university of groningen / faculty of science and engineering

Technical Report: Data-Driven Forecasting in Urban Environments: Predicting NO_2 and O_3 Levels in Utrecht

Machine Learning for Industry Practical

WBAI070-05.2024-2025

Title of Deliverable:	Model development
Version:	03
Group Number:	20
Author(s):	Christian Kobriger (s5232686) Lukasz Sawala (s5173019) Csenge Szőke (s5139090) Aleksandar Todorov (s5100534)
Delivery Date:	28/10/2024
Code Repository	https://github.com/atodorov284/air-quality-forecast
Updates:	Model deployment, API data fetching, UI and parsing implementation.

Contents

Executive Summary

As of the 25th of October, the following objectives were achieved:

- Data collection
- Data analysis
- Data preprocessing and preparation for model selection and training
- Model development, tracking, description, and evaluation
- Model selection
- Applied measures for model interpretability
- Model Deployment through HuggingFace Spaces
- API data fetching with automatic scheduled calls
- Command parsing
- Model monitoring system
- Fully functioning, user-friendly UI with extra features

1 Introduction

1.1 Problem Statement

Air pollution is a significant environmental concern, especially in urban areas, where the high levels of nitrogen dioxide (NO_2) and ozone (O_3) can have a negative impact on human health, the ecosystem and on the overall quality of life (WHO, 2024). Given these risks, monitoring and forecasting the level of air pollution is an important task in order to allow for timely actions to reduce the harmful effects.

In the Netherlands, cities like Utrecht experience challenges concerning air quality due to urbanization, transportation, and industrial activities. Developing a system that can provide accurate and robust real-time air quality monitoring and reliable forecasts for future pollution levels would allow authorities and residents to take preventive measures and adjust their future activities based on expected air quality. This project focuses on the time-series forecasting of air pollution levels, specifically NO_2 and O_3 concentrations, for the next three days. This task can be framed as a regression problem, where the goal is to predict continuous values based on historical environmental data. Moreover, it provides infrastructure for real-time prediction, based on recent measurements.

1.2 Background

Air pollution is recognized as a critical issue affecting public health and climate. Nitrogen dioxide and ozone are two key pollutants that contribute significantly to poor air quality, especially in urban environments. The pollutant NO_2 is primarily produced by road traffic and industrial emissions, while O_3 forms through chemical reactions between other pollutants in the presence of sunlight. Prolonged exposure to these pollutants can lead to respiratory diseases, cardiovascular issues, and reduced life expectancy, as noted by the World Health Organization (WHO, 2024).

In order to decrease the impacts of air pollution, the accurate monitoring and forecasting of pollutants is needed. Traditional air quality monitoring relies on ground-based stations that collect real-time data, but these stations often have spatial limitations and can be expensive to maintain. However, recent advancements in technology have led to the development of more sophisticated monitoring and forecasting systems. One such system is the Internet of Things (IoT), which utilizes low-cost sensors and real-time data transmission over vast areas. IoT systems, consisting of interconnected sensors, communication networks, and cloud-based platforms, provide continuous data collection (Toma et al., 2019). However, the vast datasets generated by IoT require advanced analysis to identify meaningful pollution trends and interactions. Machine learning (ML) techniques provide a solution.

The availability of large-scale data through IoT devices paved the way for the application of ML techniques in air quality forecasting. Machine learning models can analyze complex, non-linear relationships between environmental factors and pollutant levels, thus offering more accurate and dynamic predictions compared to traditional methods (Gangwar et al., 2023). Several approaches have been developed:

- Linear Regression: While it is a classical statistical model and is fairly easy to implement, linear models may struggle with the non-linearity in air pollution data and have known drawbacks in the field (Das et al., 2022).

- Support Vector Regression (SVR) and Random Forest Regression (RFR): More advanced algorithms like SVR and RFR have been used to capture the complex patterns in air quality data. Srivastava et al. (2020) used these methods in an IoT-based system to predict pollution levels and trigger alerts when necessary.

- Neural Networks (NN): Neural networks, particularly deep learning models, have also been successfully applied to air quality forecasting in smart cities. These models excel at recognizing complex patterns in large datasets, and studies like Kök et al. (2017) demonstrate the potential of neural networks for high-accuracy air pollution predictions.

In summary, the integration of machine learning models is transforming air quality monitoring and forecasting. These models provide more accurate, scalable, and real-time solutions to address the challenges of urban air pollution.

1.3 Project Objectives

The primary goal of this project is to create a data-driven air pollution monitoring and forecasting system for Utrecht. The system's aim is to predict NO_2 and O_3 concentrations for the next three days (based on previous measurements), using historical and real-time data. To reach this final goal, intermediate steps must be taken, such as: data and feature engineering, model development and evalutaion and model deployment. The respective subgoals of these steps are described in Table 1.

1.4 Structure of the Report

The project was split into three main stages, described below. However, the separation between those stages is not strict due to the complexity of the problem and the accepted methodologies in the field. Some of the stages could be revisited in further developmental phases to improve model quality or address potential issues that were missed at first. A brief overview can be seen in Table 1. Each of those is rigorously described in the next sections of this report to ensure replicability and reproducibility.

Stage	**Brief Description**
Data Engineering and Preprocessing	Collect high-quality data, analyze data patterns, outliers, and anomalies. Select relevant features and build code as infrastructure (pipelines). Transform, split, and scale the data so it is ready to be used by the model.
Model Training and Evaluation	Select potential candidates for the model, run and record experiments, and perform cross-validation and hyperparameter tuning on the training and validation sets. Evaluate the model on the test set using a suitable metric. Report the results of the training and evaluation process.
Deployment and Integration	Deploy the model on a cloud server and ensure proper infrastructure and workflows for continuous development and integration.

Table 1: The three developmental stages of the project, without a strict separation in between each of them. At any point, the team can revisit their previous methodology.

2 Data Collection and Preprocessing

This section of the report discusses the data engineering stage in detail, as described in Table 1.

2.1 Chapter Highlights

The highlights of this chapter include discussing the collection of data and the merging of two datasets. The process of extracting the valuable features is described, including removing and adding features based on inspection of correlation and domain knowledge. The distribution of the selected features is graphed and the issue of missing values is discussed. Lastly, the structure of a datapoint in the preprocessed dataset is described.

2.2 Data Sources and Format

The data obtained for training the model consists of datasets from two sources, a private meteorological company (VisualCrossing) and a non-profit research project The World Air Quality Index. The data was collected daily, between the 29th of January 2014 and the 11th of September 2024, constituting 3870 days.

As the World Air Quality Index is a non-profit scientific project, the use of their data for research purposes is encouraged and free. On the other hand, VisualCrossing's data is licensed, so after paying a small amount and agreeing to the company's terms and conditions, we obtained the dataset requested from the query. The dataflow from the APIs is highly reliable and has not posed issues or thrown exceptions.

The data from the private company is collected from De Bilt (close to Utrecht) weather station, while the World Air Quality Index gathers the data from the Griftpark station (in the center of Utrecht). The two stations are 4 kilometers apart and the neighbor and urban infrastructure is relatively similar. We therefore assume from now on that the two stations reflect more or less the same air quality conditions, and can be combined without loss of generality.

The datasets have been downloaded in a comma-separated values format and are accessible under our project repository. The human-readable format allows us to access the data more easily. However, no manual modifications have been done to ensure reproducibility.

A description of the two datasets from VisualCrossing and The World Air Quality Index project can be found in Table 2 and Table 3, respectively.

2.3 Data Analysis and Engineering

The two datasets were combined (merged) based on the date each measure was taken. This allowed us to start exploring the patterns in the data. Due to the complexity that 40 variables offer, feature selection was performed prior to going deeper into exploratory data analysis. A thorough analysis of such a huge number of variables is costly, therefore, we wanted to reduce the complexity before continuing with modeling.

2.3.1 Feature engineering

The high dimensionality of the features invites potential under- or overfitting to our model, hence a significant effort was put into extracting valuable features and information.

Several variables had incompatible formats, were duplicated, or were rephrasing other numerical variables. Namely, the following features were immediately dropped:

- Datetime – this column was duplicated after the merging of the datasets.

Variable	Description
Temperature Data	Minimum, maximum, and average temperature
Relative Temperature	Minimum, maximum, and average temperature felt
Humidity and Dew Point	Averaged humidity and dew point
Precipitation Data	Probability, average, coverage, snow level, snow depth, and type of precipitation
Wind Data	Average speed, strongest gust, and direction of the wind
Sea Level Pressure	Atmospheric pressure measured at sea level
Visibility and Cloud Cover	Average visibility in km and cloud cover in %
Radiation Data	Average solar radiation, energy, and UV-index
Severe Weather Risk	Severe risk indicator
Dayphase Data	Time of sunset, sunrise, phase of the moon
General Conditions	Textual description of general weather conditions

Table 2: A brief description of the variables of the data acquired from VisualCrossing. The metrics of the measurement can be found on https://www.visualcrossing.com/resources/documentation/weather-data/weather-data-documentation/. Each data point also contained the date it was measured.

Variable	Description
$PM_{2.5}$	Particulate matter 2.5 micrometers or less in diameter
PM_{10}	Particulate matter 10 micrometers or less in diameter
O_3	Ozone level
NO_2	Nitrogen Dioxide level
SO_2	Sulfur Dioxide level

Table 3: A brief description of the dataset variables, acquired from The World Air Quality Index Project. All variables values are in $\mu g/m^3$. Each data point also contained the date it was measured.

- Sunrise and sunset time – a feature in the `HH:MM:SS` format, which we also deemed not to have any potential benefit for the model
- Precipitation type – rain, snow, or both. Instead of discretizing this feature, it could be modeled by other features (e.g. precipitation and temperature).
- Descriptive text data (condition description, icon, name of the station of measurement) – those were mostly for weather report purposes and had no effect on O_3 or NO_2.

A quick reformatting to remove whitespace and special characters in the column names was performed to ensure consistent access in the later stages. This was likely an issue because of the data sources and the merging process.

Next, a missing value analysis was performed. All the missing values occurred completely at random, which was concluded after a moving average analysis, perhaps due to sensor issues or incorrect/removed measurements. The associated ratio of missing values can be seen in Table 4.

Variable	$PM_{2.5}$	PM_{10}	O_3	NO_2	SO_2	Visibility	Severe Risk
Missing Values (%)	9.62%	0.79%	4.14%	0.71%	92.70%	0.46%	73.41%

Table 4: Missing values and their percentages

Missing values are the only anomalies that can be observed in our data.

Given the high proportion of missing values in `Severe Risk` and SO_2, these variables were excluded from further analysis to avoid introducing bias into the result.

Going further, a correlation analysis was performed to select relevant features. We calculate the Pearson-Correlation coefficient r between two variables random X and Y as

$$r = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y},$$

where $\text{cov}(X,Y)$ is the sample covariance between the two variables and σ_X and σ_Y are their sample standard deviations, respectively. This coefficient was calculated using the `pandas` Python library for all the 29 remaining variables in the dataset. This results in a correlation matrix, which is symmetric and has only ones along the off-diagonal (due to autocorrelation always being 1). We set an arbitrary threshold of $|r| > 0.3$ with our predictor variables (O_3 and NO_2) if a variable should be kept. This threshold is subject to revisit in the future.

Moreover, mutually correlated selected predictor variables with a correlation $|r| > 0.7$ were dropped, as in general, models perform better with non-related predictor features. Nonetheless, this threshold is also subject to change. An upper-triangular correlation map, excluding the autocorrelation, can be seen in Figure 1. Note that only correlations $|r| > 0.3$ are displayed for clarity.

At this stage, the selected variables were $PM_{2.5}$, PM_{10}, O_3, NO_2, average temperature, humidity, visibility, and solar radiation. As a last step of feature engineering, we dug into the literature on air pollution to determine whether domain knowledge could benefit our model. We decided to reintroduce precipitation, wind speed, and wind direction, as those have been empirically observed to have an effect on ozone and nitrogen dioxide formation and could have non-linear relationships with them. The exact reasons for their incorporation can be seen in Table 5.

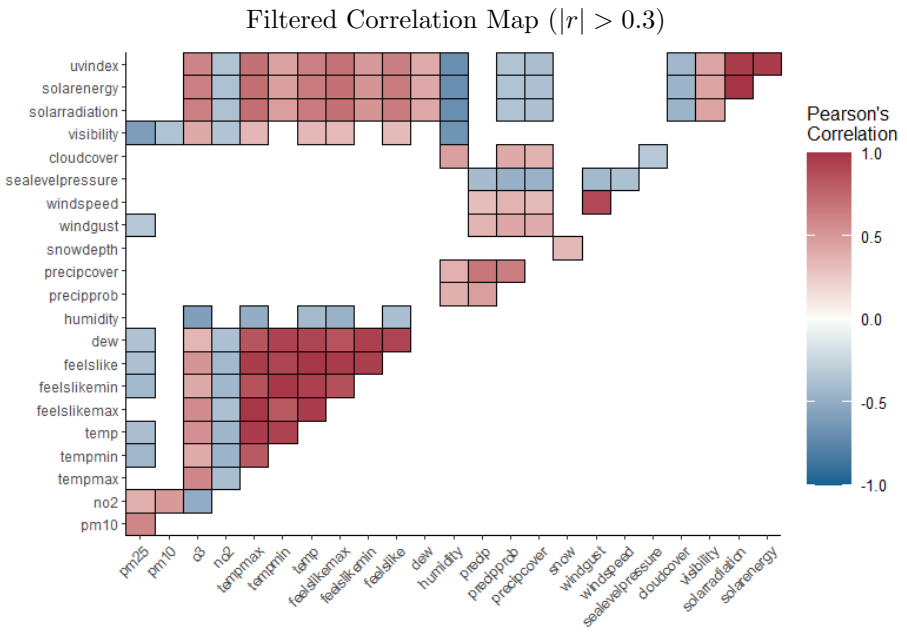

Figure 1: The correlation matrix of all variables in the joint dataset. Only tiles with significant correlation ($|r| > 0.3$) are displayed. An upper-triangular form is preferred due to the huge number of variables.

Removed Feature	**Domain Knowledge**
Precipitation	High/extreme rain tends to wash out NO_2 and other particles considered (e.g. PM_{10}), leading to lower levels in the following days (Sun et al., 2019; Tian et al., 2021)
Wind speed	High/extreme wind speeds tend to increase the variance of the measurements of air particles, mainly O_3, PM_{10}, and $PM_{2.5}$ (Li et al., 2014; Xie et al., 2022; Yassin, 2013)
Wind direction	Wind blowing from highly industrialized areas can have impacts on air quality compared to wind from the sea area (Xie et al., 2022; Yassin, 2013)

Table 5: Motivation for reintroducing some of the features after exploring the literature on air pollution.

Taking into consideration all previous steps, we keep 11 variables from the initial list of 39 (not counting the date). To reiterate, the dropped variables either exhibited an extreme ratio of missing values, were not related to our predictor variables, or were mutually correlated with each other. With this, we conclude the feature selection section.

2.3.2 Data Analysis

The reduction of the number of variables in the dataset allowed us to perform extensive exploratory data analysis and identify key patterns and structures. A table of the descriptive statistics can be seen in Table 6. The data distribution can be seen in Figure 2.

The data is relatively well-behaved, with clear patterns for either normal, uniform, or exponential (right-skewed) distributions and no severe outliers. Features that look exponentially distributed could benefit from a log-transform to ensure normality (discussed in the next section). From the descriptive statistics, we observe that $PM_{2.5}$, PM_{10}, O_3, NO_2, and visibility have fewer available values than the rest – those features have missing values and they are missing completely at random. This will also be addressed in the following section.

Furthermore, all features seem not to vary a lot, except solar radiation and wind direction. However, this is to be expected as the values for solar radiation fluctuate considerably during the four seasons of the year. Regarding wind direction, even though there is a great amount of variance, there seems to be some kind of a pattern, perhaps due to Utrecht's proximity to the sea.

Lastly, to see if potentially a model could predict ozone and nitrogen dioxide, based on the previous day values, a correlation analysis was performed based on the values from the day before the measurements were taken (predictor features with lag 1). The results can be found in Table 7. This provides a promising direction of developing the model for prediction, as the features from the previous day seem to be related and could be used to predict ozone and nitrogen dioxide for the following day.

	$PM_{2.5}$	PM_{10}	O_3	NO_2	Temp	Hum	Visib	SR	Precip	Wind Sp	Wind Dir
Count	3317	3641	3518	3644	3670	3670	3653	3670	3670	3670	3670
Mean	39.54	17.03	25.83	7.73	11.35	79.68	22.95	148.23	2.30	20.59	189.74
SD	21.70	7.96	11.70	4.86	6.06	10.20	10.03	100.52	4.66	7.59	90.54
Min	1	5	1	1	-8.3	36.4	0.1	1.1	0	7.2	0.1
Max	161	86	107	32	29.6	99.8	46	360	48.9	64.8	359.8

Table 6: Summary and descriptive statistics of the remaining features.

2.3.3 Pre-processing

As previously mentioned, several of the variables ($PM_{2.5}$, PM_{10}, O_3, NO_2, and visibility) have missing values, two of which are the target predicted variables. This poses an issue, both from an engineering and business perspective.

Currently, in the field, there are two main ways of handling missing values – either modifying the data (dropping or inputting them) or reducing the space of potential models to ones that can handle missing values (usually tree-based, gradient-boosting, and ensemble approaches).

To further examine why the inputting approach is less preferred in the current scenario, we need to examine the mathematical perspective of the variables and the final goal of this project. A naive way is to input the mean of the feature to replace the missing values. However, this poses serious issues as it adds both bias and variance to the joint distribution under which the dataset was generated. More advanced techniques such as linear interpolation could also be used. Nonetheless, since the missing

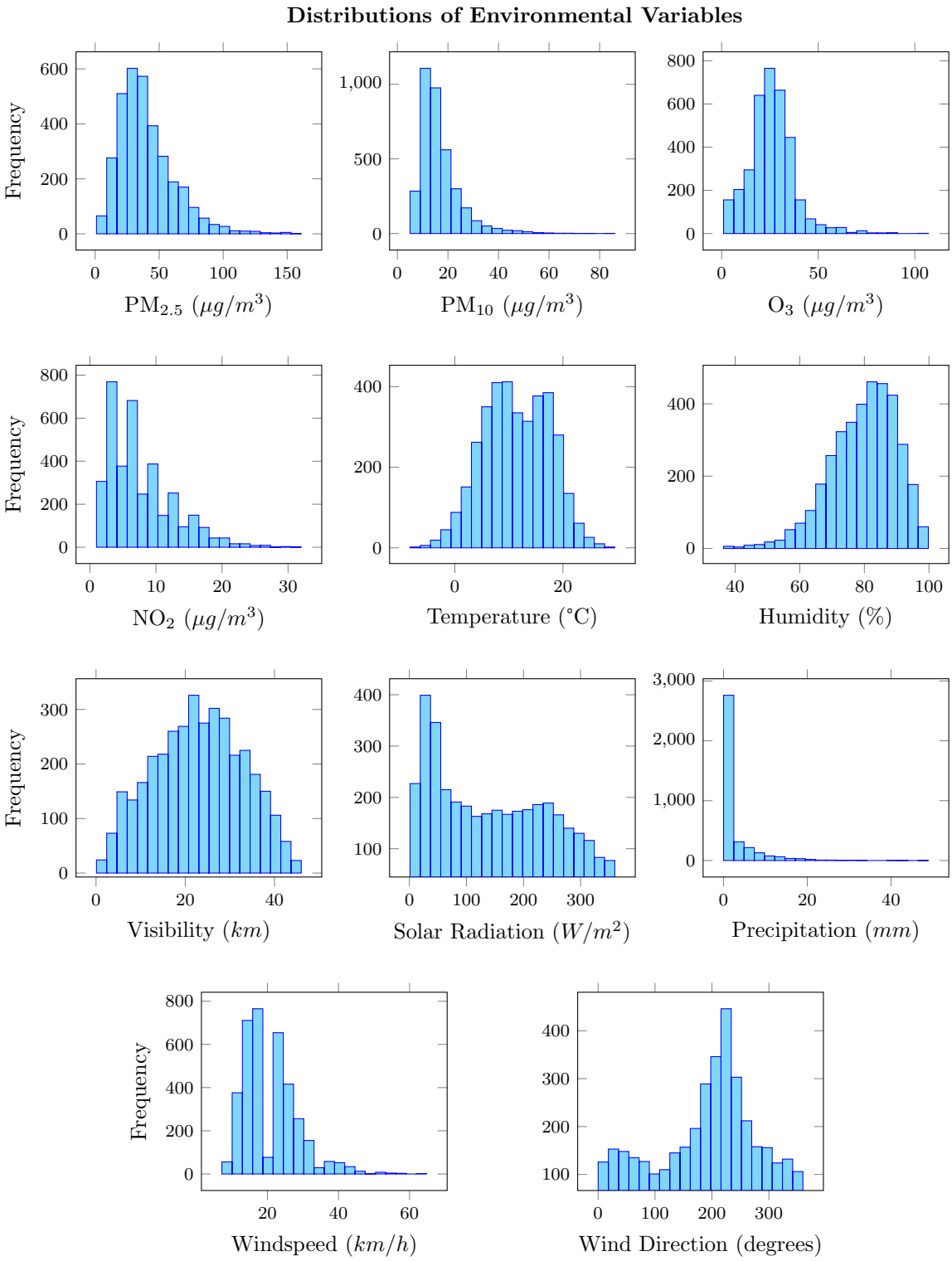

Figure 2: Inspection of the distributions of selected variables after the feature engineering stage. Right-skewed variables could benefit from a log-transform to enforce normality, however, at this stage has not been performed.

Previous Day Variable	*r* **to O_3**	*r* **to NO_2**
$PM_{2.5}$	-0.22	0.30
PM_{10}	-0.15	0.34
O_3	0.77	-0.43
NO_2	-0.45	0.68
Temperature	0.51	-0.44
Humidity	-0.52	0.29
Visibility	0.37	-0.29
Solar Radiation	0.61	-0.41
Precipitation	-0.05	-0.05
Wind Speed	-0.02	-0.06
Wind Direction	-0.03	-0.06

Table 7: Correlation matrix of O_3, NO_2 with previous day variables.

values occurred completely at random, a missing value may be encountered during inference time. This could be a significant issue of a model that cannot handle missing values, and inputting them in real time seems infeasible. We want to design a robust product, being able to predict reliably not relying on the validity of the data all the time – if a sensor malfunctions, we still want to make a prediction.

Since O_3 and NO_2 are the predicted variables, filling them with any type of value that is not accurately measured introduces researcher's bias. Hence, all rows that had missing oxygen and nitrogen dioxide were removed to ensure proper validation of our target feature. This decision was further motivated by the fact that O_3 had missing values for three months straight. This resulted in the removal of 161 rows from the dataset.

The rest of the features that had missing values, namely $PM_{2.5}$, PM_{10}, and visibility were left untouched due to the aforementioned reasons. Missing values are an integral part of any weather measurement dataset and a model that can handle them is preferred. Even if we inputted the missing values in our dataset, it is fairly likely that a feature that did not have missing inputs in the dataset could be missing during inference time. In this case, the system would fail until the researchers find a way of dealing with it, which we prevent by developing a robust model, handling missing values.

Needless to say, this step could always be revisited at any time, should it be the case that a better way of addressing the issue is developed in the future. However, for now, we believe this is the best way to minimize the bias of the model and increase its robustness and quality.

As observed during data analysis, several variables could benefit from a transformation to assume normality. However, this was not performed yet at this stage, as we would like to examine the model's performance on less preprocessed data. This decision will definitely be revisited in the future. Nevertheless, in this initial stage of preprocessing a good rule of thumb is not to modify the data too much without an actual benchmark.

As the last part of the preprocessing section, we transform the dataset by lagging it three days in the past and three days in the future. Since we are predicting the ozone and nitrogen dioxide for the following three days ($3 \times 2 = 6$ predictions), based on all the features from the last three days ($3 \times 11 = 33$ predictors). An example of lagging the data is presented in Table 8.

Date	**Predictors** (X)	**Target** (Y)
Today's date	All $3 \times 11 = 33$ features from the last three days	O_3 and NO_2 for today, tomorrow, and the day after that

Table 8: Example entry in the dataset

2.3.4 Preparation for Model Training

A 70-30 train-test split was performed using the random seed 4242 (for reproducibility). The subsection for the training set was further subdivided to include a validation set, which will be elaborated on in section 3.5 Because not all variables were normally distributed, a MinMax[1] scaler was used on the predictor matrices, using the train set statistic to avoid leakage. The dataset was not shuffled to preserve temporal dependencies. The date range of the split can be seen in Table 9.

Set	**Start Date**	**End Date**	**Length (Days)**
Train Set	01/02/2014	21/06/2021	2452
Test Set	22/06/2021	09/09/2024	1052

Table 9: Date Ranges and Lengths of Train and Test Sets

3 Model Training and Evaluation

This section of the report discusses the development, training, and evaluation of the models, as described in Table 1.

3.1 Chapter Highlights

This chapter starts by describing and motivating three models developed in the project. Then, it defines how their predictions can be interpreted, and how their training process is tracked. The training process and the cross-validation scheme is explained. Bayesian search, which is a method we used for hyperparameter tuning, is motivated and presented. Lastly, we define the metrics used in training and evaluation and motivate the choice of the model to be deployed.

3.2 Model Selection

The decision not to handle missing values restricts the space of our available models, however, allows for more robust inference during deployment. Tree-based methods allow us to compensate for the missing values both during training and inference. Therefore, a decision tree was chosen as a baseline, as it remains the simplest tree-based model while still being able to handle missing values. The following sections give a brief overview of the chosen models. Note that the reader is expected to be already familiar with the inner workings of the models after having taken a basic machine learning course. This is a high-level overview and this report focuses on the engineering side of model development.

3.2.1 Decision Tree Regression

This model serves as a baseline against which the other two models will be evaluated. This rather simple model splits the data into smaller subsets based on feature thresholds, forming a tree-like structure. At each node, the algorithm selects the best feature and value to split on in order to minimize the prediction error. This model was chosen as a baseline for two main reasons – simplicity and ease of interpretability.

[1] https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.MinMaxScaler.html

However, a significant downside is the lack of generalization, as decision trees overfit easily and can produce the same regression values for different inputs.

3.2.2 Random Forest Regression

The natural extension of decision trees is a random forest, which is an ensemble method consisting of multiple different decision trees working together in parallel, and casting the output through a voting system. This method has empirically been shown to reduce overfitting and improve performance, even though it is slightly harder to train and hypertune.

Moreover, the ensemble nature also makes the model more robust to noise and outliers which is an important factor for model selection. In other words, random forests are able to capture all the complexity and advantages of decision trees, while providing some extra benefits. This can be useful, for example, whenever an outlier (e.g. New Year's Eve or Kingsday) is present, as the random forest likely has a sophisticated tree (or trees) for those days that can track the buildup from the previous days (e.g. some people test their fireworks on the 29th and 30th of December, hence should detect the spike at New Year's Eve).

3.2.3 XGBoost

Another natural extension of the basic tree methods is gradient boosting, which builds sequential tree regressors, and each one is fitted on the errors (residuals) of the previous one. The Extreme Gradient Boosting (XGBoost) method was chosen over other gradient boosting methods due to its good documentation in Python as it has its own separate library (`xgboost`) and ability to be partially parallelized when ran on the GPU[2]. This can be quite useful for scaling purposes.

XGBoost also implements a plethora of regularization techniques, such as tree pruning, L_1, and L_2 regularization, so it is robust to outliers. This, compared with its efficient memory usage allows for handling large datasets and complex models more systematically than random forests and decision trees.

3.3 Model Interpretability

An important aspect of all the models used is their ability to provide feature importance scores, which we equate with interpretability. These scores show which features are the most influential in predicting O_3 and NO_2. The insight gained from extracting the feature importance scores can be valuable if retraining or optimizing according to different business needs. For instance, less important features could be removed so that the model is faster and fewer measurements need to be taken, or focus could be put into sensors that track valuable features. Lastly, these scores allow to explicitly determine how the model makes its predictions.

There are several ways to extract feature importance scores for all proposed models. However, due to the complexity of the dataset and the number of feature interactions to be analyzed, we decided to use SHAP values, which directly determine the influence of each feature on the prediction (Lundberg & Lee, 2017).

The Shapley Additive Explanations (SHAP) evaluation model is widely recognized in interpreting complex tree-based models due to its accuracy (calculating a per-feature score is valid even when considering predictions based on multiple features) and consistency (minor changes influencing the process of calculating the score by i.e. data perturbation are reflected through small changes in the score given). Furthermore, SHAP-based scores are easy to analyze and interpret, which is desired for clarity purposes. The SHAP scores determine a direct link between the feature value and its effect on the prediction.

[2]The sequence of decision trees is executed sequentially, however, each decision tree's branches are built and executed in parallel.

The inner workings behind the calculation of the SHAP score are beyond the scope of this report. Readers are expected to familiarize themselves first if they would like to do a critical analysis (Lundberg & Lee, 2017). Otherwise, it is expected that the reader understands that the mean absolute SHAP score is directly proportional to the feature importance.

3.4 Experiment Tracking

All executed experiments were tracked through the Python library `MLFlow`. It implements an interactive dashboard, in which system metrics (CPU/GPU/RAM usage), training metrics (scores and evaluation metrics), test metrics, hyperparameters, Pickle files for saved models, requirements, and many more are automatically logged. This library combines very well with the chosen models as it has support for automatic experiment tracking for decision trees, random forests, and XGBoost.

The code infrastructure we provide offers a way to automatically start an MLFlow server on your device without needing additional setup. Check the code documentation and `README` for more details on how to see the experiments conducted by the team.

3.5 Timeseries Validation Scheme

The train dataset from the preprocessing pipeline was further divided into a train and a validation split (`n_samples // (k + 1)` $\approx 16\%$ of original train set) through the timeseries k-fold cross-validation scheme. A default value for k is 5, which we did not further change. This makes each fold contain 408 samples.

The timeseries validation scheme provides a way to split time series, observed at fixed time intervals into train and validation sets, so that temporal patterns are preserved. In each split, validation indices must be higher than before. This cross-validation object is a variation of classical k-fold cross-validation. In the k_{th} split, it returns first k folds as the train set and the $(k+1)_{\text{th}}$ fold as the validation set. Note that unlike standard cross-validation methods, successive training sets are supersets of those that come before them.

Note that future (validation) data is never exposed to the model. The reader is further expected to be familiar with different cross-validation schemes, their usefulness, and how predictions are evaluated [3]. This validation split can be seen in Figure 3.

3.6 Hyperparameter Tuning and Optimization

Hyperparameter tuning is a notoriously difficult task even for the simplest models, as it requires domain knowledge, good intuitions, and sometimes even a brute-force approach. Hyperparameter tuning was done based on the cross-validation scheme seen in the previous section.

All models with their respective hyperparameter configurations are trained on the train set, and the model with the best metric (mean squared error, discussed in the next section) is selected to be further evaluated on the held-out test set from the preprocessing section, which has never been seen by the model.

Bayesian search introduces an informed and efficient approach to hyperparameter optimization. Unlike grid and random search, which evaluate hyperparameter combinations without considering past evaluations, Bayesian search builds a probabilistic model of the objective function, typically using a Gaussian process. This allows Bayesian search to predict and sample from promising regions of the hyperparameter space based on prior evaluations, focusing the search where high-performing hyperparameters are likely to be found. By balancing exploration (trying new areas) and exploitation (refining known good

[3] Otherwise, check https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

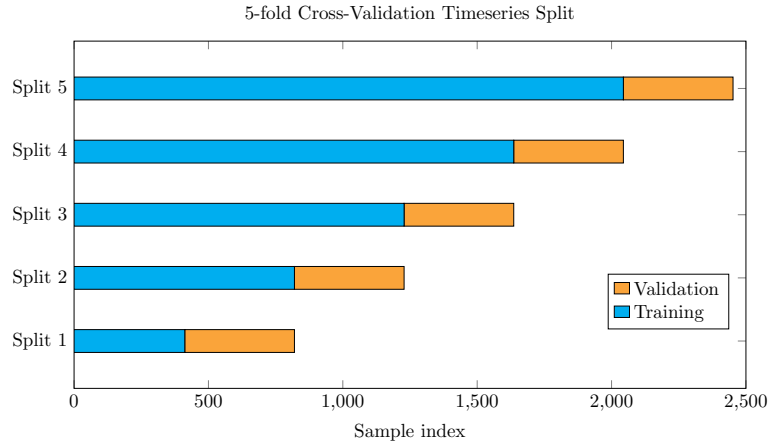

Figure 3: The timeseries 5-fold cross-validation scheme used to train all models. The earliest date is 01/02/2014 and the latest date is 21/06/2021. The sample index corresponds to which day after the start date a measurement was taken.

areas), Bayesian search drastically reduces the number of evaluations needed to find optimal hyperparameters.

Bayesian optimization iteratively updates its model, using the results of each hyperparameter configuration to guide future searches, making it more efficient and targeted than random search, which samples hyperparameters without considering prior outcomes. This informed search process makes Bayesian optimization particularly effective for our task, where the models are relatively fast to train, allowing us to take advantage of this strategy without significant computational costs. The use of an informed search is further motivated by the scarce available literature on predicting air quality through tree-based methods and the lack of experience and intuition within the team. Bayesian search can exhaust a larger hyperparameter space faster and more efficiently than both grid and random search. The hyperparameter ranges, the best hyperparameters, and system training metrics can be found in Appendix A and Appendix B, respectively.

The evaluation was executed on the University of Groningen's computing cluster Habrok, as it provides more access to computational power. This was the main bottleneck of the project and from a business point of view, this is where resources and model quality are proportionally related.

3.7 Model Evaluation

The model evaluation technique in this project was inspired by Oldenburg et al. (2024). Namely, it was split into two parts – training metric, and another separate evaluation metric. For both metrics, a lower value is better and they are not mutually exclusive.

3.7.1 Training Evaluation

The mean squared error (MSE) is a common regression metric and it was used during the training (and hyperparameter optimization). Its main benefit is the penalization of larger errors, as it calculates the average of the squared difference between the model's prediction and the actual values. This characteristic is especially important for air quality prediction because extreme pollution levels (e.g., high concentrations of nitrogen dioxide) can result in significant health risks, which makes the accurate predictions of these outliers critical.

Formally, we can define the mean squared error of a prediction vector $\hat{y} \in \mathbb{R}^N$ and a ground-truth vector

$y \in \mathbb{R}^N$ as

$$\text{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where y_i and $\hat{y}_i$ are the i-th entries of the respective vectors.

This metric was further used to select the best model during the hyperparameter search. In other words, the best decision tree, random forest, and XGBoost models were selected according to this metric.

3.7.2 Final Model Evaluation

For the final model evaluation and selection, the root mean squared error (RMSE) was used. Since the target variables (ozone and nitrogen dioxide) are in $\mu g/m^3$, the RMSE provides a direct comparison between the models' predictions and the ground truth. In other words, if a model's RMSE is 3.00 for a whole month, that means that, on average, for each day the predictions deviated by 3 $\mu g/m^3$. Formally, we can define the RMSE of a prediction vector $\hat{y} \in \mathbb{R}^N$ and a ground-truth vector $y \in \mathbb{R}^N$ as

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2},$$

where y_i and $\hat{y}_i$ are the i-th entries of the respective vectors.

Thus, taking into account its robustness to outliers and direct interpretability, using RMSE allows us to make meaningful comparisons across models and interpret the magnitude of prediction errors in a way that is directly relevant to air quality standards and regulations.

Results are discussed in the next section.

3.8 Selecting a model for Deployment

Based on the evaluation results (as seen in the next section), XGBoost was selected as the model to be deployed. It performs significantly better than the baseline, and better than random forest. As mentioned before, the feature importance of all the models can be extracted (discussed in the results section). The inference time of all models is virtually instant, even for a large prediction such as the held-out test set, on which the final evaluation happened.

Moreover, since XGBoost is partially parallelizable with GPUs, retraining such a model on a larger dataset (e.g. several cities, or a larger time span) could be faster than the random forest and decision tree, which are limited in that direction. Theoretically, decision trees and random forests could also be parallelized. However, the library we used for both of those models, `sklearn`, does not support GPU utilization, as non-deep-learning methods benefit less from parallelization. Regardless, model training and hyperparameter optimization is the main bottleneck of this project, hence business costs could be reduced with a GPU approach.

Finally, the `xgboost` library encapsulates way more functionality than `sklearn`, which mostly provides data-science-oriented solutions. The `xgboost` library also supports Ruby, C, C++, Swift, and many other programming languages. This could prove especially useful when developing a web-based or mobile app for air quality prediction.

4 Model Deployment, Monitoring, and Maintenance

This section of the report discusses the deployment, monitoring, and maintenance methods of the models and explains the final deliverable developed in the project, summarized by the dataflow chart in Figure 4.

4.1 Chapter Highlights

This chapter outlines the deployment, monitoring, and maintenance of the developed air quality prediction system. Key takeaways include the choice of HuggingFace Spaces and Streamlit for deployment, the use of an automated workflow for data updates and prediction, and the integration of a real-time monitoring system. The chapter emphasizes the design of both the user and admin dashboards, ensuring accessibility and effective oversight. Additionally, it highlights the project's decisions regarding data processing times and handling challenges related to real-time data variability and technical constraints.

4.2 Front-End Development

Streamlit is an open-source Python library for the development of web applications only through Python code, avoiding the need for explicit JavaScript, HTML, or CSS knowledge and implementation. Hence, it was the chosen platform for developing the application of the project, as it provides a default user-friendly interface, and more importantly, a variety of deploying strategies, including self-hosting, cloud, and HuggingFace Spaces. **add more (?)**

4.3 Deployment Environment

HuggingFace (HF) Spaces is a community platform, specifically designed for deployment of various machine learning models. It provides a free tier, consisting of 16GB of RAM, 2 CPU cores, and 50GB of disk space, which is suitable for the resource use of this project. At this development stage, it was chosen as the best option, as it does not need any pre-configuration or system requirements to use.

4.4 Streamlit and HuggingFace Integration

HuggingFace Spaces supports direct deployment of Streamlit apps. In other words, the front-end of the project was built using Streamlit and independently deployed on the HuggingFace Spaces domain[4]. It is worth emphasizing that no specific coding was done for HuggingFace Spaces. This majorly simplifies a possible migration of the application to another hosting service.

Moreover, the deployment of the app from Streamlit to HuggingFace Spaces was done automatically, using a scheduled workflow and no manual work had to be done. This is discussed in the next section.

4.5 Automation, Continuous Integration and Deployment

Two GitHub workflows were set in place to ensure proper setup of the deployment environment.

The first, simpler workflow, ensures code style according to PEP8 guidelines, namely through the `flake8` extension. This particular practice maintains consistent code quality. It is triggered on every push of code, with an alert sent to the administrator of the project on failure.

The second workflow is a combination of fetching new pollution and weather data through API calls, using the deployed model to make new predictions, and uploading them to the HuggingFace Space.

The API data points are accessed through unique keys, which were encoded as repository variables, to avoid any potential misuse, as it is a public repository. The data providers were the ones used in the data collection process (WAQI and VisualCrossing), to guarantee consistent format. Collecting the data is free of charge and fully automatic. After each new call, the new data is fed to the model to obtain up-to-date predictions for the next three days. Lastly, both the predictions and the new data are pushed to the HF space, updating the values visible to the user. If either the API call or the deployment to HuggingFace fails, the administrator of the project is alerted via email. It is important to note that the app will not be dysfunctional in case of a workflow fail, but rather it will not have the latest data

[4]https://huggingface.co/spaces/03chrisk/air-quality-forecasting

available.

This workflow is automatically executed every 6 hours, starting at 04:15 GMT+2 every day. It should be noted that any future team working on the project can easily adjust the update frequency to another designated time period. The choice of the timing is also completely arbitrary and was chosen as a starting point. Amendments to the scheduling of this workflow can easily be done through the corresponding `.yaml` file, according to the application's needs and web traffic.

A summary of the automation and data flow can be seen in Figure 4.

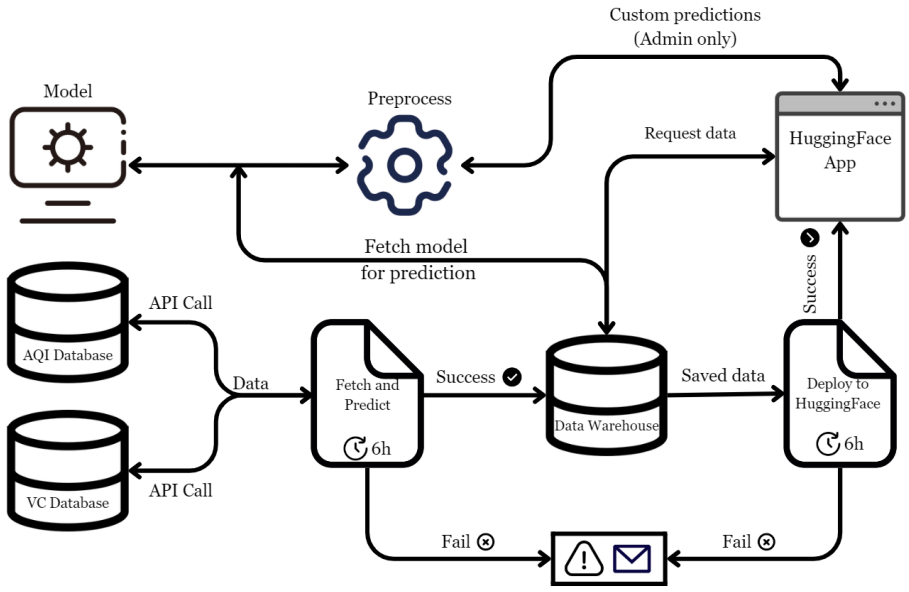

Figure 4: Dataflow chart displaying the structure of the final product. Data is fetched through automated API calls, the model makes a prediction, and pushes it to HuggingFace Spaces, where the app resides. In case of a workflow failure, the project administrator is alerted, whereas the app is still functioning, but will not have the latest data and predictions.

4.6 Data Storage

An important part of the deployment setup is the data warehouse. It keeps every data piece required for realizing the flow, including the models used for inference, historical data, predictions, and monitoring statistics. The data storage is currently done on GitHub, either in a CSV or a JSON format, which is a bad practice in the long-term continuation of this project. This is discussed further in limitations.

The data documentation is extensively outlined in a separate `README.md` file, under the `data` folder in our repository.

4.7 Out-of-Distribution Detection

A simple z-score out-of-distribution (OOD) detection was implemented for all incoming data in the model. The z-score is calculated as

$$z = \frac{X - \mu}{\sigma},$$

where X is the input data in a matrix form, μ is the column-wise data mean, and σ is the column-wise standard deviation. Hence, a separate $z-$score is computed for each data point in the input matrix X. If any specific $z-$score exceeds the value of 3.0 (interpreted as three standard deviations), an alert is

displayed that an input value is potentially out of distribution. Note that this does not stop the user from making a prediction, but rather alerts them that the model's prediction might be inaccurate.

Moreover, this technique only works properly for normally distributed variables, while some of the training variables were not (see the exponentially-distributed variables in Figure 2. Nonetheless, we consider it a good first step for OOD detection, which can and should be extended to account for the actual distributions of the variables.

Note that even in the case of missing data, the model can make a reliable prediction, as intended from the early stages of development.

4.8 User Dashboard

The User Dashboard provides a comprehensive and user-friendly overview of current air pollution levels, health awareness, and recommendations. The dashboard is divided into several sections to present relevant information effectively:

- **Air Quality Monitoring Panel:** Shows the current date alongside real-time pollutant concentrations for (NO_2) and (O_3), compared against WHO guideline limits. Each pollutant's status is color-coded, indicating whether it remains within safe thresholds for user convenience.
- **Air Quality Awareness Section:** Contains expandable informational boxes which offer concise descriptions about the impact of air pollution on health, emphasizing the risks associated with elevated levels of O_3 and NO_2.
- ***Did You Know* Section:** A dedicated area for highlighting quick facts, this section raises awareness about global air quality concerns.
- **Health Recommendations Based on Current Levels:** Provides actionable health guidance based on the present air quality data. When pollutant levels are within safe limits, the dashboard encourages outdoor activities with a reassuring message about air safety.
- **Visualizing Air Quality Predictions:** A dropdown menu enables users to select different visualization types for future air quality predictions. This feature allows users to explore forecasted data in two graphical formats: gauge plots and a line plot.
- **Quiz on Air Pollution:** Presents a question related to the impacts and causes of air pollution, prompting users to consider the broader effects on society through a simple multiple-choice format.

The interface of the user dashboard can be seen in Figure 5.

4.9 Admin Dashboard

The Admin Dashboard provides advanced functionalities for data monitoring, prediction visualization, and model evaluation. This part of the app can be accessed after entering a password[5]. It is organized into multiple sections and includes a dropdown menu for easy navigation between different features:

- **Air Quality Monitoring Panel:** Similar to the User Dashboard, this panel on the left displays the current date and pollutant concentrations (NO_2 and O_3), compared against WHO guideline limits, providing a quick overview of current air quality.
- **Page Selection Dropdown:** The *Select a Page* dropdown offers the following options:

[5] `admin123`

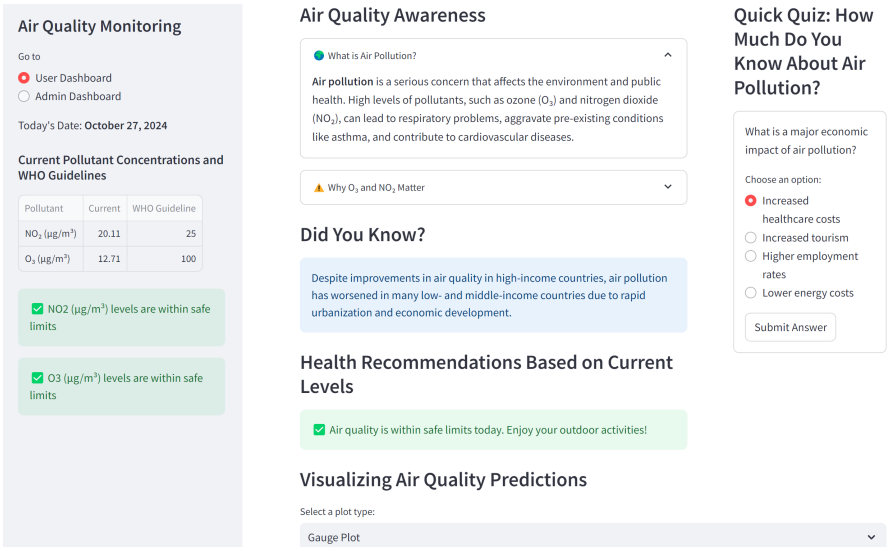

Figure 5: User dashboard of the application

  - **Display Predictions:** Shows model predictions for pollutant levels with visualizations as in the user dashboard. A notification alerts the user if any input features exceed predefined thresholds, indicating potential out-of-distribution data that may affect prediction accuracy.
  - **Make Predictions:** Allows users to upload a custom dataset and generate predictions, provided all required criteria are met. This feature facilitates testing the model with user-supplied data.
  - **Feature Importances:** Displays a feature importance plot generated using SHAP values, derived from the trained XGBoost model. This visualization helps users understand the contributions of individual features to the predictions.
  - **Model Metrics:** Evaluates the performance of the model based on the three most recent predictions using the Root Mean Squared Error (RMSE) metric, allowing users to assess prediction accuracy over recent data.

- **Alerts and Notifications:** The dashboard provides warnings if any (either custom or the actual API calls) input data feature values are outside the expected range, helping users identify potential data issues before interpreting predictions.

The Admin Dashboard is designed to support both data exploration and model performance monitoring, equipping users with tools to make informed assessments of air quality data and prediction accuracy. The interface of the admin dashboard can be seen in Figure 6.

4.10 App Downtime

The app has a scheduled daily downtime between 00:00 and 04:15. During those hours, no predictions are shown to users who enter the website. This was done purposely on the basis of two reasons:

- The measurements obtained through the API calls are averaged on a daily basis. Hence, at midnight, a new API call would be needed to refresh the predictions for the next three days. However, this measurement would only be based on the data from 00:00 and will not be representative of the actual daily values. We wish to wait a small time period so that data for the current day accumulates, hence resulting in a more reliable prediction. To execute this, the next three day predictions

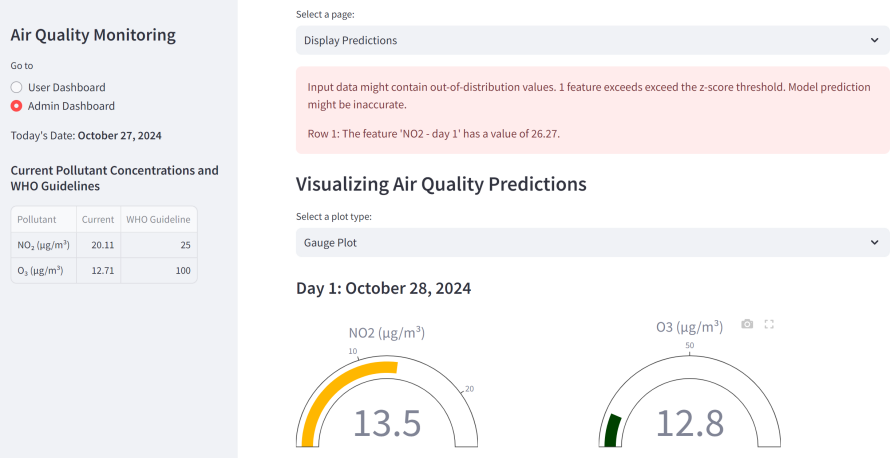

Figure 6: Admin dashboard of the application

are not available between 00:00 and 04:15, and rather an informative message is displayed. The first API call happens at 04:15.

- Regarding sustainability, we expect little to no traffic between 00:00 and 04:15, so operating a prediction service is simply a waste of resources, especially in the early deployment stages of the app.

Lastly, this decision could always be reconsidered and adjusted according to the needs and traffic of the app. There is no motivation to start at *particularly* 04:15 GMT+2 and execute the workflows *every* 6 hours, apart from resource efficiency and not overloading the APIs with requests at this development point.

4.11 Monitoring and Maintenance

The continuous monitoring of the project is split into two parts – tracking the model's predictions, and overseeing the running application and website.

All the model's predictions are saved daily in the data warehouse, hence allowing comparison of the actual values of air pollutants versus the model's forecast. This could be useful, for instance, for the team to decide that the model is not performing as expected, or to provide a comprehensive long-term summary of the model's reliability.

The consistent practices of continuous integration and deployment offer an email alerting system, in case a workflow fails, as mentioned previously. This is crucial as, for example, API keys can get outdated or the APIs themselves could change the way they provide data. Moreover, this allows the project administrator be alert of any potential unplanned downtimes of the running application, in which case a quick fix or a reversion to a previous version would be needed.

4.12 Documentation and Scalability

The codebase contains two independent components, the Streamlit app, and the pipelines used for the data engineering and model training. As the codebase scales, it is essential to have proper documentation of the code to ensure consistency and ease of use. Hence, a full HTML documentation for all functions, classes, and uses of the code was done using `Sphinx`. It can be accessed through the `docs` folder in the repository, which has its own `README.md` for further instructions.

4.13 Retraining Protocol

In the case of retraining, the team wants to offer a consistent and self-contained way of doing so, to avoid any potential deviation from what was originally intended. Hence, a custom-made parser was implemented to enable efficient training, hyperparameter tuning, and evaluation of a different model, specified by the user. The use of the parser for the retraining protocol is exhaustively described in the repository `README.md` and the code documentation.

For ease of use, the only specifications the user needs to provide to retrain the model are datasets and hyperparameter ranges. The protocol will then begin and track an experiment in `MLFlow`, train the model, find the best hyperparameters for the given train set, based on a time-series validation set, and evaluate the model on a held-out test set and save the model, precisely as this project was done.

In addition to the current retraining protocol, in the future enhancements could be incorporated that make the system more adaptive and responsive to data changes. While the current protocol initiates retraining manually, a future version could establish automated retraining triggers, such as setting a regular retraining interval, detecting data distribution shifts, or initiating retraining once a specified number of new samples are collected. Applying these triggers would allow the model to capture and account for the evolving data patterns with minimal user intervention. Additionally, the retraining process could be refined to allow the user to choose between retraining from scratch or fine-tuning the model with recent data. This choice would be based on the nature of data shifts observed, where retraining from scratch could address significant changes in data patterns, and fine-tuning would support a more gradual adaptation. These improvements would maintain the project's goals of consistency and ease of use, while ensuring that the retraining protocol can adapt to long-term data trends.

5 Results

This section of the report discusses the results and thoroughly evaluates the training process of the models.

5.1 Chapter Highlights

In this chapter, the performance of the three models was evaluated – decision tree, random forest, and XGBoost - based on MSE during training and RMSE for final model selection. XGBoost proved to be the top performer. Feature importance was assessed using SHAP scores, underlining the importance of recent data in the model's predictions. Lastly, the limitations of the project are highlighted, focused on the restricted depth of discussion and explanation of complex methodologies due to the report's page limit.

5.2 Performance Evaluation

During training and hyperparameter tuning, the best decision tree, random forest, and XGBoost regressor were selected according to the lowest MSE on the time-series validation set. During evaluation, the best model for deployment was selected through the RMSE. The other metric is provided as a reference and detection of overfitting, which did not occur. The results are presented in Table 10 and Figure 7. The test set included the months of December, April, July, and October, as per the project requirements.

The final model, selected for deployment was XGBoost, as it had the lowest RMSE score. Since the test set spans from 22/06/2021 to 09/09/2024, an RMSE score of 5.04 means that the XGBoost model, on average, is off by 5.04 $\mu g/m^3$ of ozone and nitrogen dioxide for any day out of the three predictions (in total, more than a 1000 predictions for the three unseen years). We consider this result to be extremely good, taking into consideration all the events that happened in those three years including the pandemic

and its consequences. The inference time is almost immediate, which will be a significant benefit during deployment.

Model	Training and Tuning		Evaluation	
	MSE	RMSE	MSE	RMSE
Decision Tree	35.32	5.66	36.69	5.76
Random Forest	28.60	5.11	31.74	5.36
XGBoost	**21.78**	**4.40**	**28.70**	**5.04**

Table 10: MSE and RMSE Results for the candidate models. The best (lowest) values are bolded.

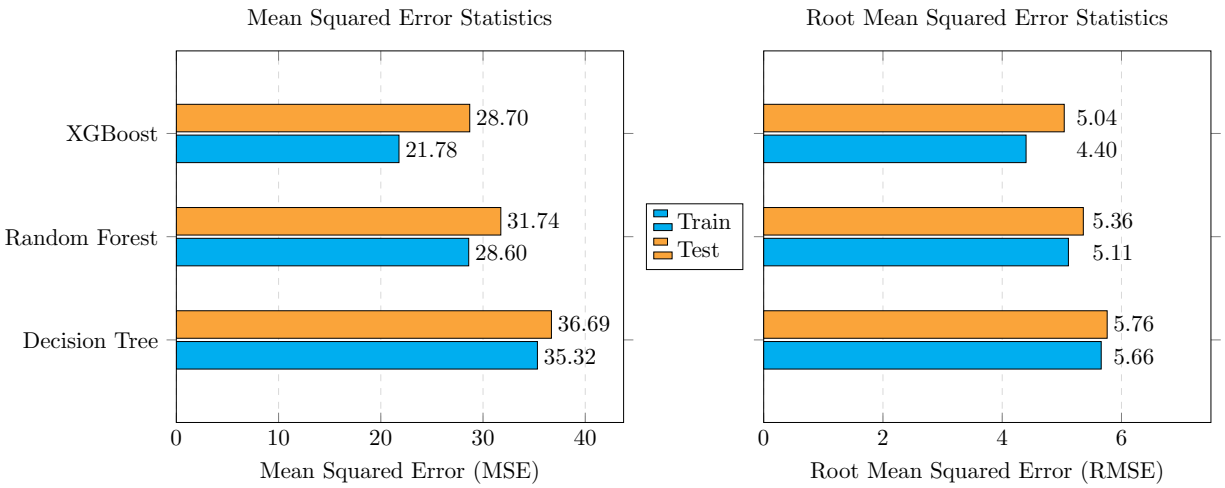

Figure 7: MSE and RMSE Results for the candidate models. Note the difference in the scales for the two plots.

5.3 Feature Importance

The dataset used to calculate the SHAP scores was the held-out test set, as it was not seen previously by the final selected model. The mean absolute SHAP scores, directly interpretable as feature importance can be seen in Figure 8. It can be observed that in general, the most recent data available to the model has the strongest influence on the predictions, which is expected but raises a potential issue. This will be followed up on in the Future Research section.

5.4 Challenges and Limitations

The team wishes they could motivate some of the choices made more in-depth. In particular, we feel that the sections about timeseries CV, SHAP values, and Bayesian search are slightly under-described and rely on the reader's prior knowledge and trust, which can be an issue depending on the target audience. We advise the reader to check the sources provided, as the motivation for the methodology stems from there. As this report has a strict requirement of 30 pages, the team had to reduce their explanations in some parts to abide by those specifications.

6 Conclusion

This section of the report discusses the conclusions of this project combined with impacts and future research recommendations.

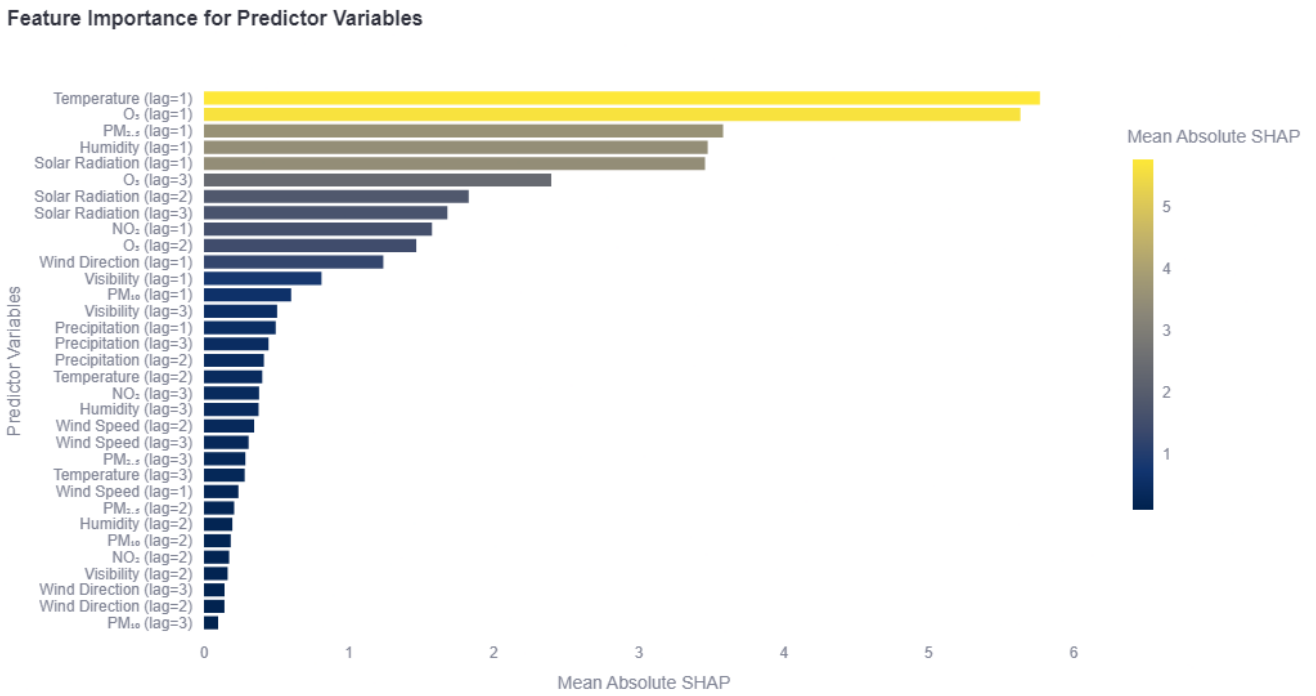

Figure 8: The feature importance of all predictor variables in terms of the mean absolute SHAP value. The lag shows how many days ago the measurement was taken.

6.1 Summary of Main Contributions

The project developed a comprehensive data-driven system for predicting O_3 and NO_2 levels in Utrecht using machine learning techniques.

Data from public and private sources was combined, preprocessed and engineered to tackle this task. Three machine learning models—Decision Tree, Random Forest, and XGBoost were trained, cross-validated, optimized, and evaluated through advanced and consistent techniques. The best-performing model, namely XGBoost, was selected for deployment due to its predictive performance and robustness to missing data.

A Streamlit application was built, incorporating the model's performance into its interface, and was deployed on HuggingFace Spaces, featuring a real-time air quality monitoring system. The app contains a user dashboard, displaying current air pollution values, the model's forecast for the next three days, and numerous awareness-rising sections. An admin dashboard could further be accessed with a password for advanced monitoring techniques, such as custom predictions and comparing the model's historical predictions with the actual pollution values.

The deployment process is highly independent and was automated using GitHub workflows. The codebase was fully documented and a retraining protocol was established for a standardized potential retraining of the models.

6.2 Future Work and Recommendations

While describing the system architecture, an issue related to data storage was raised, related to the fact that every data file is currently stored as a CSV/JSON file in the GitHub Repo, instead of an external

database. We believe that the main idea for improvement would be the addition of an SQL database to allow long-term scalability and clean data practices, for instance, observation of model trends, data degradation, or distribution shifts. Moreover, this would allow for easier storage and versioning of the models. The current storage of the models directly in the code repository is a bad practice and should be avoided as it consumes additional resources, supplied by the version control system. Adding an external database for data and model storage should be the first step implemented if this project is to be extended in the future. We recommend PostgreSQL or MySQL, as they synchronize well with Python.

A further point for improvement would be to transition this model scope from the proof of concept to actual industry deployment, with higher access to computational and financial resources to obtain more data from more sources. This project could be used as a part of a larger warning system e.g for civilian and public health. Alternatively, it could be implemented in proposed as a weather service, with possible predictions for not only pollutants but also other weather data such as precipitation or temperature.

From a research perspective, in this project, tree-based models were trained on a regression task and separately evaluated using the SHAP method for feature importances. During the time the model was being developed, a brand new approach was presented in the field, where the SHAP values become a part of the training process, and the feature importances can be directly manipulated before a model is finished training (Brown et al., 2024). That way, the information from the SHAP values can be extracted and used by the model to adjust its predictions. A future study could address this and compare the obtained metric values to the results of this study.

Another interesting viewpoint of the feature importances is their ability to serve as grounds for model compression. As highlighted in the Feature Importance section, features from the previous day (`lag=1`) seem to have a greater influence on the final prediction of the model, compared to previous values. The model could use fewer variables, by setting a threshold for contribution to the final prediction and eliminating the variables that do not reach that threshold. Hence, the model is of reduced size, which is advantageous both from a research (more control over the model) and business (fewer measurements need to be taken, less data to be stored) perspective. Alternatively, a business company could consider better sensors with more precise measurements for the variables that contribute majorly to the final prediction.

6.3 Broader Impacts

While the air quality forecasting model developed in this project stems from a politically neutral and environmentally friendly perspective, it is critical to address possibly harmful uses. Two main issues can arise from overly trusting the predictions of the system.

Firstly, if the model predicts safe air quality values, while in reality, it is incorrect, it can have a negative impact on the health and well-being of the system users, especially for risk groups, such as people with respiratory issues or pregnant mothers.

In a political manner, powerful stakeholders could use this open-source project to base their decisions on potentially incorrect air quality predictions. For instance, a politician might decide to close off industrial areas in the city to help decrease air pollution, based on the long-term predictions of the model. However, if those predictions are incorrect, this might lead to price inflation of particular products, negatively impacting a large part of citizens.

Responsible use of the platform is, therefore, **strongly recommended**, and any decision made based on the predictions and observed prediction data should be adjusted to match the uncertainty about the validity of those inferences.

On a positive note, the application of machine learning to the air quality issue helps in raising awareness

about its risks and opens future research possibilities, as suggested previously. The final product of this project, namely the deployed application, along with the model, is of interest to both the average citizen, concerned about their own health, and more tech-oriented professionals, with a drive to design and improve existing air quality forecasting platforms. The team hopes that their efforts in developing an easy-to-scale and well-documented air pollution prediction system will inspire individuals to extend, modify, and improve this existing project as a means of improving public health on a large scale.

References

Brown, M. G. L., Peterson, M., Tezaur, I., Peterson, K., & Bull, D. (2024). Random forest regression feature importance for climate impact pathway detection. https://arxiv.org/abs/2409.16609

Das, R., Middya, A. I., & Roy, S. (2022). High granular and short term time series forecasting of pm 2.5 air pollutant-a comparative review. *Artificial Intelligence Review*, *55*(2), 1253–1287.

Gangwar, A., Singh, S., Mishra, R., & Prakash, S. (2023). The state-of-the-art in air pollution monitoring and forecasting systems using iot, big data, and machine learning. *Wireless Personal Communications*, *130*(3), 1699–1729.

Kök, İ., Şimşek, M. U., & Özdemir, S. (2017). A deep learning model for air quality prediction in smart cities. *2017 IEEE international conference on big data (big data)*, 1983–1990.

Li, Y., Lau, A., Wong, A., & Fung, J. (2014). Decomposition of the wind and nonwind effects on observed year-to-year air quality variation. *Journal of Geophysical Research: Atmospheres*, *119*(10), 6207–6220.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems 30* (pp. 4765–4774). Curran Associates, Inc. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

Oldenburg, V., Cardenas-Cartagena, J., & Valdenegro-Toro, M. (2024). Forecasting smog clouds with deep learning: A proof-of-concept. *ICML 2024 AI for Science Workshop*. https://openreview.net/forum?id=UQa2PEVHMF

Srivastava, H., Mishra, S., Das, S. K., & Sarkar, S. (2020). An iot-based pollution monitoring system using data analytics approach. *Electronic Systems and Intelligent Computing: Proceedings of ESIC 2020*, 187–198.

Sun, Y., Zhao, C., Su, Y., Ma, Z., Li, J., Letu, H., Yang, Y., & Fan, H. (2019). Distinct impacts of light and heavy precipitation on pm2.5 mass concentration in beijing. *Earth and Space Science*, *6*(10), 1915–1925. https://doi.org/https://doi.org/10.1029/2019EA000717

Tian, X., Cui, K., Sheu, H.-L., Hsieh, Y.-K., & Yu, F. (2021). Effects of rain and snow on the air quality index, pm2.5 levels, and dry deposition flux of pcdd/fs. *Aerosol and Air Quality Research*, *21*(8), 210158. https://doi.org/10.4209/aaqr.210158

Toma, C., Alexandru, A., Popa, M., & Zamfiroiu, A. (2019). Iot solution for smart cities' pollution monitoring and the security challenges. *Sensors*, *19*(15), 3401.

WHO. (2024, September). Ambient (outdoor) air pollution. https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

Xie, J., Sun, T., Liu, C., Li, L., Xu, X., Miao, S., Lin, L., Chen, Y., & Fan, S. (2022). Quantitative evaluation of impacts of the steadiness and duration of urban surface wind patterns on air quality. *Science of The Total Environment*, *850*, 157957. https://doi.org/https://doi.org/10.1016/j.scitotenv.2022.157957

Yassin, M. F. (2013). Numerical modeling on air quality in an urban environment with changes of the aspect ratio and wind direction. *Environmental Science and Pollution Research*, *20*, 3975–3988.

Appendices

A Hyperparameter Search Spaces and Best Values

In Tables 11, 12, 13 the hyperparameter search spaces, along with the best values found after the Bayesian search are available.

Parameter	Range	Best Selected Value	Range Type
max_depth	[2, 150]	150	Integer
min_samples_split	[2, 20]	2	Integer
min_samples_leaf	[25, 35]	30	Integer
max_leaf_nodes	[20, 60]	60	Integer
ccp_alpha	[0.0, 0.1]	0.1	Real
min_weight_fraction_leaf	[0.0, 0.5]	0.00589	Real
min_impurity_decrease	[0.0, 1.0]	0.17054	Real

Table 11: Decision Tree Configuration Space with Best Values

Parameter	Range	Best Selected Value	Range Type
eta	[0.001, 0.1]	0.0072	Real
n_estimators	[100, 1000]	1000	Integer
max_depth	[1, 20]	3	Integer
subsample	[0.5, 1.0]	0.5	Real
colsample_bytree	[0.5, 1.0]	1.0	Real
reg_lambda	[0.0, 2.0]	1	Real
min_child_weight	[1, 15]	5	Integer
reg_alpha	[0.0, 2.0]	10.0	Real

Table 12: XGBoost Configuration Space with Best Values

B System Metrics During Model Training

The system metrics from training and hyperparameter optimization can be seen in Table 14.

Parameter	Range	Best Selected Value	Range Type
max_depth	[1, 50]	49	Integer
ccp_alpha	[0.0, 0.1]	0.0959	Real
n_estimators	[5, 500]	186	Integer
max_leaf_nodes	[2, 50]	50	Integer
min_samples_split	[2, 25]	2	Integer
min_samples_leaf	[2, 20]	13	Integer
max_features	[0.1, 1.0]	0.3112	Real

Table 13: Random Forest Configuration Space with Best Values

Metric	Decision Tree	Random Forest	XGBoost
CPU Utilization Percentage	8.7%	1.9%	0.0%
Disk Available Megabytes	213483.1	213480.4	213481.9
Disk Usage Percentage	10.6%	10.6%	10.6%
Network Receive Megabytes	48882.34	0.0035	184.40
Network Transmit Megabytes	265.10	0.0051	224.52
System RAM Usage Percentage	8.8%	1.7%	3.5%

Table 14: System Metrics for Decision Tree, Random Forest, and XGBoost. The training used Habrok regular, 32 CPU cores, and 32Gb of RAM.