observations. First, to mitigate the influence of these extreme measurements a 2% upper capping is applied to force the data to stay within a specific range and reduce influence of extreme rare measures which can disturb the accuracy of the models. Capping is a technique implemented to limit the influence of extreme values in a variable by replacing them with a percentile threshold, as for example the 98th percentile for upper capping. Capping helps prevent extreme, unrealistic and rare events from distorting the distribution and the models fit. Limiting the top 2% of the distribution helps the model to retain realistic wind behavior and prevent outliers from dominating the feature variance leading to overfitting. Only upper capping is applied since there are no negative extreme measurements.
The distribution of the features also present rightly-skewed data so box-cox transformation is applied. Box-Cox transformation is applied to reduce the heaviness of the tails and improve the distribution symmetry by making the data follow a distribution closer to a normal one. This helps to prevent extreme values from dominating the model learning and improves the model performance.
Furthermore, robust scaling is also applied in order to help comparison between variables in the final model but also to ensure that the mean and variance won't be pulled by extreme outliers. For all linear models and for KNN, scaling is necessary because these algorithms are sensitive to the magnitude of the predictors. The variables in the dataset are measured on many different units, and thus scales. Without scaling the model would give too much influence to features with naturally larger ranges.

The correlation between stations is low, not allowing to apply mean and std feature engineering.
The same observations can be also made about the second wind variable 'fkl010h3' (gust peak) not represented here. both variables are kept in the model even if they both represent a similar weather measure. The model uses mean speed 'fkl010h0' for general wind conditions whereas the variables 'fkl010h3' for extreme and sudden turbulences.

3.4.2 Sunshine Duration and Global Radiation Analysis

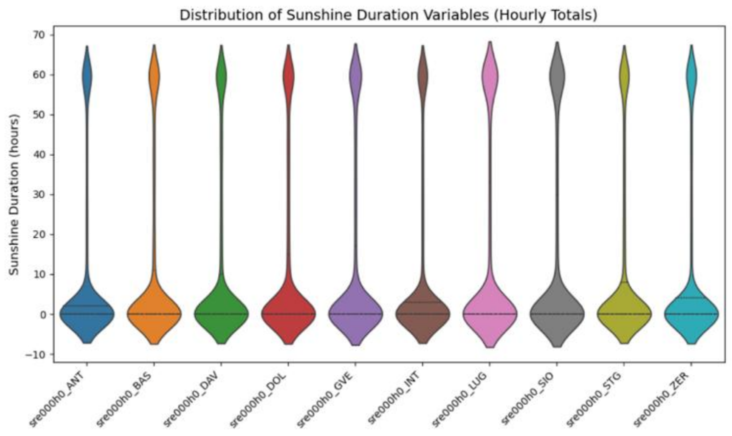


Figure 3: Violin plot - sunshine duration features 'sre000h0'

Sunshine duration shows a strong concentration of values near zero, representing nighttime or cloudy conditions. There are also long tails reaching high values representing extended sunny periods. These long upper tails do not represent erroneous, extreme, or rare observations but plausible and real periods of long sunshine duration. No capping is applied in this case.
The variables have a very high correlation at 0.92. Thus, the mean across the weather stations is taken as a representative of sunshine duration across Switzerland and the individual measurements for each station are dropped. Thus, the only representative variable for sunshine duration weather measure is the mean across the stations. Taking the mean across measurement stations captures the central tendency. Another variable taking the mean for the standard deviations across the stations was also created to capture the variation. Together they retain more information and will improve the model performance. The same observations can also be made about the variable of global radiation 'gre000h0' which is not shown here.

3.4.3 Precipitation Analysis

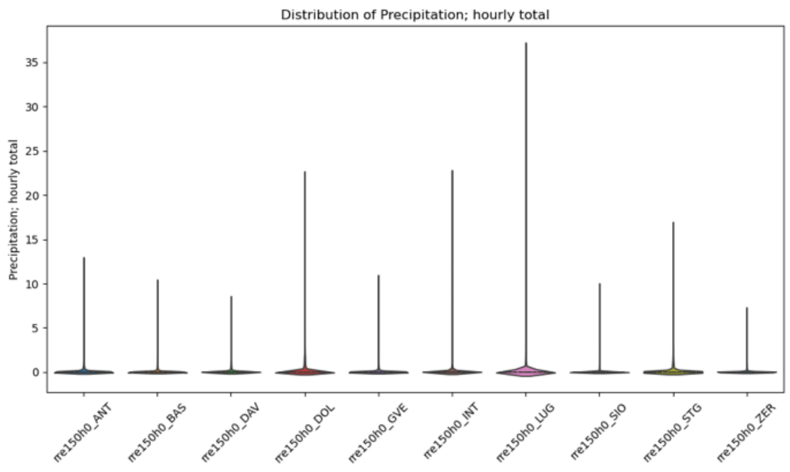
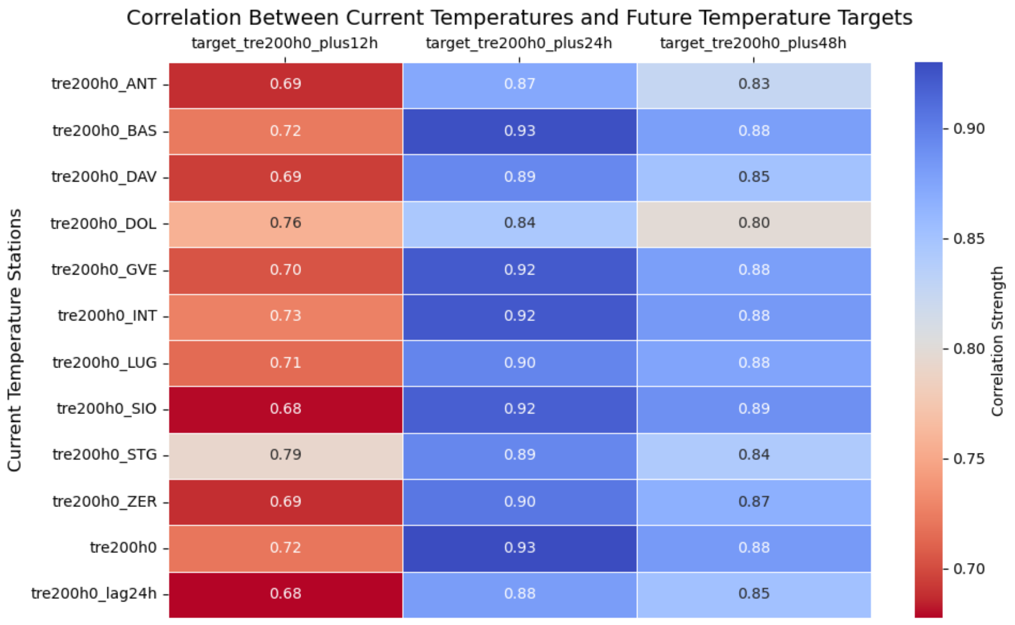


Figure 4: Violin plot - precipitation features 'rre150h0'

The precipitation distribution is dominated by zeros, with occasional surges during rain events that stretch the tails upward. There are no negative values and lower tail extreme observations. This structure reflects the intermittent nature of rainfall, where long dry periods alternate with short, intense bursts. The influence of rare heavy rainfall events needs to be limited in the model. The peaks represent real events, but they do not represent typical rainfall conditions and can distort the models by pulling the mean and scale upward.
As such, on this variable a 2% upper capping is applied as well as a robust standard scaling. The data is also rightly skewed, thus box-cox transformation is applied. As the correlation is low, no feature engineering is made.

3.4.4 Temperature Analysis

Since the main objective of this project is to forecast the temperature in Bern, it is reasonable to examine the temperature variables more closely. Temperature is typically one of the most informative meteorological features, and measurements from different stations often move together due to shared regional weather patterns.
The correlation of temperature across stations is 0.92. Thus, the mean and the standard deviation across stations is taken to represent this meteorological variable. Nevertheless, temperature variables for STG and INT are retained as standalone variables. From the table below, it is observed that all stations show moderate to very high correlation with Bern's temperature forecast, confirming that temperatures behave similarly across regions. Yet, STG and INT are two weather stations consistently ranked among the strongest predictors across the three forecast periods, justifying their retention as standalone variables in addition to the mean and standard deviation across weather stations. The variable tre200h0_lag24h is also kept as a standalone variable, since it is directly linked to Bern, indicating the temperature in Bern 24 hours ago. The variable 'tre200h0' was also listed as a standalone variable, as it indicates the hourly temperature in Bern and is therefore closely linked to the forecast for Bern.



	target_tre200h0_plus12h	target_tre200h0_plus24h	target_tre200h0_plus48h
tre200h0_ANT	0.69	0.87	0.83
tre200h0_BAS	0.72	0.93	0.88
tre200h0_DAV	0.69	0.89	0.85
tre200h0_DOL	0.76	0.84	0.80
tre200h0_GVE	0.70	0.92	0.88
tre200h0_INT	0.73	0.92	0.88
tre200h0_LUG	0.71	0.90	0.88
tre200h0_SIO	0.68	0.92	0.89
tre200h0_STG	0.79	0.89	0.84
tre200h0_ZER	0.69	0.90	0.87
tre200h0	0.72	0.93	0.88
tre200h0_lag24h	0.68	0.88	0.85

Figure 5: Correlation heatmap between current temperatures and future temperatures

Finally, robust scaling was applied to the kept variables to ensure comparability between variables and a correct interpretation of the results by the models.

3.4.5 Atmospheric Pressure Analysis

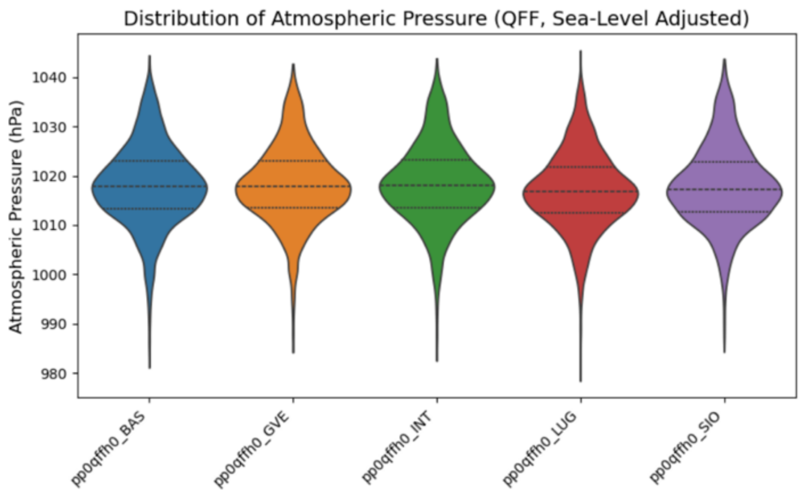


Figure 6: Violin plot - atmospheric pressure reduced to sea level features 'QFF'

Atmospheric pressure reduced to sea level (QFF) variables do not show extreme peaks or heavy tails. From the violin plots it is observed that the values are clustered around the mean of 1015 hPa with a symmetrical spread, showing a near-normal shape. Following these observations, box-cox transformation and capping are not applied. There are no extreme outliers or unrealistic measures in the observed variables and 'pp0qffh0' is not part of the weather measures with more than 5% of outliers.

Correlation between stations is high, thus the two variables mean, and standard deviation are created. These two variables are therefore representative for the 10 stations. The individual 5 variables are dropped. As before, robust standard scaling is also applied.
The same observations can be made about the variable of atmospheric pressure at barometric altitude. Although QFF and QFE both measure the same weather variable of atmospheric pressure, they use different reference levels. Therefore, both quantities are needed for atmospheric pressure measurements.

3.4.6 Relative Air Humidity Analysis

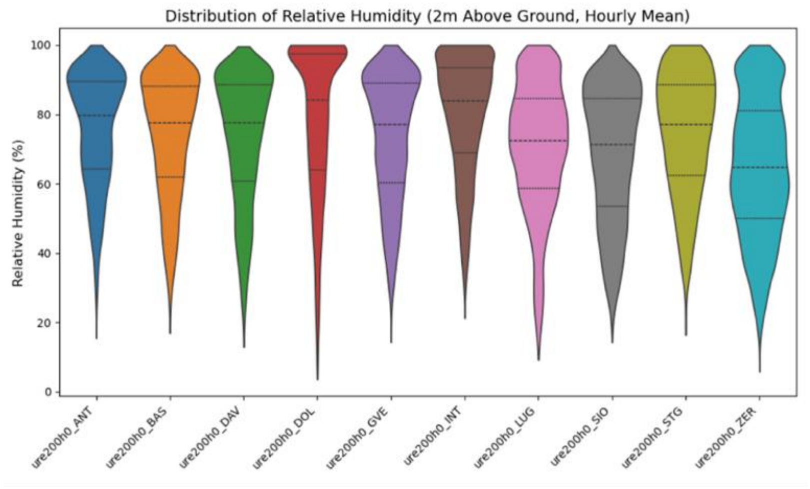


Figure 7: Violin plot - relative air humidity features 'ure200h0'

Humidity is high across all stations. Most of the variable's observations lie between 60-100%. The lower tails of the violin plots extend to 10-20% humidity. These lower observations shouldn't be considered as unrealistic, rare events or extreme measures but they represent plausible meteorological events, such as dry air events. Thus, no capping is applied.
Furthermore, correlation between variables is low thus the mean and standard deviation variables are not created. Only robust scaling was applied to the variables for scaling reasons. There is also no right-skewness that needs to be corrected by box-cox transformation.

3.4.7 Season Analysis

Season is a categorical variable which is converted into dummy numerical variables as machine learning models cannot directly interpret text categories.

3.4.8 Hour Analysis

Hour feature is encoded using two new trigonometric encodings, sin and cos, in order to reflect the natural 24-hour cycle in temperature. Hours naturally repeat into cycle: After 23 comes 0 again. If hours are not converted using sin and cos to reflect the seasonal pattern but remain as numerical values the model will assume 23 is far from 0.

3.5 Data Preprocessing for Non-Parametric Models

For non-parametric models such as Bagging, Random Forests and Boosting, the preprocessing requirements are much lighter compared to linear models or KNN. These tree-based methods do not need scaling, log-transformations or strict outlier handling, because of how decision trees split the feature space. A tree only checks whether a value is smaller or larger than a threshold. Thus, the absolute scale of the variable has no influence on the split. For the same reason, trees are naturally robust to skewness and outliers since extreme values are simply placed in separate branches without distorting the model. Because of this property, Random Forests can work directly with the raw variables, as they internally manage non-linearities and variable interactions. Boosting also follows this behavior, since it builds many shallow trees that focus on correcting previous errors. These models therefore do not rely on the strong assumptions of linear models and will only need a few preprocessing steps as detailed below.

In order to allow these models to capture temporal temperature patterns, two new features were engineered around the temperature variable for Bern 'tre200h0'. Because the data in the dataset 'train.csv' is not chronologically ordered, it was not possible to use lagged variables with the shift operator to account for temporal patterns. Yet, temperature data is highly influenced by the season and the hour of the day. To capture this, the dataset was grouped by season and hour of the day. The following two variables "mean season-hour temperature" and "standard deviation of season-hour temperature" are created:

$$\mu_{s,h} = \mathbb{E}[tre200h0 \mid season = s, hour = h]$$
$$\sigma_{s,h} = Std[tre200h0 \mid season = s, hour = h]$$

These two variables indicate the mean temperature level as well as the variability expected at a given hour and specific season. They allow to summarize historical behavior without relying on temporal ordering. A third variable capturing "temperature anomaly", is created which captures deviations from the expected weather behavior. It relies on the "mean season-hour temperature" variable created previously.

$$temp\_anomaly = tre200h0 - \mu_{s,h}$$

This variable measures if the current temperature is unusually warm or cold relative to the baseline measurements for an hour and season. These new features allow to encode temporal structure in tree-based models and enable the models to distinguish between expected temperature measures and abnormal temperature conditions. Tree based models do not understand time by themselves. If a temporal structure is not explicitly provided, the model will infer this pattern from the raw variables which is difficult and inefficient. By introducing these three new features the temporal knowledge of the data is translated into features the models can easily use. After creating new features, the variable hour is dropped. Hour was previously encoded using trigonometric function to represent the 24h cyclical time, this is also used by RF and Boosting models. Finally, Season variable is transformed into a dummy variable, as tree-based models with scikit-learn require all input features to be numerical.

4 Models Overview and Training

This chapter describes the different models applied in the project and explains how they were implemented in Python.

Before implementing the different models, the dataset is split into predictor variables X, which contain all the variables that can be used to predict future temperatures, and target variables y, which correspond

to the temperature in Bern for three time horizons (12h, 24h, 48h).
After that, the data is split into training and test sets for each time horizon. The models are fitted on the training set, while the test set is used to evaluate the models on unseen data. The goal of the test set is to approximate as closely as possible the model's real performance (Engelke, 2025a, p.1). The next sections explain how to train and tune each model.

4.1 Linear Regression

4.1.1 Model Overview

Linear Regression models the relationship between a response variable y and one or more predictor variables X. The relationship is modeled using the linear function $f(x_i)$.

$$f(x_i) = \beta_0 + \beta_1 x_i$$

The goal is to predict the response variable $\hat{y}_i = f(x_i)$ (Engelke, 2025b, p.2). In this project, the X variables represent all the meteorological and temporal variables except for the target temperature variables (12h,24h,48h) which are represented by the y variable. To model this relationship, the linear equation can be extended to include multiple predictors (Engelke, 2025b, p.5). Thus, the Linear Regression model used for weather forecasting with the available data resembles the following:

$$f(x_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \ where \ i = 1, \ldots\ldots, n$$

The model assumes that the actual underlying function $f(x_i)$ is linear in the predictors. However, if the assumption of linearity does not apply, a simple linear fit is incorrect. The model will not fit the data well resulting in underfitting. This is an example of how simple parametric models such as Linear Regression have low flexibility, i.e., high bias but low variance (Engelke, 2025a, p.5). In order to model nonlinear relationships, further adjustments are necessary through transformations such as squaring or multiplying the predictors (Engelke, 2025b, p.8). Although this is the classic approach, its implementation in this project did not lead to better results. Linear regression was used as a baseline model for comparison with more flexible models.

4.1.2 Model Implementation

In Python, the *LinearRegression* class from *sklearn.linear_model* (scikit-learn package) is used. The model learned the relationship between the predictors X and the response variable y using the training dataset. *LinearRgression* uses Ordinary Least Squares (OLS) to estimate the optimal coefficients. The ordinary least square method minimizes the Residual Sum of Squares (RSS) between the observed y and the model predicted $\hat{y}$. These estimates of the coefficients are then used in the model to describe the relationship between X and y (Engelke, 2025c, pp.1-3). Linear Regression has no hyperparameters. Therefore, tuning these parameters is not necessary.
Finally, using the best coefficients, predictions for Bern temperature for the three future time horizons (12h, 24h, and 48h) based on unseen data are calculated. The performance for each of the prediction models is evaluated using the MAE.

4.2 K-Nearest-Neighbors

After examining Linear Regression, this chapter takes a closer look at K-Nearest Neighbors (KNN).

4.2.1 Model Overview

KNN is a non-parametric model, i.e., it does not assume a specific function, as is the case with Linear Regression. Furthermore, the model does not attempt to learn coefficients during training in order to explain the relationship between X and y. Instead, KNN predicts the target value y for a new point based on the average of the y values of the K nearest neighbors of the new point in the training data:

$$f(x_0) = Ave(Y|X\epsilon N_k(x_0) = \frac{1}{K}\sum_{x_i \epsilon N_k(x_0)} y_i \ where\ i = 1, \ldots.., n$$

$N_k(x_0)$ are the K nearest points x_i to x_0 (Engelke, 2025d, p.6). Since KNN is a distance-based model that uses the distance between data points to make predictions about unknown data, standardization is important to prevent features with large and different scales from dominating the distance calculation. As the prediction depends on how many K neighbors the model considers, K is a tuning-parameter that can be adjusted.

The choice of K determines the bias-variance tradeoff. A small K leads to high variance and overfitting, while a large K leads to high bias and underfitting. A major disadvantage of this model is the so-called "curse of dimensionality", which means that the performance of KNN decreases as the number of predictor variables in X increases (Engelke, 2025d, p. 7). This could prove problematic for predicting temperatures based on many variables. Nevertheless, the model was implemented because it allows for good interpretability.

4.2.2 Model Implementation

In Python, the *KNeighborsRegressor* class from *sklearn.neighbors* (scikit-learn package) is used. The first step in implementing the KNN model is to define a range of values for the number of neighbors K to test. The goal is to find a value of K that is best suited for predictions and, in addition, a value of K that optimizes for the bias-variance tradeoff using the One Standard Error (1SE) rule. To obtain these values, the model is evaluated using F-fold cross validation (e.g., F=10). The training data must first be divided into approximately equally sized (n/F) folds C_1, …, C_F. For each fold the KNN model is trained on the F-1 folds and evaluated on the remaining fold based on the MAE.

For each value of K, in the code a range of values between 1 and 50, a MAE value is calculated per fold. The cross-validated MAE for each unique value of K is then calculated as the average MAE across all F folds (Engelke, 2025e, p.2). In Python, hyperparameter tuning is performed using GridSearchCV. The cross-validated MAE for each unique value of K is represented by:

$$CV_{(F)}(K) = \frac{1}{F}\sum_{f=1}^{F} MAE_f(K)$$

Based on this result, the K value that minimizes the cross-validated MAE can be determined. The model with this number of neighbors has the highest predictive performance on average across all folds on which it was trained and tested.

In addition, a more conservative approach for model selection is applied using the 1SE rule. The goal is to find the largest value of K whose cross-validated MAE is within the 1SE of the minimum cross-validated MAE. The one standard error rule choses the simplest model with a cross-validation MAE which is undistinguishable from the minimum error model (Engelke, 2025e, p.9). This approach leads to a model with a lower variance and better robustness. This model thus generalizes better on unseen data. The 1SE decision for the value of K is represented by:

$$CV_{(F)}(K) \leq CV_{(F)}(\widehat{K}_{min}) + SE\left(CV_{(F)}(\widehat{K}_{min})\right)$$

The following figure illustrates this using the example of the KNN model for the 12-hour forecast. The lowest blue point represents the lowest cross-validated MAE, $CV_{(F)}(\widehat{K}_{min})$, while the K value that minimizes the cross-validated MAE, $\widehat{K}_{min}$, is 7. The 1SE rule selects a more conservative tuning parameter of 9, $\widehat{K}_{SE}$, result of the cross-validated MAE for 1SE, $CV_{(F)}(K)$, represented by the blue dotted vertical line, the highest value of K below the green line drawn at the lowest cross-validated MAE + standard error of t: $CV_{(F)}(\widehat{K}_{min}) + SE\left(CV_{(F)}(\widehat{K}_{min})\right)$

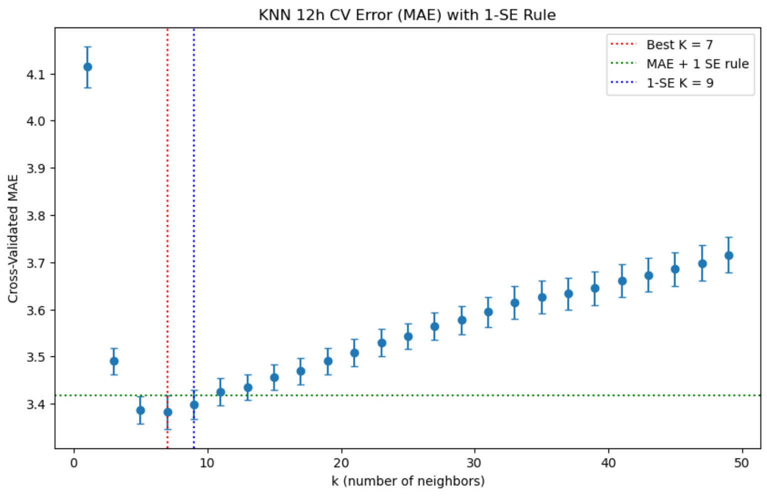


Figure 8 Cross Validated MAE chart for K-Nearest-Neighbors 12h horizon

Finally, predictions for Bern temperature, on unseen data, for the three time horizons (12h, 24, 48h) based on the best K and the 1SE K are performed. The performance for each of the prediction models is evaluated using the MAE.

4.3 Ridge and Lasso Regression

The next models to be considered are Ridge and Lasso. Both are quite similar to Linear Regression. The two are covered in the same section because the difference between them is mainly based on the L_1 and L_2 norms, which are explained in more detail below.

4.3.1 Model Overview

Linear Regression minimizes only the loss function (Engelke, 2025f, p. 3), Lasso and Ridge extend Linear Regression models by adding a penalty term to the loss function (MSE) to shrink/regulate the coefficient estimates and reduce overfitting. In Ridge, this is the L_2 penalty, the sum of the squared coefficients, while

Lasso uses the L_1 penalty, the sum of the absolute values of the coefficients (Engelke, 2025f, p.8). The resulting loss functions for Ridge and Lasso are shown below:

$$J(\theta)_{Ridge} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{p}\beta_j^2$$

$$J(\theta)_{Lasso} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{p}|\beta_j|$$

Ridge and Lasso are trained using the squared error loss function but the choice of the optimal regularization parameter α is selected using MAE, as MAE is the primary performance metric used to evaluate models in this project.

The penalty terms encourage smaller coefficient values, thus minimizing the loss function and reducing model complexity (Engelke, 2025f, p. 5). The resulting constraints enable Lasso to perform automatic variable selection by setting some coefficients exactly to zero, while Ridge retains all predictors and only shrinks some of them to values close to zero (Engelke, 2025f, pp.8-9).

The degree of shrinkage is controlled by the tuning parameter α, as shown in the formulas above. In this project, the optimal value of α is selected using cross validation MAE. Both Ridge and Lasso are useful for reducing overfitting to the training data, thereby improving performance on unknown test data. Both also require standardization of the predictors, as regularization treats all predictors equally when using the L_1 or L_2 norm (Engelke, 2025f, p.2).

4.3.2 Model Implementation

In Python, the *Ridge* class from *sklearn.linear_model* (scikit-learn package) is used to implement Ridge Regression. For Lasso Regression, the *Lasso* class is used from the same package. The code for Lasso and Ridge is similar to the one for Linear Regression, the difference is that a tuning parameter α is introduced. As with KNN, a range of potential values for α is first defined. This is important since the performance of both models depends entirely on the value of this tuning parameter.
The next step is to use F-fold cross-validation (e.g., F = 10) to evaluate the model as previously applied for KNN. Cross-validated MAE estimates are obtained for each value of α (Engelke, 2025e, pp.1-2). In Python, this is performed using GridSearchCV. The cross-validated MAE for each value of α is represented by:

$$CV_{(F)}(\alpha) = \frac{1}{F}\sum_{f=1}^{F} MAE_f(\alpha)$$

Based on these results, the value of α that minimizes the cross-validated MAE is identified. On average, this value has the highest predictive performance over all the folds on which it was trained and tested.
As before, an alternative is to use the 1SE rule to find the largest alpha value whose cross-validated MAE still lies within one standard error of the minimum MAE. This will lead to a simpler model that is statistically indistinguishable from the best performing model. In Ridge and Lasso a larger tunning parameter α leads to a more parsimonious model.
The following figure shows an example of this process for the Ridge model and 12h horizon. While the value of α that minimizes the cross-validated MAE is 0.010, the one standard error rule allows a tuning parameter of α =72.790 (vertical blue dotted line).

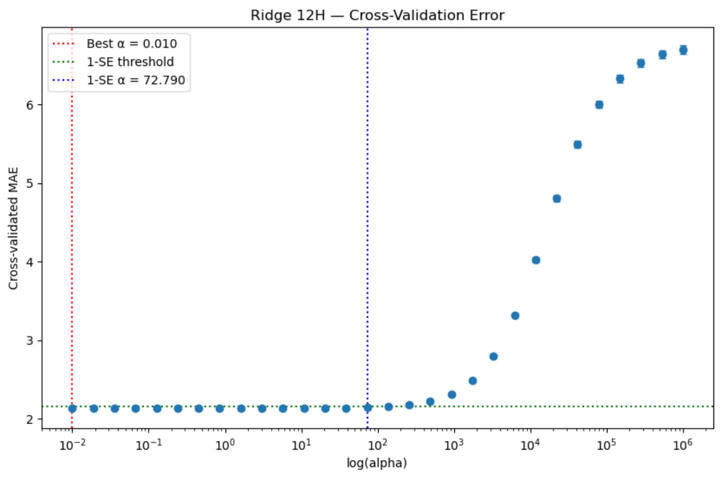


Figure 9: Cross-Validated MAE chart for Ridge Regression 12h horizon

Finally, predictions for Bern temperature, on unseen data, for the three time horizons (12h, 24h, 48h) based on the best α and the 1SE α are performed. The performance for each of the prediction models is evaluated using the MAE.

4.4 Random Forest

The next nonparametric model to be implemented in this project is Random Forest (RF). RF is similar to another approach called Bagging, with the difference that it only uses a random subset of the predictors that are considered at each split of the decision trees. RF is bagging plus "feature randomness". This approach decorrelates the single trees and increases the prediction performance compared to Bagging (Engelke, 2025g, p.3). Therefore, this project does not implement Bagging, but exclusively RF.

4.4.1 Model Overview

Random Forest differs significantly from the previous models in that it is an ensemble method, which, similar to Bagging, trains many independent complete fully grown decision trees b=1,…,B on different bootstrap samples of the data $Z^{*1}, \dots, Z^{*B}$ (Engelke, 2025h, p.3). For the present regression problem, the prediction of the RF corresponds to the average of the predictions of all individual decision trees:

$$\hat{f}_{RF}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

By averaging all these predictions of these trees, the variance of the model is substantially reduced, which leads to a higher prediction performance for complex, nonlinear tasks (Engelke, 2025h, p.5).
In addition, due to the bootstrap sampling, each tree is trained on approximately 63% of the data, which makes it possible to calculate the out-of-bag error estimate corresponding to an internal estimate of the generalization error. This can be used as an alternative to cross validation.

$$Err_{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}_{RF}^{OOB}(x_i)\right)^2$$

where $\hat{f}_{RF}^{OOB}(x_i)$ is the OOB prediction for observation x_i. Calculated as the average prediction over all trees for which observation x_i was not part of the boostrap sample.

$$\hat{f}_{RF}^{OOB}(x_i) = \frac{1}{|I_{x_i}|} \sum_{b \epsilon I_{x_i}} \hat{f}^{*b}(x_i)$$

4.4.2 Model Implementation

In Python, Random Forest is implemented using the *RandomForestRegressor* class from *sklearn.ensemble* module, with out-of-bag estimation enabled (oob_score=True). In contrast to the linear and KNN models, many parameters must be tuned for RF. The modeling implementation of RF consists of three steps. First, a parameter grid i set up, specifying the list of hyperparameters to be tuned. One of the parameters used to tune model is the number of trees in the forest B, which is "is not a classical tuning parameter since large B will not overfit the data; the error will just stabilize" (Engelke, 2025g, p.5). Further tuning parameters are the maximum depth of each tree, the minimum number of samples per leaf necessary to split a node, and the minimum number of samples per leaf. The last tunning parameter is the number of predictors considered at each split (max_features). A common choice for this last parameter is m ≈ p/3 (Engelke, 2025g, p.4), where m defines the number of predictors randomly selected at each split and p is the total number of available predictors.
Next, for each combination of the hyperparameters defined in the grid, the model is fitted on the training data using boostrap samples and the OOB score is calculated. In scikit-learn, the OOB score corresponds to the coefficient of determination (R^2). This differs from the previously mentioned OOB error which is based on the MSE loss. This difference stems from the way scikit-learn works. But the general concept remains the same.
Lastly, grid search is performed by iterating over all the possible combinations of the hyperparameters and retaining the combination that maximize the OOB score. Thus, explicit cross validation is not used, as the OOB score is calculated using observations that are not part of the boostrap samples used in the training. The hyperparameter combination with the highest OOB score is retained and used to fit the RF models for the different time horizons. The highest OOB is retained because R^2 is a performance measure rather than an error metric.
Finally, using the best parameters from grid search for each time horizon, predictions for Bern temperature for the three future time horizons (12h, 24h, and 48h) based on unseen data are calculated. The performance for each of the prediction models is evaluated using the MAE.

4.5 Boosting

In this chapter, another nonparametric model is considered in more detail, namely Boosting.

4.5.1 Model Overview

As the Random Forest algorithm, Boosting uses decision trees. Yet, instead of growing many decorrelated deep trees with high variance and low bias and then averaging them to reduce the variance, Boosting follows a different approach. The Boosting approach consists of sequentially growing shallow flat trees with low variance but high bias (Engelke, 2025i, p.1). The goal of this sequential approach is to reduce the bias, as each new tree is trained to correct the errors made by the previously fitted trees. Even though the two approaches differ substantially, both Boosting and RF still rely on combining many trees to improve the predictive performance. The final model is based not only on one, but on many different trees, which improves performance, but loses interpretability.

First, Boosting starts with an initial model*:* $\hat{f}(x_i) = 0$ and the initial residuals are defined as: $r_i = y_i$. At each iteration $b = 1, \ldots., B$ a flat regression tree $\hat{f}^b(x_i)$ is fitted to the current residuals r_i. The initial model is updated by adding a shrunk version of the new tree: $\hat{f}(x_i) \leftarrow \hat{f}(x_i) + \lambda \hat{f}^b(x_i)$, where λ is a shrinkage parameter that shrinks the contribution of each tree (Engelke, 2025i, p.2). The residuals are updated accordingly: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$. After B iterations the final boosted model, which corresponds to a combination of all successively grown trees, can be represented by:

$$\hat{f}_{Boosting}(x_i) = \sum_{b=1}^{B} \lambda \hat{f}^b(x_i)$$

The tunning parameters for Boosting are the number of trees B, tree complexity, e.g., number of splits or max depth, and the learning rate λ.

4.5.2 Model Implementation

In Python, Boosting is implemented using *GradientBoostingRegressor* class from the *sklearn.ensemble* module. As with RF, a parameter grid with various tuning parameters is set up to find the Boosting model. Some of the parameters of Random Forest are also used for the boosting algorithm, namely the number of trees B, the maximum depth of each regression tree, the minimum number of samples required to split a node, as well as the minimum number of samples required in a leaf node. In contrast, the learning rate λ is a new parameter included in the grid (Engelke, 2025i, p.2). The steps for applying the grid search to find the best combination of parameters, which are then used to fit the final model to the training data remain the same as with RF. As before R^2 is maximized during grid search, and not the MAE. MAE is only used for the final prediction models evaluation.
Finally, using the best parameters from the grid search for each time horizon, predictions for Bern temperature for the three future time horizons (12h, 24h and 48h) based on unseen data are calculated. The performance for each of the prediction models is evaluated using the MAE.

5 Model Diagnostic

This chapter is about comparing the performance amongst the different models that are applied to predict the temperature in Bern. This comparison uses the MAE, which is useful because it provides an interpretable measure of how many degrees the predictions of the models deviate from the actual temperature. Lower MAE values are better.
The figure below is a heatmap that compares the MAE values of the different prediction models for the 3 different prediction horizons. Test and Train MAEs are calculated. A clear general trend emerges showing that the MAE increases as the prediction horizon increases. This is expected, as it is harder to predict longer term forecasts, e.g., 48 hours ahead, compared to short-term forecast, e.g., 12 hours ahead, due to the inherent uncertainty of many weather variables.

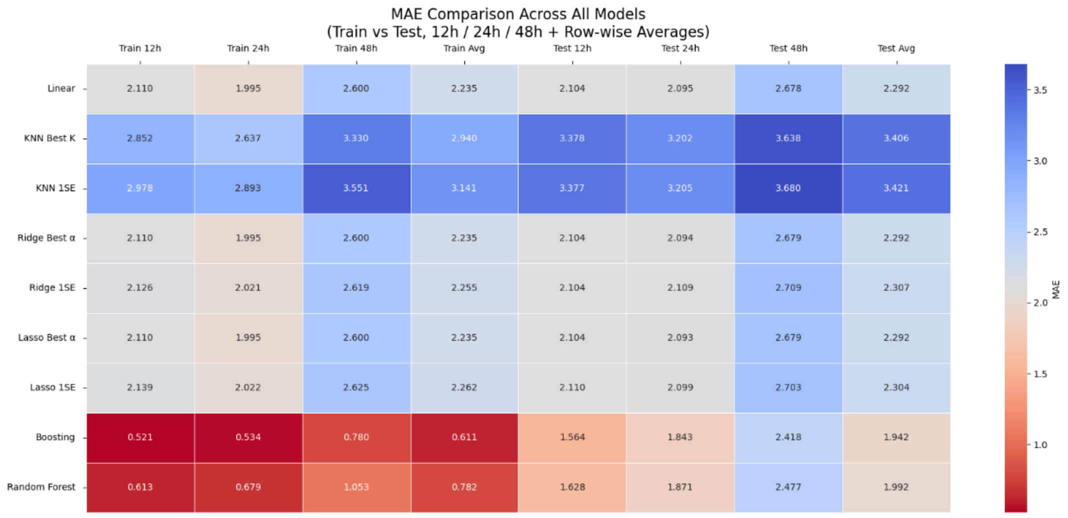


Figure 10: MAE Heatmap by prediction models and horizon

When inspecting the train and test MAE of the Linear Regression models, it can be seen that all MAE values are relatively similar for all time horizons and regularization techniques such as Ridge and Lasso. This becomes especially clear when comparing these Linear Regression models to the tree-based methods such as Random Forest and Boosting. This pattern possibly indicates underfitting, i.e., high bias and low variance, indicating that linear models are too simple to capture the complex relationship between the temperature forecast and the present meteorological variables.

Furthermore, the heatmap for Linear Regression model illustrates a surprising performance pattern where the MAE decreases from the 12h to 24h prediction horizon before increasing again for the 48h prediction horizon. This may indicate the presence of a structure in the data that the linear models are not able to capture. Another observation is that the test 12h MAE for linear regression is lower than the train 12h MAE, e.g., 'Linear': 2.104 vs. 2.110. This might be explained by the test set which contains patterns that are easier to model than those in the train set for this time horizon.

Ridge and Lasso models for 'Best α' and '1SE', yield similar MAE results to the plain Linear Regression model, which suggests that the regularization/shrinkage penalty is not significantly effective for this weather data set and does not improve performance. Furthermore, the '1SE' models, which deliberately trade a small amount of accuracy for increased robustness, perform only slightly worse than the 'Best α' models. This makes them a viable option when prioritizing simpler and more robust linear models.

While a larger gap between train and test error exists for the KNN models, for example: 'KNN Best K' with a train MAE of 2.852 vs. a test MAE of 3.378 for the 12h horizon. The overall train and test performance of KNN models remains poor relative to the other models. KNN models, especially 'KNN Best K' has significantly higher MAE values in comparison to all the other parametric and non-parametric models. This poor performance is likely due to the "curse of dimensionality," where the high dimensional feature space of meteorological variables renders the distance to the nearest neighbors less meaningful and thus, less effective for prediction. Similar to the linear models, the '1SE' KNN performs worse than the 'Best K', reflecting the trade-off between robustness and predictive accuracy.

When inspecting the tree-based ensemble methods, it can clearly be stated that these models show the lowest MAE values on the train and test set compared to the rest of the models. For example, Boosting has a train MAE of 0.521 and RF has a train MAE of 0.613 for the 12h horizon. However, the significant gap observed between train and test MAE for both models provides evidence of overfitting and thus high variance. For example, Boosting has a train MAE of 0.521 vs. a test MAE of 1.564 for the 12h horizon.

The high model complexity present both in Boosting and RF allows them to capture complex patterns in the data at hand but also leads to fitting noise in the training set which does not generalize well to unseen data from the test set.
While both RF and Boosting perform better than all the other models, Boosting consistently outperforms RF. This suggests that the sequential nature of Boosting, lead to a successful bias reduction and superior predictive accuracy compared to RF’s parallel aggregation strategy.
This performance patterns described in this chapter are also seen during the Kaggle competition, where the Boosting algorithm performed best with an MAE of 1.805, followed by RF, linear models, and KNN at last.

6 Model Comparison and Recommendation

After a comparison of the various models by their performance metrics in the previous chapter, this chapter uses these results as well as further evaluation criterions such as the interpretability and context to compare the different modes and explain why this project recommends the Boosting algorithm implemented as the final model for prediction.
As it was seen in Chapter 5, Boosting consistently achieved the lowest MAE on the unseen test set across all three time horizons of 12h (MAE: 1.564), 24h (MAE: 1.843), 48h (MAE: 2.418) and an overall average MAE of 1.942, demonstrating its superior predictive performance. The reason for this superiority lies in the sequential approach of the Boosting algorithm. While RF and Boosting both outperformed linear and KNN models, Boosting grows trees sequentially, with each new tree trained to correct the residual errors of the previous models. This leads to a significant reduction in bias and superior accuracy compared to RF’s parallel aggregation approach.
However, one trade-off to be considered for Boosting is its high degree of overfitting and high variance. Nevertheless, its test MAE remains the lowest, declaring it as the best performing model for generalization in terms of error magnitude this project explored.
Boosting is inherently complex due to its nature of using a vast amount of decision trees in order to make predictions. This complexity leads to loss of interpretability and makes it difficult to attribute predictions to a specific input feature of the dataset. Despite this, the project considers the lack of interpretability an acceptable loss in light of the superior performance gained.
Another reason to support Boosting recommendation comes from the nature of the data. Boosting has minimal preprocessing requirements due to the fact that, as it can be seen in section 3.5 of this project, the algorithm is not sensitive to feature scaling, outliers, or specific distributions of the data.
In contrast, linear models require many more preprocessing steps which would make the deployment of the final model more complex. Moreover, the flexibility offered by tree-based ensemble models allows Boosting to capture complex relationships, such as nonlinear relationships and intricate feature interactions in the weather dataset.
When comparing Boosting to KNN, the Boosting algorithm handles the high-dimensional feature space of the dataset (train.csv and test.csv) well, in contrast to KNN, where the model performance deteriorates in high dimensional feature spaces due to the “curse of dimensionality”. Boosting also does not rely on distance-based measures.
To conclude, Boosting is the recommended model, accepting lower interpretability in exchange for a higher predictive performance, lower preprocessing requirements, and strong capabilities to model complex and high dimensional data.

7 Conclusion

This project identified and implemented the Boosting algorithm to solve the problem of weather prediction, achieving the lowest overall MAE on the test data across all prediction horizons concluding to an average MAE of 1.942. Possible future paths and improvements to increase predictive performance may be explored. One option is to further investigate the Boosting algorithm by conducting deeper hyperparameter search to reduce overfitting without sacrificing test performance. For this purpose, the regularization parameters for the maximum depth of each regression tree as well as the learning rate λ might be of interest. Another approach is to explore further models. Neural Networks might be able to outperform Boosting by capturing subtle non-linearities and long-term temporal dependencies that the tree-based models implemented in this project still miss. On the other hand, Support Vector Machines for regression could be another option to be followed up on due to their capacity of handling high-dimensional spaces with the use of kernels such as the radial/gauss kernel (Engelke, 2025j, p.3).

9 References

Engelke, S. (2025a). Machine Learning, Training Versus Test Error, University of Geneva, Geneva, Switzerland, pp.1-5.

Engelke, S. (2025b). Machine Learning, Linear Regression, University of Geneva, Geneva, Switzerland, pp.2-8.

Engelke, S. (2025c). Machine Learning, Gradient Descent, University of Geneva, Geneva, Switzerland, pp.1-3.

Engelke, S. (2025d). Machine Learning, Regression Framework, University of Geneva, Geneva, Switzerland, pp.6-7.

Engelke, S. (2025e). Machine Learning, Cross-Validation, University of Geneva, Geneva, Switzerland, pp.1-9.

Engelke, S. (2025f). Machine Learning, Linear Model Selection and Regularization, University of Geneva, Geneva, Switzerland, pp.2-9.

Engelke, S. (2025g). Machine Learning, Random Forest, University of Geneva, Geneva, Switzerland, pp. 3-8.

Engelke, S. (2025h). Machine Learning, Bagging, University of Geneva, Geneva, Switzerland, pp. 3-5.

Engelke, S. (2025i). Machine Learning, Boosting, University of Geneva, Geneva, Switzerland, pp. 1-3.

Engelke, S. (2025j). Machine Learning, Support Vector Machines, University of Geneva, Geneva, Switzerland, p.3.

Näf, J. (2025), Forecasting with Applications in Business, University of Geneva, Switzerland, p.65

11 Appendices

Appendix 1: Use of Generative Artificial Intelligence

Generative artificial intelligence was used during the project as part of EDA and data preprocessing. This approach is intended to allow the comparison of various implementation ideas, as well as recommendations for their implementation. For example, AI for the Kaggle Submission, for the generation of violin and distribution plots, for the within correlation calculation in Chapter 3 as well as for the MAE heatmap in Chapter 5. The models were fitted and trained following the codes from the seminars, part of the course S403011 Machine Learning at the University of Geneva.