Harnessing Machine Learning in Air Pollution Management: An In-depth Exploration of Random Forest Models in Predicting NO_2 Levels

Bennet Weiss

October 2023

Abstract

Leveraging the emergent relevance of machine learning, this study applies a Random Forest (RF) model to predict NO_2 concentrations, navigating through the potential of machine learning in managing air pollution control and emphasizing the urgency due to NO_2's adverse health impacts. Data from a measurement station near Genovevalaan in Eindhoven, the Netherlands between 2015 and 2017 is used for training, 2018 is used for testing. Here, NO_2 levels are suspected to be lead by human sources, especially traffic. Multiple RFs are employed using *scikit-learn*: RF1, based on meteorological data; RF2, extended with temporal data; and RF3, further inclusive of ozone and odd oxygen concentration data. While RF1 manages to reproduce basic features of the testing dataset, achieving a Pearson correlation coefficient of 0.64, it falls short in aligning predictions with observed variances (explained variance: 0.41). Wind speed and temperature emerge as the most important meteorological predictors with Gini importances of 0.31 and 0.24, respectively. The inclusion of temporal features, serving as proxies for hourly to seasonal traffic variations, enhances model performance: RMSE diminishes by 14 %, and the Pearson number ascends by nearly 20 %, albeit with a potential uptick in overfitting. In RF3, the addition of ozone and odd oxygen concentrations, which instantaneously become the paramount feature with a combined Gini importance exceeding 96 %, significantly elevates the model. It reduces RMSE by over 85 % compared to RF1, albeit necessitating the deployment of additional concentration sensors at the measurement station.

1 Introduction

Air pollution is a critical global concern with far-reaching environmental and public health implications (WHO, 2022). Among the various air pollutants, Nitrogen Dioxide (NO_2) holds a significant place due to its adverse effects on human health, ecosystem integrity, and its role as a precursor to the formation of other harmful pollutants such as ozone and particulate matter. Exposure to elevated NO_2 levels has been linked to various respiratory diseases. (EPA, 2023)

Given the importance of NO_2 pollution control, accurate prediction of NO_2 concentrations is essential for informed decision-making, public health interventions, and environmental policy development. Traditional methods for NO_2 concentration prediction rely on complex chemical transport models, which often require extensive computational resources and detailed emissions data (Zhang et al., 2023). However, the advent of machine learning and the availability of meteorological data provide an opportunity to develop more efficient and accurate NO_2 prediction models.

In this project, a Random Forest (RF) machine learning model is applied to predict NO_2 concentrations based on meteorological data. Random Forest is an ensemble learning technique known for its robustness, versatility, and ability to handle complex relationships in data. By leveraging meteorological data, which includes parameters such as temperature, humidity, wind speed, and atmospheric pressure, the aim is to develop a predictive model that can capture the intricate interplay between atmospheric conditions and NO_2 levels.

This paper is organized as follows: In the subsequent section, an overview of the data sources and preprocessing steps is provided. Additionally the general setup, optimization, training and evaluating procedure for the RF is described. Three different RF will be trained and optimized; first, one is only trained on meteorological data (RF1). In a second iteration more predictors, solely based on time are included to improve forecast capability without the requirement of additional sensors (RF2). Third, additional chemical components associated with the Nitrogen cycle are included to showcast potential model performance if relevant sensors are available (RF3). The performance of all of these RFs is presented in section 3. Lastly, the results are discussed and the project is concluded.

2 Data and Methods

This section introduces and characterizes the dataset used in this project. Thereafter, methods used are presented, including an overview of the RF method and details of implementation as well as techniques used to analyze performance and behaviour of the RF.

2.1 Data

The data used to train and test the model originates from a measurement station measuring city traffic type chemical air compositions and meteorological variables. This included hourly data for concentration of NO_2 and other components such as Ozone (O_3) and odd oxigen and meteorological data including wind direction, wind speed, temperature, specific humidity, hourly rainfall, pressure, cloud coverage and relative humidity. The respective time was logged in Dutch winter time (UCT + 1h).

The measurement station is situated in Eindhoven, the Netherlands next to Genovevalaan, a five-lane street at that location. This measurement station belongs to the RVIM, the Dutch National Institute for Public Health and the Environment. In this project, data spanning the years 2015 to 2018 was used. As a standard practice in machine learning, the whole dataset was divided into training data, covering the period from 2015 to 2017, and testing data for the year 2018. A time series of the split data of NO_2 concentrations is plotted in Figure 1. The concentrations hover around a mean of $25\,\mu g\,m^{-3}$ with quite large variance and a visible yearly cycle. The training data is subsequently used to optimize and train the RF, while the testing data stays untouched during model development. Once the model has been trained, the testing data is used to check the model robustness on an independent dataset. In Figure 1, columns of nan-values have already been removed as they cannot be processed in the RF. This option was chosen over the possibility to fill nan-values with proxies such as mean or extrapolated data as this would introduce a bias to the forest. Naturally, this assumes that columns with nan-values do not represent significant features that would now be missed in the dataset. Additionally, unphysical negative concentrations have been set to zero.

In this project, multiple different RF have been trained, each one different in the number of different features that the forest can use to predict. Before we turn to the three different RF presented in the following chapter, an overview over this machine learning approach in general and the methods used is given in the following subsection.

2.2 Methods

The Random Forest (RF), the machine learning method used in this project, is renowned for its effectiveness in handling regression and classification problems through the deployment of an ensemble of decision trees. Each tree is developed using a distinctive bootstrap sample from the training data, thus providing a multifaceted modeling approach. At every decision node within a tree, a randomly chosen subset of predictor variables, such as wind speed and temperature, is assessed for splitting. In this sense, all trees are founded on different data and employ unique splits and together, these random trees form forest, hence the name. For predictions, each tree in the forest delivers its own NO_2 concentration estimate, and the final prediction is procured by averaging these various estimates. This ensures a reliable and steadfast prediction, adept at handling the non-linear and intricate relationships between the predictors and the response variable. Moreover, the RF model provides not

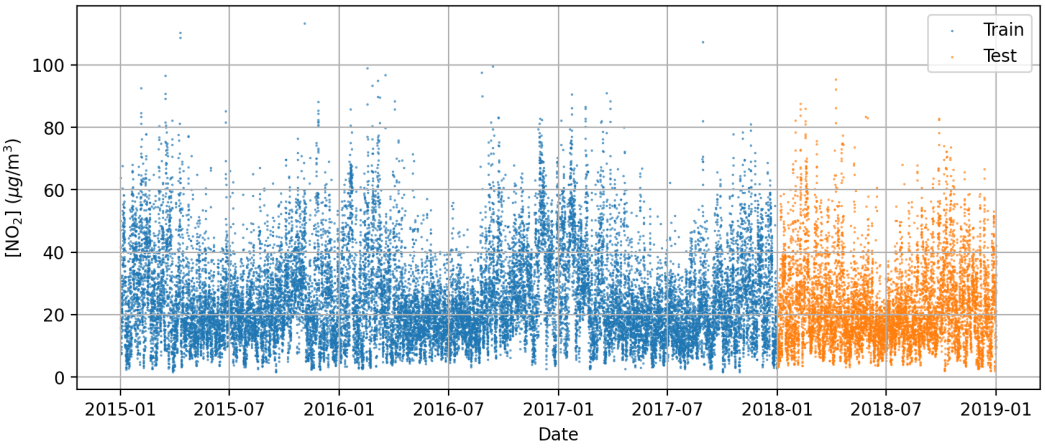

Figure 1: Time series of NO_2 concentration at the measurement station. Training data is marked with blue dots, testing data by orange ones.

only predictions but also insights into the importance of each predictor, highlighting the pivotal variables for accurately predicting NO_2 concentrations.

This project was realized in a *jupyter notebook* (Kluyver et al., 2016) environment, using *scikit-learn* (Pedregosa et al., 2011) for the RF. More explicitly, its *RandomForestRegressor* from *sklearn.ensemble* is used as a base for the forest. Now, the procedure of RF setup, training and testing is briefly outlined.

The aforementioned, now already split, data is further separated into two arrays: the so-called features (X), also referred to as predictors, and the labels. The features include all data that the RF can use to predict. In the first RF, this might only be meteorological data, but the feature set is expanded in the subsequent RF versions. The labels (y) consist of an array of the data that is to be predicted; in this case NO_2 concentrations. How many and which features are included is the main difference of the RFs presented in this paper.

A RF is characterized by its hyperparameters that guide the learning process and can significantly influence model performance. To improve model performance, these hyperparameters have to be optimized. The parameters can be interdependent and thus not all have to be considered. For that, relevant hyperparameters are selected and a value range is prescribed. In this project, five hyperparameters were considered. Firstly, `n_estimators`, representing the number of trees in the forest, is considered within a range of 20 to 200. While a larger number can provide more robust predictions, it also increases the computational load - a good optimum is thus crucial. As will be seen later, adding more trees to a forest does not always translate into performance improvements. Secondly, the `min_samples_leaf`, the least number of samples required at a leaf node, is varied between 3 and 10, safeguarding against overfitting by avoiding trees that are too finely branched. This prevents noise to be picked up as patterns by the forest. Thirdly, `max_samples`, specifying the fraction of samples utilized to train each base estimator (tree), is sampled uniformly between 0 and 0.9, regulating the diversity among the trees in the forest. For values below 1, a tree is not provided with the full dataset but with subset. This can reduce overfitting significantly which is why the upper bound has been chosen to be below 1. Similarly, `max_features`, determining the fraction of features considered during each split. It is explored within a [0, 1] range, enabling control over the bias-variance trade-off of the model. A value below one might introduce a model bias, but can decrease the tendency of the forest to overfit. This range is deliberately kept larger to allow the model to use single features excessively if they prove useful. Lastly, `max_depth`, which dictates the maximum depth of each tree in the forest, is assessed between 2 and 20, guiding the model towards capturing necessary complexity in the data without overfitting, as a deeper forest is more expensive computationally. Balancing these hyperparameters is vital to ensure the RF model is robust, accurate, and computationally efficient.

Traditionally, a machine learning model can be optimized by trying all possible hyperparameter combinations and minimizing the model performance, i.e. the squared error in this project. Even though this is just a selected number of hyperparameters, trying all different combinations is virtually impossible as this is already a five

dimensional problem set including continuous variables. To bypass this problem, *RandomizedSearchCV* from *sklearn.model_selection* is used. This class allows the input of the RF with the hyperparameter set including their value distributions. Here, only constant distributions were used. The number of different hyperparameter combinations that are tried out can be controlled and was set to 500 for all RFs. This number is expected to be sufficient to get a reasonably optimal hyperparamters while still being computationally feasible. By default, *RandomizedSearchCV* uses cross-validation and the number of folds was kept to 5 in this project. This strategy partitiones the training data into 5 equal-sized subsamples. One of these samples is used as validation data, while the rest is used to train the RF with the selected hyperparameter set. Once trained, the validation data is used to evaluate model performance. Subsequently, these steps are repeated until every data subset has served as validation data once. This approach can mitigate overfitting, provided that the number of cross-validation folds is chosen judiciously. A higher number of cross-validation folds typically reduces bias in the model evaluation, but it also entails a higher computational cost and reduces the size of the validation dataset in each fold. The latter might lead to higher variance in model performance estimates, potentially providing a less stable evaluation of how the model might perform on unseen data.

After optimal hyperparameters have been found, the model is trained on the full training dataset within *RandomizedSearchCV* to provide the best model possible. The testing data can then be plugged into the model and can then be used to evaluate the model performance on an independent dataset. In this project, model performance is quantified using four distinct statistical metrics: the Root Mean Square Error (RMSE), the Mean Average Error (MAE), the Pearson correlation coefficient, and the Explained Variance Score. Each of these metrics provides unique insights into the accuracy and predictive capabilities of the model, helping to evaluate its effectiveness in diverse dimensions. The Root Mean Square Error (RMSE) quantifies the model's prediction error by computing the square root of the average squared differences between the predicted and observed values. It is particularly sensitive to large errors and thus provides a useful measure of how well the variance is captured. The Mean Average Error (MAE) represents the average of the absolute errors between the predicted and actual values. Unlike RMSE, MAE is not sensitive to the square of the errors, providing a more direct average of the error magnitudes and offering a straightforward interpretation of the model's accuracy. The Pearson correlation coefficient measures the linear relationship between the actual and predicted values, providing a value between -1 and 1. A coefficient of 1 indicates a perfect positive linear relationship, -1 a perfect negative linear relationship, and 0 no linear relationship. It offers insights into how closely the model's predictions follow the observed data trend. The Explained Variance Score gauges the proportion of the dataset's variance that is captured by the model. A score of 1 indicates that the model perfectly predicts the observed values, while a score below 1 suggests that the model is only capable of predicting a portion of the variance in the data, and a score of 0 or below indicates that the model fails to predict the variance altogether. In the context of this project, the model should aim to get RMSE and MAE as close to zero as possible, and the correlation and the explained variance as close to one as possible.

While the performance metrics offer a comprehensive evaluation of the model, understanding the influence exerted by each predictor or feature facilitates a more nuanced interpretation of the results and model behaviour. To quantify the importance of a feature, the attribute *feature_importances_* of *RandomizedSearchCV* is used which works as follows. A RF splits the data in order to decrease the impurity as much as possible. The higher the impurity reduction, the more important is the split considered to be and thus the more important is the respective feature. Based on this idea, the Gini importance quantifies the features impact on impurity reduction at all nodes across the whole forest to obtain a metric of feature importance. This is what is ouput by *feature_importances_* used in this project. While Gini importance is widely utilized due to its ease and computational efficiency, it bears limitations such as a potential bias towards features with more categories or higher cardinality, and might not fully reveal the true predictive power or relevance of a feature in the context of correlated variables within the model. (Nembrini et al., 2018)

As the basis for the data used, and the set up as well as analysis methods of RFs used in this project are laid out, the next chapter presents the results of the RF models.

3 Results

In this section, three RF models are presented. First, a RF (RF1) is optimized and trained solely based on meteorological data. Second, the dataset is expanded using time metrics and a new RF (RF2) is optimized and trained. Third, measured concentrations of ozone odd oxygen are added to the latest dataset to train the most capable RF model (RF3) of this project. All RFs are assessed using the statistical and feature importance metrics introduced in the earlier chapter.

3.1 RF Model Trained with Meteorological Data (RF1)

First, a basic RF has been optimized and trained based only on hourly meteorological data including wind direction, wind speed, temperature, specific humidity, hourly rainfall, pressure, cloud coverage and relative humidity. The optimized hyperparameters of the RF can be found on the leftmost column of Table 1.

Table 1: Optimized hyperparameters for the three different Random Forest models. RF1 refers to the RF based purely on meteorological data. RF2 stands for the RF based on meteorological data extended with temporal data. RF3 includes data for ozone and odd oxygen concentration.

Parameter	RF1	RF2	RF3
Max. Depth	18	18	17
Max. Features	0.39	0.53	0.97
Max. Samples	0.26	0.74	0.65
Min. Samples Leaf	6	4	3
N. Estimators	172	195	59

Here, the optimal number of trees was found to be around 170. To assess the importance of the number of trees for the RF, the RF was trained multiple times based on the optimized hyperparameters but with varying number of trees. For each number of trees the forest was trained 10 times independently to get a statistically more robust picture. The RMSE of the prediction for testing and training data is plotted in Figure 2. It is evident that the model performs consistently better on the training data than the testing data. This might be a result of overfitting but it is also possible that the testing data shows patterns that the RF has not seen during its training period, possibly related to internal climate variability. What is also visible is the significantly reduced RMSE for higher tree numbers. This is not surprising as increasing the number of trees in a RF generally allows the model to capture more complexity and variations within the data, ultimately leading to improved predictive accuracy and reduced error rates. At very high tree numbers in the order of 100, however, the improvement seems to vanish and higher tree numbers do not continue to improve model performance. At this point, the model has extracted as much information as possible from the data, and additional trees merely consume computational resources without enhancing the model's predictive capability. Consequently, a number of 172 estimators seem to be located at an optimum with minimal RMSE at acceptable computational cost, making it a good choice for the RF indeed.

Turning to the prediction performance of this RF, Figure 3 scatters the predicted NO_2 concentration as a function of the observed concentrations for training (blue) and testing (orange) data. As indicated by the black dashed line, the 45 degrees diagonal shows the optimal distribution of these points. Both distributions seem to overestimate concentrations for low observations as they are found north of the black line for low measured concentrations, while the model generally underestimates the high concentrations, found south of the black line to the right of the plot. The model seems to be more confident in predicting higher concentrations on the training data, though.

A time series of the prediction for training (top) and testing (bottom) is depicted in Figure 4. The observations are plotted, too, so one can immediately see the difference. Here, the time resolution is so low that only large-scale features can be observed. Those seem to be followed only in the most general way. Most striking is the incapability of the RF to reproduce the variance of the observations.

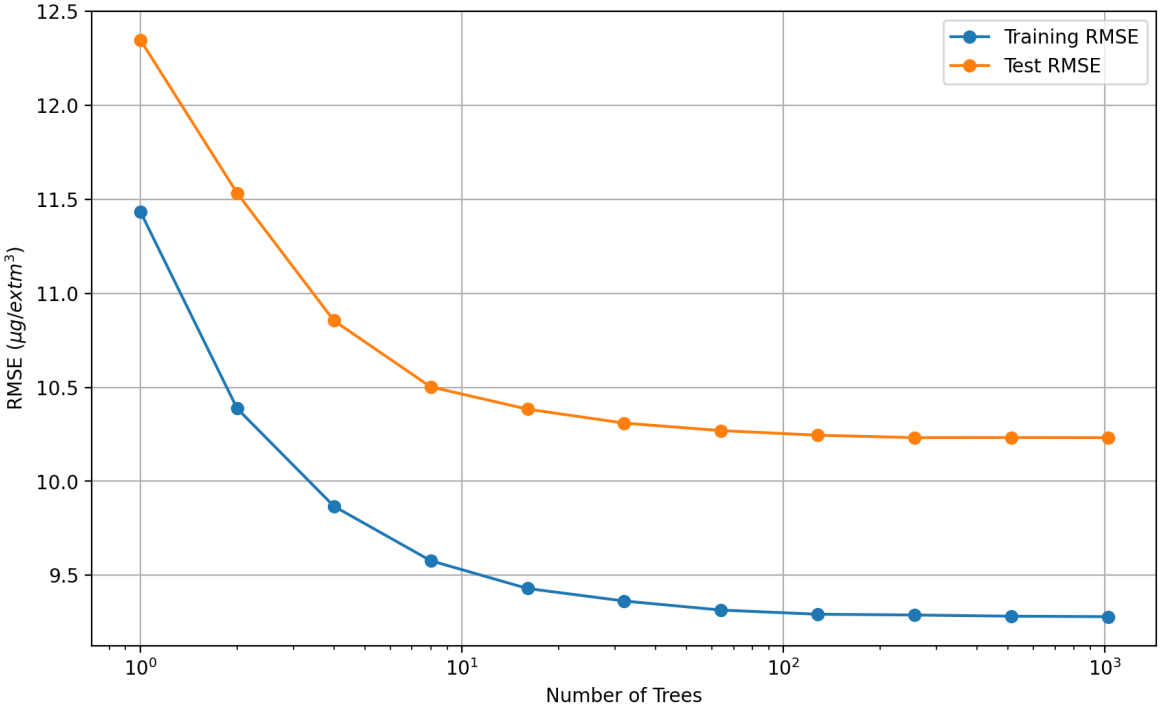

Figure 2: Root mean square error of training (blue) and testing (orange) data for RF with hyperparameters as in 1 (left) but varying number of estimators.

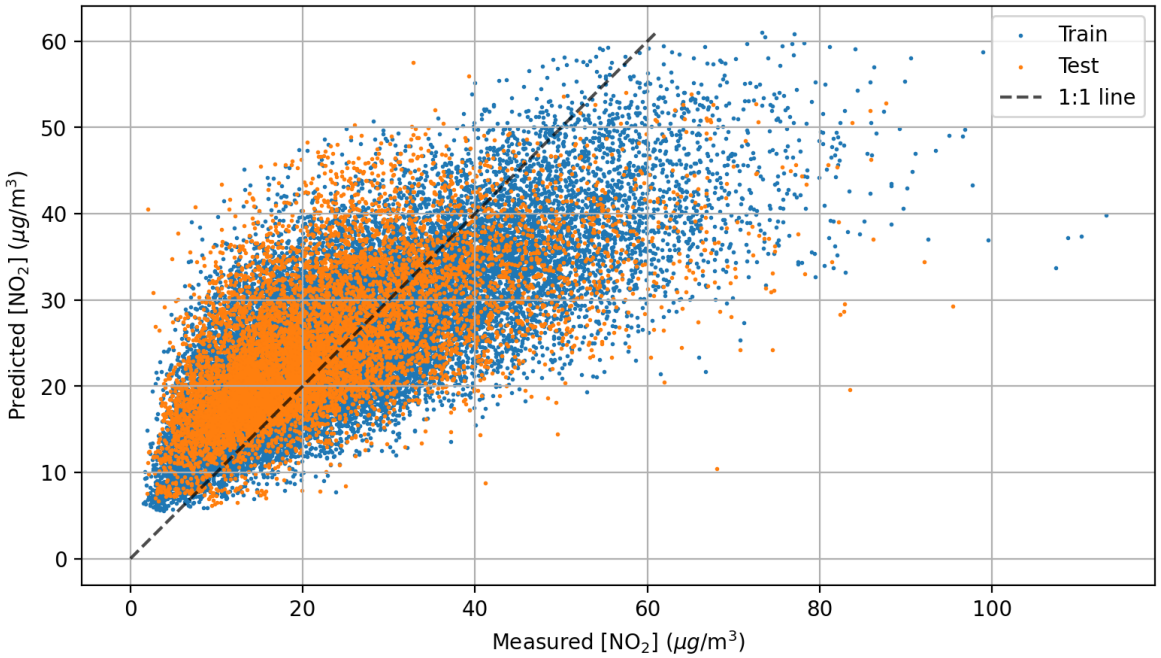

Figure 3: Scatter plot of predicted vs. observed NO_2 concentrations with hyperparameters as in 1 (left) for training (blue) and testing (orange) data. The black line indicates the optimum line.

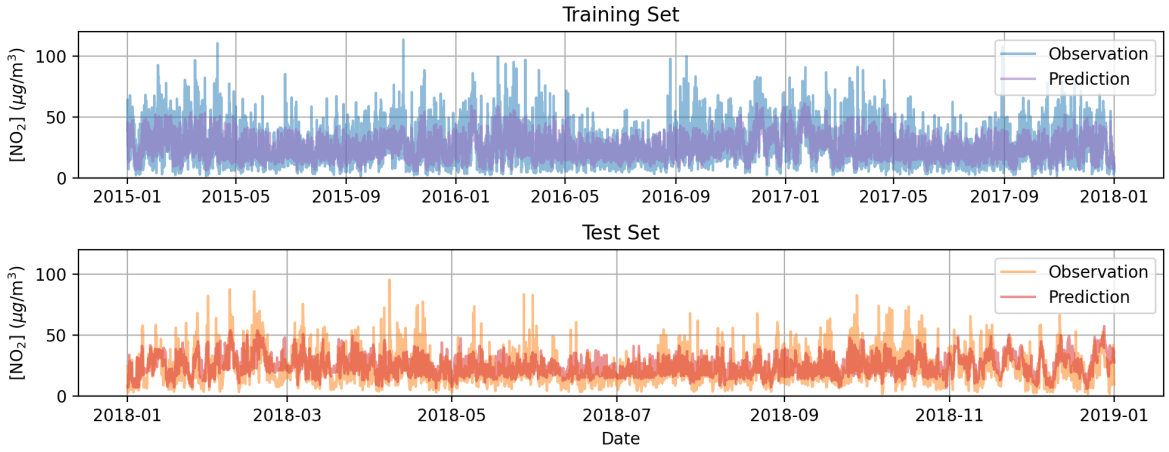

Figure 4: Time series of predicted and observed NO_2 concentrations with hyperparameters as in 1 (left) for training (top) and testing (bottom) data.

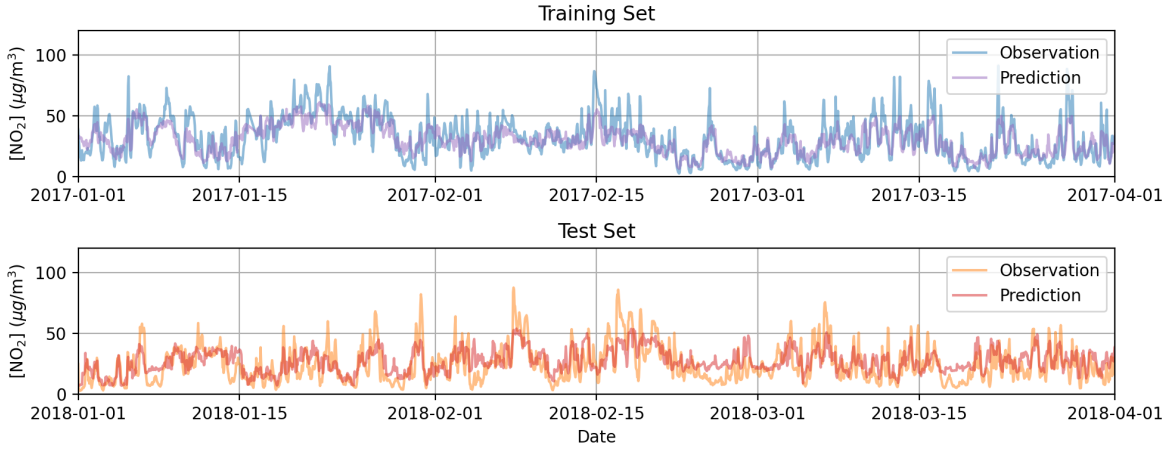

Figure 5: As in Figure 4 but zoomed in on first three months of 2017 (top) and 2018 (bottom) for training and testing data, respectively.

To get a better sense for the model performance on short timescales, Figure 5 shows the time series of the concentrations as in Figure 4, but only for the first 3 months of the year 2017 and 2018 for training and testing data, respectively. From this, it is evident that, while the model is able to reproduce the general trend, high peaks are never predicted.

This is also reflected in Table 2 that shows the various statistics metrics for this random forest. When comparing the performance of the RF on training and testing data it becomes clear that it performs better on the training data, as discussed before. Also, the fact that the RMSE is higher than the MAE implies that the RF has problems predicting the more extreme values. The full variance of the observations is thus not covered which can also be seen by looking at the explained variance which is in the order of 50 %. The correlation coefficient is higher, though, implying that the general features are predicted.

To get a better sense of the reasoning of the RF, Figure 6 shows the feature importance sorted from the most important to the least important feature. Wind speed and temperature seem to be the most important predictor, followed by pressure and wind direction. To understand this, it is good to first recall the sources of NO_2. Even though anthropogenic NO_x ($= NO + NO_2$) emissions have been decreasing significantly in the Netherlands over the last decade, especially road transport is still dominant source of NO_x, making up 28.5 % of all anthropogenic

Table 2: Statistical Metrics including root mean square error (RMSE), Pearson Number for correlation, explained variance score and mean average error (MAE) for RF1 on training and testing data.

Metric	Train	Test
RMSE ($\mu g/m^3$)	9.28	10.23
Correlation	0.77	0.64
Explained Variance	0.58	0.41
MAE ($\mu g/m^3$)	6.93	7.84

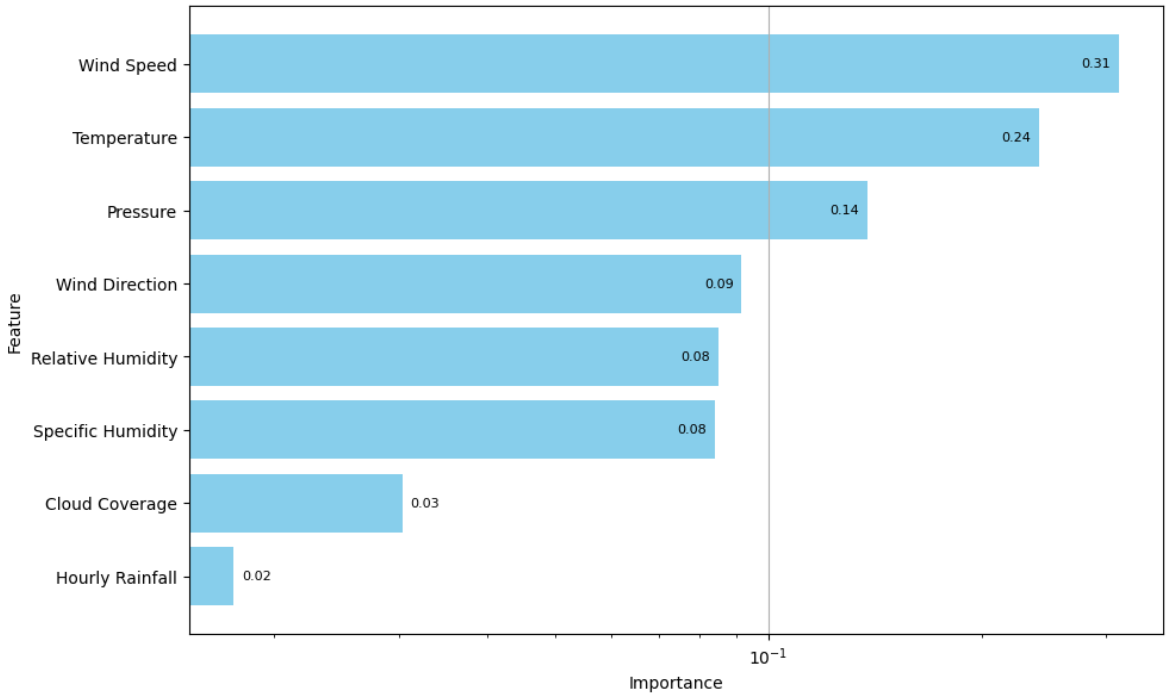

Figure 6: Gini importance for features used in RF1. The combined importance of all features is 1. Note the logarithmic scale.

NO_x emissions (European Environment Agency (EEA), 2021). Although most emissions are in the form of NO, the cycling between NO_x components happens on the timescale of minutes during the day which makes it unnoticeable for the hourly measurements in the used dataset (Jacob, 2000, p. 212). Therefore, it is reasonable to assume that a significant fraction of the measured NO_2 concentration at the measurement station originates from proximity. As wind strengths controls the mixing efficiency in the boundary layer it effectively governs the NO_2 concentrations at the street levels. If the wind is weak, for instance, the NO_2 could accumulate around the street leading to a high concentration. Similarly, the wind direction plays a role as the emissions from the street might be blown towards or away from the sensor. Consequently, it is not surprising that wind speed is the single most important predictor of the RF.

Temperature plays a significant role for the NO_x concentrations, too. NO production in particular is promoted at higher temperatures as its equilibria with N_2 and O_2 are shifted in favor of NO then (Jacob, 2000, p. 212). This could explain why temperature is the second most important feature of this RF. It is interesting to note, that hourly rainfall is the least important feature to predict NO_2 concentrations. This may seem counter-intuitive, as lightning is a significant source of NO_2 in the troposphere (Jacob, 2000, p. 217-218) and rainfall might be well correlated with lightning. Lightning is, however, only a source in the upper troposphere, far away from the sensor.

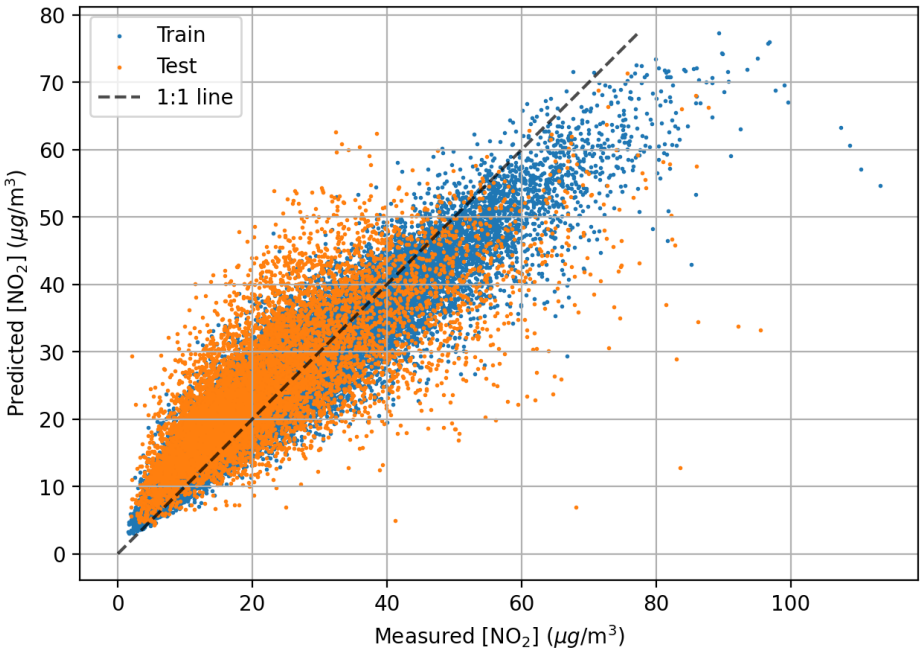

Figure 7: As in Figure 3 but for RF2.

If local traffic is indeed a significant contributer to the measured concentrations, it might be a good idea to include traffic data into a model. A good proxy for traffic could be time, which is what is incorporated in the RF presented in the follwing subsection.

3.2 Including Time Data as Predictor (RF2)

In addition to the meteorological data, this RF2 aims to resolve the time dependence of NO_2 concentrations. However as a combination of date and time is a data format a RF cannot handle, they have to be converted into plain numbers. One way to do this, is to split up the date and time into three categories: hour of day, day of the week and day of the year. Other time based categories have been tested, too, including working day, week of year, holiday, day of month and winter time. However, neither of these improved the prediction capabilities, potentially because they are already implicitly integrated within the chosen categories, or because they might just not be a good variable to describe traffic. Therefore, they have been omitted.

The optimized hyperparameters of the RF using these additional predictors can be seen in Table 1 (middle). The scatter plot showing the predictions vs. observations is shown in Figure 7. In comparison to Figure 3, the distribution of training and testing data is closer to the optimum line which implies improved predictions. Again, however, the predictions based on the training data are better than those based on the testing data.

The time series of the predictions are again comprised in Figure 8 for the whole dataset and for three months in Figure 9. The former shows significantly reduced variance, especially for the training data, while the latter exemplifies especially the improved correlation, also most notably for the training data.

Nevertheless, the predictions have not only improved for the training data, but also for the testing data. This can also be seen in Figure 10 that visually compares the statistics metrics from Table 2 and and 3, which contains the statistic metrics for RF2. This figure illustrates the enhancement in performance achieved by incorporating temporal predictors. Consequently, while RF2 might represent a more overfitted model than RF1, its absolute performance is still superior. Interestingly, the difference between training and testing improvement are much more significant along absolute error metrics such as RMSE and MAE than along the correlation and explained variance metrics. This is also visible in Figure 9 (bottom), where the red prediction line seems to follow the

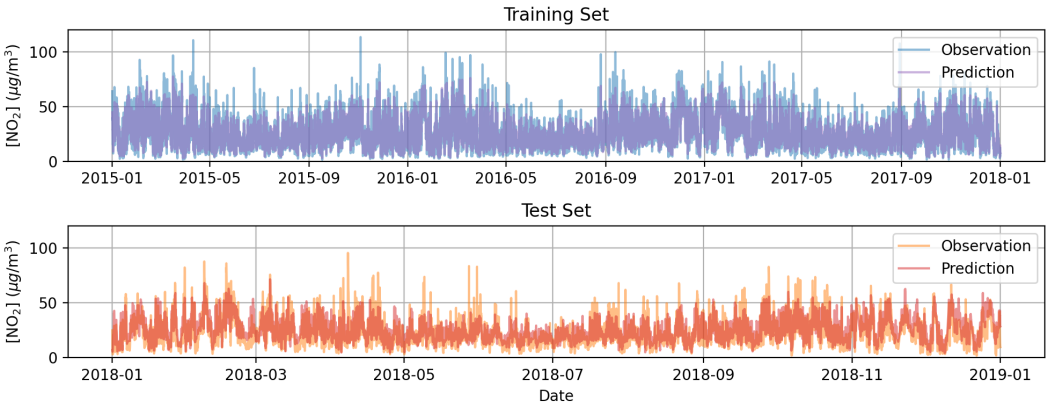

Figure 8: As in Figure 4 but for RF2.

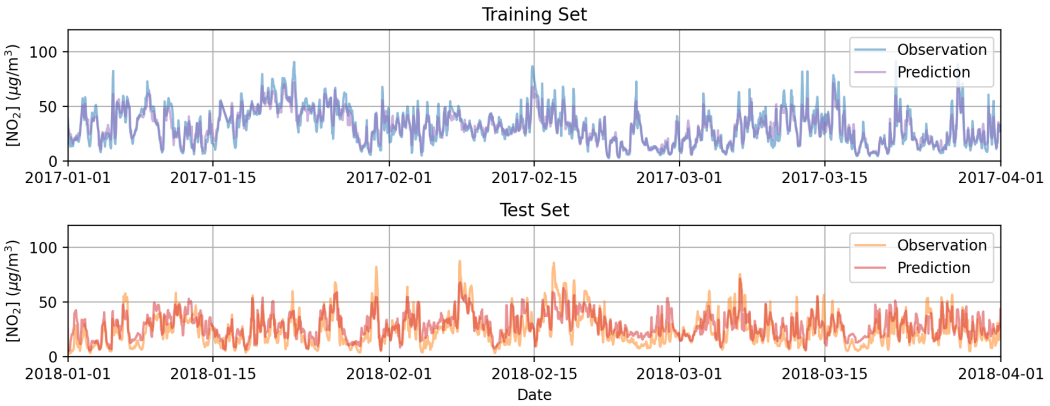

Figure 9: As in Figure 5 but for RF2.

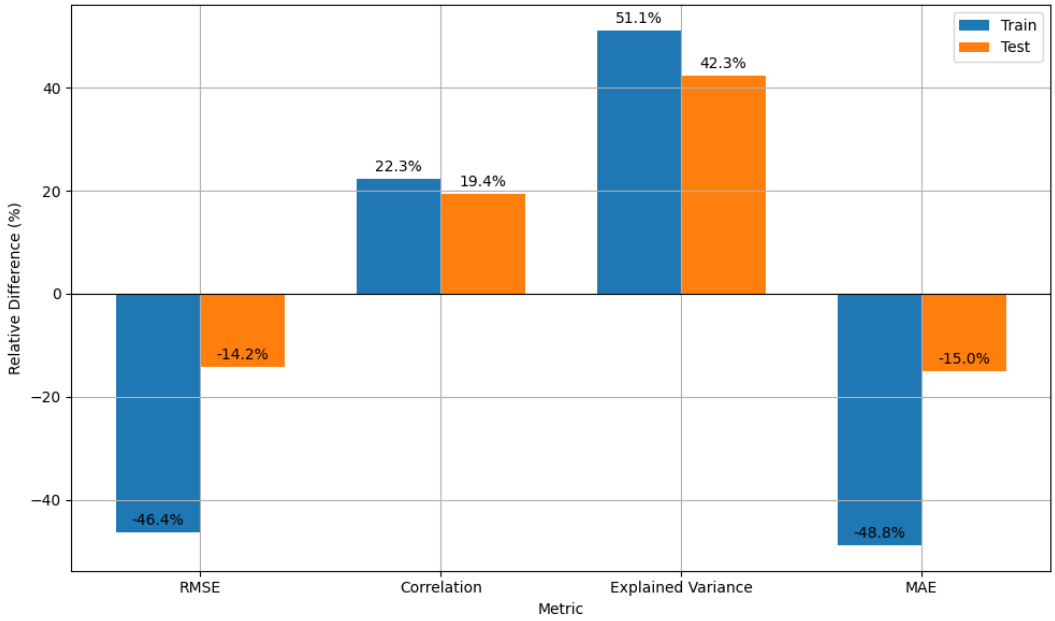

Figure 10: Quantitative improvement along selected statistics metrics from RF1 to RF2 by relating data from Tables 2 and 3.

orange observation line quite well but seems to be offset by a constant. This behaviour is not visible at the top of the figure, and might explain why the error metrics do not improve as much for the testing as the training data. At the same time the variance and correlation improves by a similar magnitude as the overall shapes are predicted correctly. A possible explanation could be the aforementioned decreasing trend of NO_x concentrations which is not picked up by the RF.

Table 3: As in Table 2, but for RF2.

Metric	Train	Test
RMSE ($\mu g/m^3$)	4.9769	8.7821
Correlation	0.9446	0.7630
Explained Variance	0.8800	0.5807
MAE ($\mu g/m^3$)	3.5512	6.6602

By looking at Figure 11, which depicts the feature importance for RF2, we can learn how much the newly included temporal predictors influenced the splitting within the trees of the forest. Wind speed and temperature stayed the most important features but thereafter the temporal features are found in the bar chart. Of them, day of the year is the most important. This could be due to the changing emission profile of humans over the year. For instance, we might see elevated levels on the first day of the year due to fireworks and generally higher concentrations as there might be more people using their bike in the summer than in the winter. This is something the RF could pick up by the day of the year feature. Additionally, day of the year also captures the change from winter to summer time in the Netherlands, albeit off by a couple of days. At the same time, day of the year is a good proxy for certain meteorological conditions, especially temperature. Consequently, some of the temperature's feature importance might have migrated to day of the year. This possibility is supported by the fact that the importance of temperature sees a higher relative decrease with respect to 6 than wind speed, for example. Next, hour of the day serves as an important feature for the RF. This is most likely linked to the changed traffic situations depending on the time: rush hour in the morning and afternoon and almost no traffic

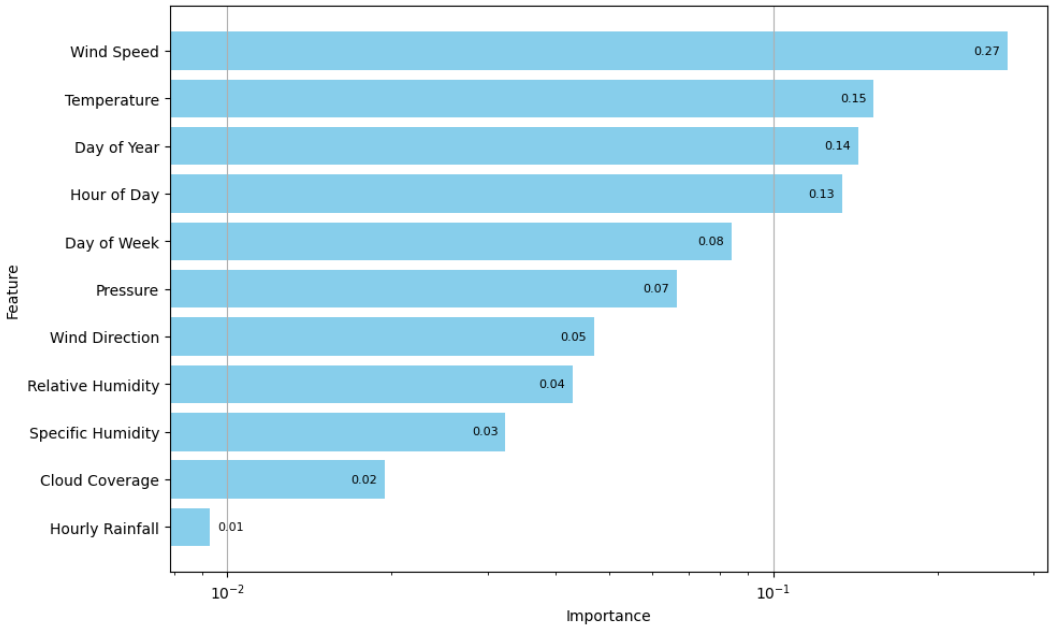

Figure 11: As in Figure 6 but for RF2.

during the night. Additionally, there might also be a link to the changed chemistry of NO_x during the night. As NO_2 needs light to react with oxygen to ozone and NO, this reaction is suppressed during the night (Jacob, 2000, p. 213). Consequently, all direct NO_2 emissions would not be in equilibrium with NO as they are during the day. However, direct NO_2 emissions are low, especially during the night with minimum traffic so it remains unclear from this analysis how important this factor is. Lastly, day of the week is a non-negligible temporal predictor. It most certainly captures the difference between weekend and weekday traffic and possibly even differences between certain days. This could be, for instance, no trucks driving on Sundays or lower traffic on Mondays and Fridays as more people stay to work from home.

In summary, including time dependent predictors does indeed improve prediction quality, even though there are differences in the magnitude of improvement across the different statistics metrics. The mixed level of improvement is especially relevant for the testing data which is most relevant to demonstrate the robustness of the model. At the same time, it is important to note that this is a virtually free improvement as no extra sensors on the measurement station are required. Subsequently, a RF based on additional sensor data, namely ozone and odd oxygen is presented.

3.3 Including Ozone and Odd Oxygen as Predictor (RF3)

The last RF (RF3) trained for this project includes, in addition to the predictors before, ozone and odd oxygen ($= O+O_3$) concentrations. Naturally, this requires additional sensors present at the measurement station and thus limits the applicability of this model. However, there are certain scenarios where such a RF might be of use. For example, at stations with a broken or no NO_2 sensor at all, reliable concentrations can be calculated. Additionally, a model like this might prove useful to check whether data produced by a NO_2 sensor is trustworthy, for instance when observing particular patterns. Other than that, one might consider this subsection as a performance presentation of the capabilities of a random forest. Including ozone and odd oxygen concentrations is expected to drastically improve NO_2 concentrations as ozone is produced in the NO_x cycle (Jacob, 2000, p. 215) and oxygen atoms can combine with N_2 to form NO (and the other way round) (Jacob, 2000, p. 212). They are thus important ingredients of the NO_x chemistry. In fact, the latter is the only thing one needs to know to employ a RF as the whole chemistry is captured by the statistics in the RF.

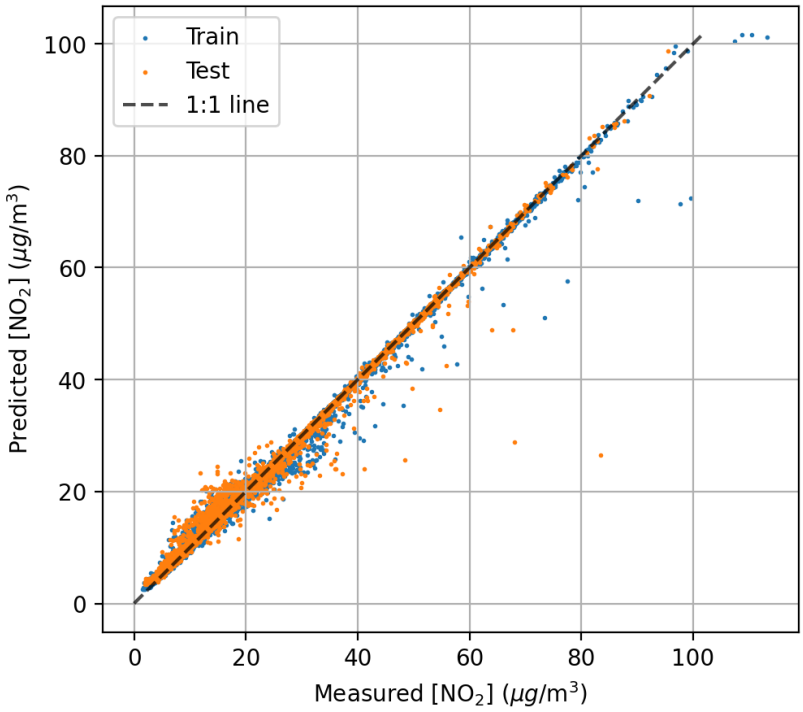

Figure 12: As in Figure 3 but for RF3.

The optimized hyperparameters for this RF can be found in Table 1 (right). It is interesting to see that this RF employs significantly less trees and also uses almost all features for each peak. The former may be connected to the significantly less complex relationship between predictors and labels due to the added features. The maximum features hyperparameter that is almost increased to 1, is probably only not exactly at 1 because this combination has not been tried in the randomized hyperparameter optimization. Both of these facts hint in the direction that now less features are significantly influencing the RF predictions. This will be discussed later, when a look at the feature importance is taken.

Figures 12, 13 and 14 perfectly demonstrate the now unlocked performance of this RF. The scatter plot in Figure 12 is very close to the 1:1 line, with only a few outliners where the RF underestimates the magnitude of a peak which can also be identified in Figure 13. The latter and the time evolution in Figure 14 show almost no difference between observed and predicted concentrations, neither for training nor testing data, as both lines perfectly overlap.

The impressive performance of RF3 is also reflected in the significantly improved statistics metrics shown in Table 4. Here, a pattern emerges once again, showcasing a larger relative difference between training and testing data for error metrics (RMSE and MAE) compared to correlation and explained variance metrics. This could once be related to the overall negative trend of NO_2 concentrations. Note that even though differences between training and testing data persist, the absolute values of all of these metrics are very good. Figure 15 shows this strong relative improvement relative to RF1.

Lastly, Figure 6 shows the feature importance of RF3. Here, ozone and odd oxygen features absolutely dominate the predictions of the RF with a combined importance of more than 96 %. This explains the simpler structure of the RF and why the optimization wanted to include all features in all trees. A tree without the features ozone and odd oxygen concentration seems to be almost worthless for this RF. Following this reasoning, it might be possible to setup a new RF model solely based on those two features without much of a performance loss. This is out of the scope of this project, though.

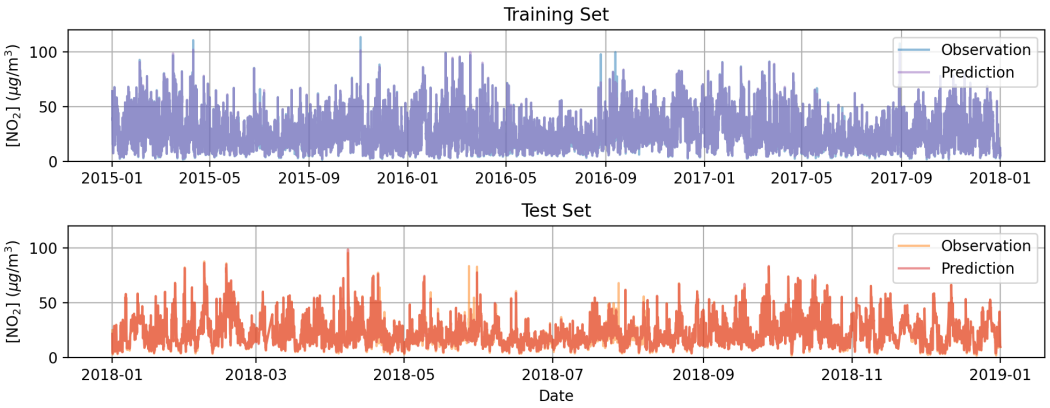

Figure 13: As in Figure 4 but for RF3.

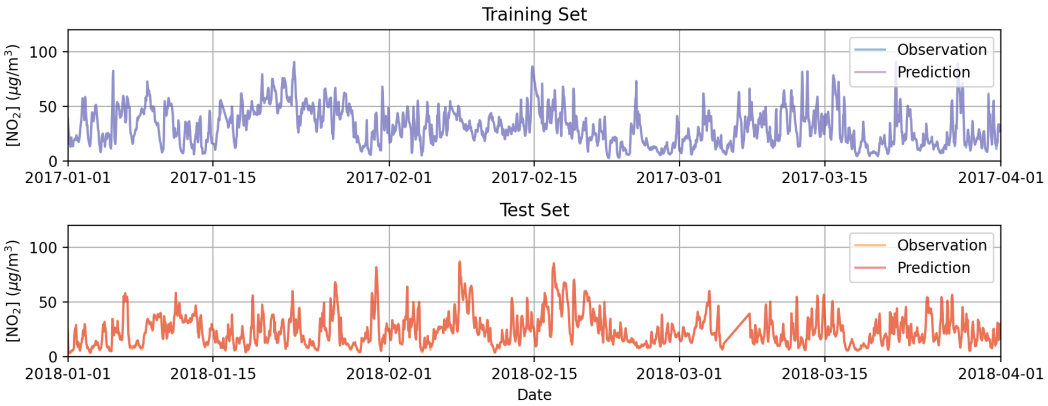

Figure 14: As in Figure 5 but for RF3.

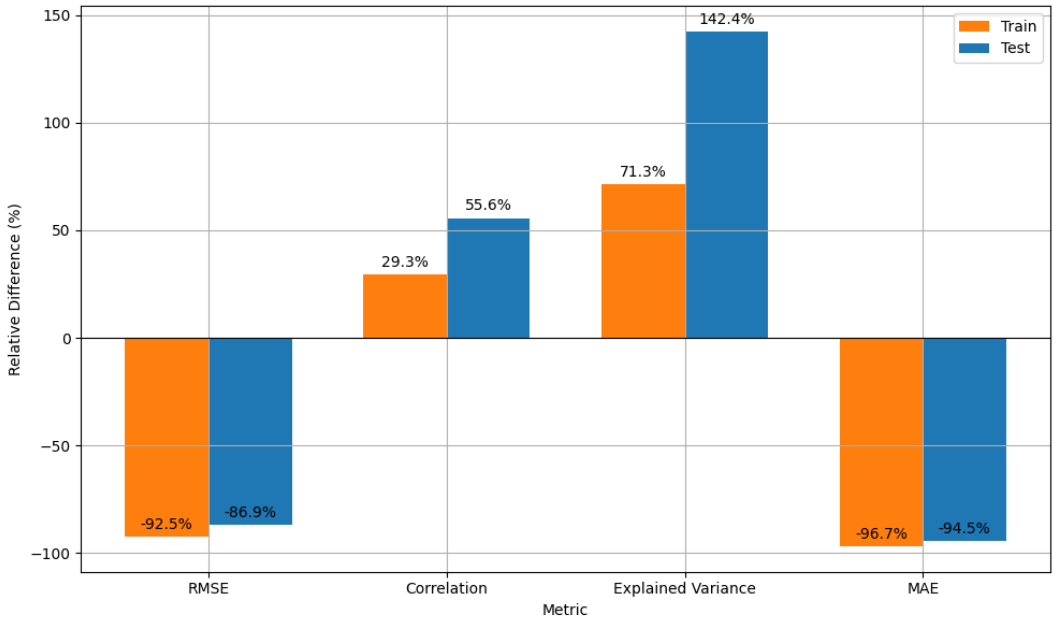

Figure 15: As in Figure 10 but for RF3.

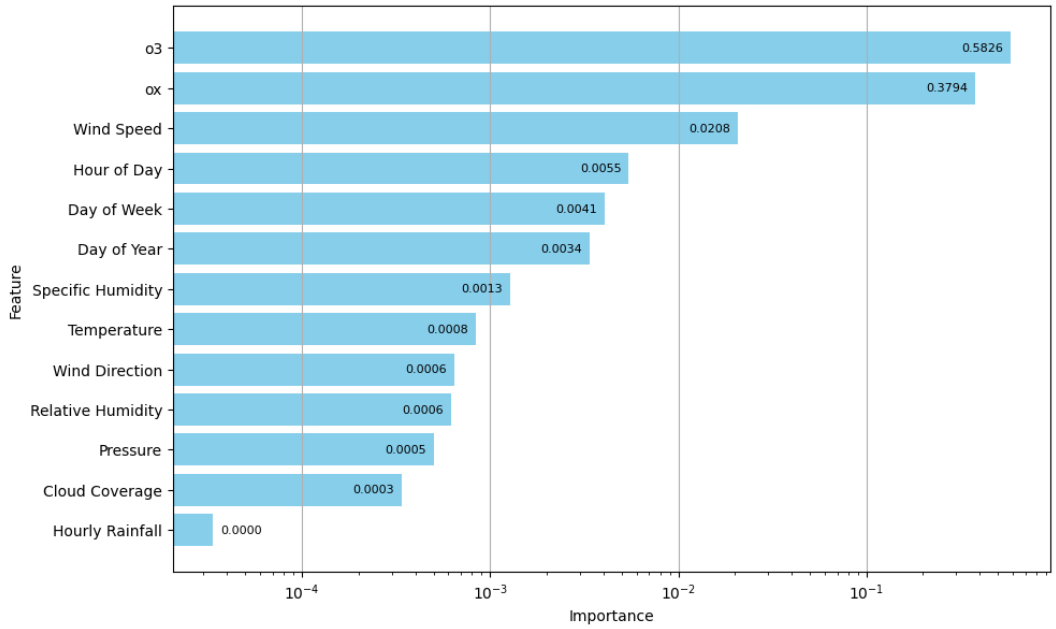

Figure 16: As in Figure 6 but for RF3.

Table 4: As in Table 2, but for RF3.

Metric	Train	Test
RMSE ($\mu g/m^3$)	0.6960	1.3366
Correlation	0.9988	0.9947
Explained Variance	0.9977	0.9893
MAE ($\mu g/m^3$)	0.2309	0.4324

4 Discussion and Conclusion

In this project, the capabilities of a random forest model to predict NO_2 concentrations at a measurement station near a busy road in Eindhoven, the Netherlands, were explored. Three RFs were employed: RF1 is solely based on meteorological data, RF2 includes temporal data and RF3 includes ozone and odd oxygen concentrations to all of that. All RFs were optimized using training data ranging from the years 2015-2017 and an optimal set of hyperparameters was found. The influence of the hyperparameter number of trees was assessed separately. It was found that the model performance decreases with increasing number of trees. After around 100 trees, however, the performance levels off and does not improve further. With the optimized parameters, predictions for the training period, and the testing period, 2018, were employed. RF1 performed the worst of all forests tested in this project, with a RMSE of around $10\,\mu g\,m^{-3}$. This is quite high considering an average concentration of around $25\,\mu g\,m^{-3}$. Still, with a correlation coefficient of 0.64, the basic movement of the concentration signal was reproduced. The variance, however could not be mirrored very well, with an explained variance score of 0.41 for the testing data. Wind speed, possibly controlling the mixing of air with atmosphere layers above was found to be the most important meteorological feature. Temperature was the second most important which was hypothesized to be connected with its influence on chemical NO equilibria.

Including temporal data at no extra sensor cost in RF2 improved the performance significantly, especially the variance was better reproduced with an explained variance score of now 0.58 for testing data. Possible reasons could be seasonal and day/night changes of traffic and chemistry that influence NO_2 levels. The improved performance came with the caveat of a possibly overfitted RF as its performance on the training data was significantly better. It was speculated that the overall trend of decreasing NO_2 concentrations could be a factor. If this is true, the model could be further improved by adding another feature like *year after 2015*. This is left open for future endeavors. While including a binary feature that tracks the change between winter and summer did not improve the overall performance of the RF, it might be interesting to see if it can improve the performance around the date of change of winter time to summer time and vice-versa. As these two dates are not always on the same day in the year, improvements might be significant here.

Lastly, a RF with ozone and odd oxygen concentration as features was trained. Not surprisingly considering the atmospheric chemistry, this RF3 performed by far the best and with correlation and explained variance score around 0.99. Due to the overwhelming feature importance of only ozone and odd oxygen, making up more than 96 % together, a model only based on these constituents would be thinkable and is left for future research. The obvious caveat of this RF is, however, that additional sensor data is required and might not always be available, or, an actual NO_2 sensor might already be in place at these stations.

With the random forest developed in this project, the network of NO_2 concentration control could become much more dense. Virtually every meteorological measurement station could be expanded with a virtual NO_2 sensor, albeit with limited prediction accuracy. This could help understanding (spatial) NO_2 concentration patterns and offer enhanced population protection.

References

EPA. (2023). *Nitrogen dioxide (no2) pollution*. United States Environmental Protection Agency. Retrieved October 8, 2023, from https://www.epa.gov/no2-pollution

European Environment Agency (EEA). (2021). *Netherlands - air pollution country fact sheet*. Retrieved October 12, 2023, from https://www.eea.europa.eu/themes/air/country-fact-sheets/2021-country-fact-sheets/netherlands

Jacob, D. J. (2000, January). *Introduction to atmospheric chemistry*. Princeton University Press. https://doi.org/10.1515/9781400841547

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. Jupyter notebooks – a publishing format for reproducible computational workflows (F. Loizides & B. Schmidt, Eds.). In: *Positioning and power in academic publishing: Players, agents and agendas* (F. Loizides & B. Schmidt, Eds.). Ed. by Loizides, F., & Schmidt, B. IOS Press. 2016, 87–90.

Nembrini, S., König, I. R., & Wright, M. N. (2018). The revival of the gini importance? (A. Valencia, Ed.). *Bioinformatics*, *34*(21), 3711–3718. https://doi.org/10.1093/bioinformatics/bty373

OpenAI. (2023). *Chatgpt*. Retrieved October 13, 2023, from https://chat.openai.com

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

WHO. (2022). *Ambient (outdoor) air pollution*. World Health Organization. Retrieved October 8, 2023, from https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

Zhang, D., Wang, Q., Song, S., Chen, S., Li, M., Shen, L., Zheng, S., Cai, B., Wang, S., & Zheng, H. (2023). Machine learning approaches reveal highly heterogeneous air quality co-benefits of the energy transition. *iScience*, *26*(9), 107652. https://doi.org/10.1016/j.isci.2023.107652

Acknowledgments

ChatGPT (OpenAI, 2023) was used in this project.

A Python Code

```
1
2  # IMPORTS
3  # general
4  import pandas as pd
5  import os
6  import matplotlib.pyplot as plt
7  import datetime
8  import numpy as np
9
10 # for RF
11 from sklearn.ensemble import RandomForestRegressor
12 from sklearn.model_selection import RandomizedSearchCV
13 from scipy.stats import randint, uniform
14
15 # for stats
16 from sklearn.inspection import permutation_importance
17 from sklearn.metrics import mean_squared_error, explained_variance_score,
       mean_absolute_error
18 from scipy.stats import pearsonr
19 from matplotlib.patches import Patch
20 from statsmodels.tsa.stattools import acf
21
```

```
# global variables
DPI = 200
N_iter = 500
# Load the data: look at street
data_folder = 'Data'
file_name = 'Street_NL10236-AQ-METEO.csv' #'Urban_NL10418-AQ-METEO.csv' #'
    Rural_NL10644-AQ-METEO.csv'
data_imported = pd.read_csv(os.path.join(data_folder, file_name), sep=";")
variable = 'no2' # which variable do we want to predict

# prepare dates
data_imported['date'] = pd.to_datetime(data_imported['date'])

# select only meterological variables + date + look at selected variable
met_vars_initial = ['wd', 'ws', 't', 'q', 'hourly_rain', 'p', 'n', 'rh']
selected_data = data_imported[met_vars_initial + ['date'] + [variable]].dropna()
selected_data.loc[selected_data[variable] < 0, variable] = 0
# Renaming columns
new_column_names = {
    'wd': 'Wind Direction',
    'ws': 'Wind Speed',
    't': 'Temperature',
    'q': 'Specific Humidity',
    'hourly_rain': 'Hourly Rainfall',
    'p': 'Pressure',
    'n': 'Cloud Coverage',
    'rh': 'Relative Humidity'
}

selected_data.rename(columns=new_column_names, inplace=True)
met_vars = [new_column_names[var] for var in met_vars_initial]
def prepare_data_for_training(selected_data, vars, variable):
    predictors = selected_data[vars]
    variable_data = selected_data[variable]

    train_mask = selected_data['date'].dt.year <= 2017
    test_mask = selected_data['date'].dt.year > 2017

    X_train, y_train = predictors[train_mask], variable_data[train_mask]
    X_test, y_test = predictors[test_mask], variable_data[test_mask]

    t_train = selected_data['date'][train_mask]
    t_test = selected_data['date'][test_mask]

    return X_train, y_train, t_train, X_test, y_test, t_test

X_train, y_train, t_train, X_test, y_test, t_test = prepare_data_for_training(
    selected_data, met_vars, variable)
plt.figure(dpi=DPI, figsize=(10, 4))
plt.scatter(t_train, y_train, label='Train', s=0.1)
plt.scatter(t_test, y_test, label='Test', s=0.1)
plt.grid()
plt.legend()
plt.xlabel('Date')
plt.ylabel(r'[$\text{NO}_2$] ($\mu g/\text{m}^3$)')
def RF_optimization(X_train, y_train, n_iter=20, random_state=None):
```

```
    # Define the hyperparameter grid
    param_dist = {
        'n_estimators': randint(20, 200),
        'min_samples_leaf': randint(3, 10),
        'max_samples': uniform(0, 0.9),
        'max_features': uniform(0, 1),
        'max_depth': randint(2, 20),
    }

    # Create the RandomForestRegressor
    rf_model = RandomForestRegressor(random_state=random_state)

    # Create the RandomizedSearchCV instance
    random_search = RandomizedSearchCV(
        estimator=rf_model,
        param_distributions=param_dist,
        n_iter=n_iter,  # Number of random combinations to try
        cv=5,       # Number of cross-validation folds --> no separate validation
            data needed
        n_jobs=-1,  # Use all available CPU cores
        random_state=random_state
    )

    # Fit the RandomizedSearchCV instance to your data
    random_search.fit(X_train, y_train)

    # Get the best hyperparameters
    best_params = random_search.best_params_
    print("Best Hyperparameters:")
    print(best_params)

    # Get the best model
    best_rf_model = random_search.best_estimator_

    return best_rf_model, best_params

best_rf_model, best_params = RF_optimization(X_train, y_train, n_iter=N_iter)

# Make predictions with the best model
y_train_pred = best_rf_model.predict(X_train)
y_test_pred = best_rf_model.predict(X_test)
def check_tree_number(params, N_tree=11, n_runs=1, plot=True, random_state=None):
    n_tree_test_arr = [2**n for n in range(N_tree)]
    train_rmse_list = []
    test_rmse_list = []

    for n_trees_test in n_tree_test_arr:
        train_rmse_accumulator = 0
        test_rmse_accumulator = 0

        for _ in range(n_runs):
            test_params = params.copy()
            test_params['n_estimators'] = n_trees_test

            # Train a model using the test_params
```

```
            rf_test_model = RandomForestRegressor(**test_params, random_state=
                random_state)
            rf_test_model.fit(X_train, y_train)

            # Make predictions
            y_train_pred = rf_test_model.predict(X_train)
            y_test_pred = rf_test_model.predict(X_test)

            # Calculate RMSE
            train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
            test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

            # Accumulate RMSE for averaging
            train_rmse_accumulator += train_rmse
            test_rmse_accumulator += test_rmse

        # Calculate the average RMSE and append it to the results list
        train_rmse_list.append(train_rmse_accumulator / n_runs)
        test_rmse_list.append(test_rmse_accumulator / n_runs)

    if plot:
        plt.figure(figsize=(10, 6), dpi=DPI)
        plt.plot(n_tree_test_arr, train_rmse_list, label='Training RMSE', marker='
            o')
        plt.plot(n_tree_test_arr, test_rmse_list, label='Test RMSE', marker='o')
        plt.xscale('log')  # because n_trees_test are powers of 2
        plt.xlabel('Number of Trees')
        plt.ylabel('RMSE ($\mu g/\text{m}^3$)')
        plt.legend()
        plt.grid(True)
        plt.show()

    return n_tree_test_arr, train_rmse_list, test_rmse_list

if True:
    check_tree_number(best_params, N_tree=11, n_runs=10);
def plot_model_pred(y_train, y_train_pred, y_test, y_test_pred):
    fig, ax1 = plt.subplots(1, 1, dpi=DPI, figsize=(10, 5))

    # Scatter plots
    size = 1
    transp = 1
    ax1.scatter(y_train, y_train_pred, label='Train', color='tab:blue', s=size,
        alpha=transp)
    ax1.scatter(y_test, y_test_pred, label='Test', color='tab:orange', s=size,
        alpha=transp)

    # 1:1 line
    line = np.array([0, np.max([np.max(y_train_pred), np.max(y_test_pred)])])
    ax1.plot(line, line, zorder=5, linestyle='dashed', color='black', label='1:1
        line', alpha=.7)
    ax1.set_aspect('equal', 'box')

    ax1.set_xlabel(r'Measured [$\text{NO}_2$] ($\mu g/\text{m}^3$)')
    ax1.set_ylabel(r'Predicted [$\text{NO}_2$] ($\mu g/\text{m}^3$)')
    ax1.legend()
```

```
    ax1.grid()

    plt.show()
plot_model_pred(y_train, y_train_pred, y_test, y_test_pred)
# Your function
def plot_model_pred_timeseries(t_train, y_train, y_train_pred, t_test, y_test,
    y_test_pred, all=True, start_month=1, n_months=3, start_day=1, n_days=3,
    residuals=False):
    if not residuals:
        fig, (ax1, ax2) = plt.subplots(2, 1, dpi=DPI, figsize=(10, 4), sharey=True
            )
    else:
        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, 1, dpi=DPI, figsize=(10,
            10))

    ax1.plot(t_train, y_train, alpha=.5, label='Observation', color='tab:blue')
    ax1.plot(t_train, y_train_pred, alpha=.5, label='Prediction', color="tab:
        purple")
    ax1.set_ylim(0, 120)
    ax1.legend(loc='upper right')
    ax1.grid(True)
    ax1.set_ylabel(r'[$\text{NO}_2$] ($\mu g/\text{m}^3$)')
    ax1.set_title('Training Set')

    ax2.plot(t_test, y_test, alpha=.5, label='Observation', color='tab:orange')
    ax2.plot(t_test, y_test_pred, alpha=.5, label='Prediction', color="tab:red")
    ax2.grid(True)
    ax2.set_xlabel('Date')
    ax2.set_ylabel(r'[$\text{NO}_2$] ($\mu g/\text{m}^3$)')
    ax2.legend(loc='upper right')
    ax2.set_title('Test Set')

    if not all:
        ax1.set_xlim(datetime.date(2017, start_month, start_day), datetime.date
            (2017, start_month + n_months, start_day + n_days))
        ax2.set_xlim(datetime.date(2018, start_month, start_day), datetime.date
            (2018, start_month + n_months, start_day + n_days))

    if residuals:
        # Calculate residuals
        train_residuals = y_train - y_train_pred
        test_residuals = y_test - y_test_pred

        # Residuals plot
        ax3.plot(t_train, train_residuals, alpha=.5, label='Train Residuals')
        ax3.plot(t_test, test_residuals, alpha=.5, label='Test Residuals')
        ax3.grid(True)
        ax3.legend()
        ax3.set_ylabel(r'Residuals ($\mu g/\text{m}^3$)')

        # Calculate and plot ACF, then perform Fourier analysis
        autocorr = acf(train_residuals, nlags=24*365, fft=True) # ACF calculation
        spectrum = np.fft.fft(autocorr)  # FFT of ACF
        freq = np.fft.fftfreq(len(spectrum))

        # ACF plot of residuals
```

```
        x = np.arange(len(autocorr)) / 24 # Convert lag to days
        ax4.plot(x, autocorr)
        ax4.set_xlabel('Lag [days]')
        ax4.set_ylabel('Autocorrelation')
        ax4.grid(True)
        ax4.set_yscale('log')
        ax4.set_ylim(1e-3, 1)

        # For visualization, usually just half of the spectrum is enough
        pos_mask = np.where(freq > 0)
        freqs = freq[pos_mask]
        periods = 1 / freqs # Convert frequency to period in days
        power = np.abs(spectrum[pos_mask])**2
        power_normalized = power / np.sum(power)

        ax5.loglog(periods, power_normalized)  # Plot the power spectrum of ACF
        ax5.set_xlabel('Period [days]')
        ax5.set_ylabel('Power')
        ax5.grid(True)

    plt.tight_layout()
    plt.show()


plot_model_pred_timeseries(t_train, y_train, y_train_pred, t_test, y_test,
    y_test_pred, all=True)

plot_model_pred_timeseries(t_train, y_train, y_train_pred, t_test, y_test,
    y_test_pred, all=False, start_month=1, n_months=3, start_day=1, n_days=0)
def get_RF_feature_imp(model, X_train, plot=False, use_permutation=False, target=
    None, random_state=None, decimals=2, relative_threshold = 0.1):
    '''Calculates feature importance using Gini importance or permutation
        importance.

    Args:
        model (object): The trained Random Forest model.
        X_train (DataFrame): The training data features.
        plot (bool, optional): Whether to plot the feature importances. Default is
             False.
        use_permutation (bool, optional): If True, calculate permutation
            importance.
            If False (default), calculate Gini importance.
        random_state (int or None, optional): Random seed for permutation
            importance. Default is None.

    Returns:
        DataFrame: A DataFrame containing feature importances.
    '''
    if use_permutation:
        result = permutation_importance(model, X_train, target, random_state=
            random_state)
        importances_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
            result.importances_mean})
    else:
        feature_importances = model.feature_importances_
```

```
        importances_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})

    importances_df = importances_df.sort_values(by='Importance', ascending=False)

    if plot:
        plt.figure(figsize=(10, 6))  # Set figure size
        bars = plt.barh(importances_df['Feature'], importances_df['Importance'], color='skyblue')
        plt.xlabel('Importance')
        plt.ylabel('Feature')
        plt.gca().invert_yaxis()  # Invert y-axis to have the highest importance at the top
        plt.grid(axis='x')  # Add a grid along the x-axis
        plt.tight_layout()  # Adjust layout to prevent clipping of labels
        plt.xscale('log')

        # Annotating the bars
        thresh = relative_threshold * importances_df.max().iloc[1]
        for bar in bars:
            # If bar width is smaller than a threshold, adjust text position and alignment
            if bar.get_width() < thresh:  # example threshold
                plt.annotate(f"{bar.get_width():.{decimals}f}",
                             xy=(bar.get_width(), bar.get_y() + bar.get_height()/2),
                             xytext=(5, 0),  # adjust the position of text to avoid overlap with bars
                             textcoords="offset points",
                             ha='left', va='center',
                             fontsize=8, color='black')
            else:
                plt.annotate(f"{bar.get_width():.{decimals}f}",
                             xy=(bar.get_width(), bar.get_y() + bar.get_height()/2),
                             xytext=(-5, 0),  # adjust the position of text to avoid overlap with bars
                             textcoords="offset points",
                             ha='right', va='center',
                             fontsize=8, color='black')

        plt.show()

    return importances_df


get_RF_feature_imp(best_rf_model, X_train, plot=True, use_permutation=False, decimals=2)
def get_RF_stats(y_train, y_test, y_train_pred, y_test_pred):
    rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

    correlation_train, _ = pearsonr(y_train, y_train_pred)
    correlation_test, _ = pearsonr(y_test, y_test_pred)

    explained_variance_train = explained_variance_score(y_train, y_train_pred)
```

```
    explained_variance_test = explained_variance_score(y_test, y_test_pred)

    mae_train = mean_absolute_error(y_train, y_train_pred)
    mae_test = mean_absolute_error(y_test, y_test_pred)

    results = {
        'RMSE Train': rmse_train,
        'RMSE Test': rmse_test,
        'Correlation Train': correlation_train,
        'Correlation Test': correlation_test,
        'Explained Variance Train': explained_variance_train,
        'Explained Variance Test': explained_variance_test,
        'MAE Train': mae_train,
        'MAE Test': mae_test
    }

    latex_table = "\\begin{tabular}{lcc}\n\\toprule\n"
    latex_table += "Metric & Train & Test \\\\ \\midrule\n"

    metrics_train = [key for key in results.keys() if "Train" in key]
    metrics_test = [key.replace("Train", "Test") for key in metrics_train]

    for m_train, m_test in zip(metrics_train, metrics_test):
        unit = " ($\\mu g/m^3$)" if "RMSE" in m_train or "MAE" in m_train else ""
        latex_table += f"{m_train.replace(' Train', '')}{unit} & {results[m_train
            ]:.2f} & {results[m_test]:.2f} \\\\ \n"

    latex_table += "\\bottomrule\n\\end{tabular}"

    print(latex_table)

    return results


RF_stats_1 = get_RF_stats(y_train, y_test, y_train_pred, y_test_pred);
# Improved RF 1
# add extra explainers
selected_data_2 = data_imported[met_vars_initial + ['date'] + [variable]].dropna()
selected_data_2.rename(columns=new_column_names, inplace=True)
selected_data_2.loc[selected_data_2[variable] < 0, variable] = 0
def is_winter_time(row):
    year = row['date'].year
    # DST starts on the last Sunday of March
    dst_start = max([day for day in pd.date_range(f"{year}-03-01", f"{year}-03-31"
        ) if day.weekday() == 6])
    # DST ends on the last Sunday of October
    dst_end = max([day for day in pd.date_range(f"{year}-10-01", f"{year}-10-31")
        if day.weekday() == 6])
    # Return 1 if date is outside DST period, 0 otherwise
    return 1 if row['date'] < dst_start or row['date'] > dst_end else 0

def get_extra_explainers(data):
    # add more variables that could explain emissions
    dutch_holidays = [
        datetime.date(datetime.MINYEAR, 1, 1),   # New Year's Day
        datetime.date(datetime.MINYEAR, 4, 27),  # King's Day
```

```
        datetime.date(datetime.MINYEAR, 5, 5),   # Liberation Day
        datetime.date(datetime.MINYEAR, 5, 25),  # Ascension Day
        datetime.date(datetime.MINYEAR, 6, 5),   # Whit Monday
        datetime.date(datetime.MINYEAR, 12, 25), # Christmas Day
        datetime.date(datetime.MINYEAR, 12, 26), # Boxing Day
    ]

    data['Hour of Day'] = data['date'].dt.hour
    data['Day of Year'] = data['date'].dt.dayofyear
    data['Day of Week'] = data['date'].dt.dayofweek
    data['day_of_month'] = data['date'].dt.day
    data['week_of_year'] = data['date'].dt.isocalendar().week

    data['month_day'] = data['date'].dt.strftime('%m-%d')
    data['holiday'] = data['month_day'].isin(date.strftime('%m-%d') for date in
        dutch_holidays).astype(int)
    data.drop(columns=['month_day'], inplace=True)
    data['holiday']

    data['Winter Time'] = data.apply(is_winter_time, axis=1)
    data['working_day'] = ((data['Day of Week'] >= 0) & (data['Day of Week'] <= 4)
        ).astype(int)
    return data

extra_explainers = ['Hour of Day', 'Day of Year', 'Day of Week']#, 'working_day',
    'week_of_year', 'holiday', 'day_of_month', 'Winter Time]
get_extra_explainers(selected_data_2);
X_train_2, y_train_2, t_train_2, X_test_2, y_test_2, t_test_2 =
    prepare_data_for_training(selected_data_2, met_vars + extra_explainers,
    variable)
best_rf_model_2, best_params_2 = RF_optimization(X_train_2, y_train_2, n_iter=
    N_iter)

# Make predictions with the best model
y_train_pred_2 = best_rf_model_2.predict(X_train_2)
y_test_pred_2 = best_rf_model_2.predict(X_test_2)
plot_model_pred(y_train_2, y_train_pred_2, y_test_2, y_test_pred_2)
plot_model_pred_timeseries(t_train_2, y_train_2, y_train_pred_2, t_test_2,
    y_test_2, y_test_pred_2, all=True)
plot_model_pred_timeseries(t_train_2, y_train_2, y_train_pred_2, t_test_2,
    y_test_2, y_test_pred_2, all=False, start_month=1, n_months=3, start_day=1,
    n_days=0)
print(get_RF_feature_imp(best_rf_model_2, X_train_2, plot=True, decimals=2,
    relative_threshold=0.05))
RF_stats_2 = get_RF_stats(y_train_2, y_test_2, y_train_pred_2, y_test_pred_2)
def compare_rf_stats_relative(RF_stats_1, RF_stats_2):
    stat_names = [
    'RMSE Train',
    'RMSE Test',
    'Correlation Train',
    'Correlation Test',
    'Explained Variance Train',
    'Explained Variance Test',
    'MAE Train',
    'MAE Test'
    ]
```

```
    RF1_stats = np.array([RF_stats_1[name] for name in stat_names])
    RF2_stats = np.array([RF_stats_2[name] for name in stat_names])

    # Calculating the relative differences
    relative_differences = ((RF2_stats - RF1_stats) / RF1_stats) * 100

    # Stripping " Train" and " Test" from labels for x-axis
    labels = [name.replace(" Train", "").replace(" Test", "") for name in
        stat_names]

    # Get unique labels
    unique_labels = []
    [unique_labels.append(label) for label in labels if label not in unique_labels
        ]

    x = np.arange(len(unique_labels))  # the label locations
    width = 0.35  # the width of the bars

    fig, ax = plt.subplots(figsize=(10, 6))
    plt.grid(True)

    # Differentiating between train and test stats by color
    train_color = 'tab:blue'
    test_color = 'tab:orange'
    colors = [train_color if 'Train' in name else test_color for name in
        stat_names]

    # Plotting Train and Test bars side by side
    for i, (label, diff, color) in enumerate(zip(labels, relative_differences,
        colors)):
        bar = ax.bar(x[i//2] + (i%2) * width, diff, width, color=color)

        # Adding percentage above the bars
        height = bar[0].get_height()
        ax.annotate(f'{diff:.1f}%',
                    xy=(bar[0].get_x() + bar[0].get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

    # Adding a zero line
    ax.axhline(0, color='black', linewidth=0.8)

    ax.set_xlabel('Metric')
    ax.set_ylabel('Relative Difference (%)')
    ax.set_xticks(x + width/2)  # position x-axis labels in the center of grouped
        bars
    ax.set_xticklabels(unique_labels)

    # Creating a custom legend
    legend_elements = [Patch(facecolor=train_color, label='Train'),
                       Patch(facecolor=test_color, label='Test')]
    ax.legend(handles=legend_elements)

    fig.tight_layout()
```

```
    plt.show()

compare_rf_stats_relative(RF_stats_1, RF_stats_2)
# Improved RF 2
# add extra components
comp_vars = ['o3', 'ox']
selected_data_3 = data_imported[met_vars_initial + comp_vars + ['date'] + [
    variable]].dropna()
selected_data_3.rename(columns=new_column_names, inplace=True)
selected_data_3.loc[selected_data_3[variable] < 0, variable] = 0
get_extra_explainers(selected_data_3)
X_train_3, y_train_3, t_train_3, X_test_3, y_test_3, t_test_3 =
    prepare_data_for_training(selected_data_3, met_vars + comp_vars +
    extra_explainers, variable)
best_rf_model_3, best_params_3 = RF_optimization(X_train_3, y_train_3, n_iter=
    N_iter)

# Make predictions with the best model
y_train_pred_3 = best_rf_model_3.predict(X_train_3)
y_test_pred_3 = best_rf_model_3.predict(X_test_3)
plot_model_pred(y_train_3, y_train_pred_3, y_test_3, y_test_pred_3)
plot_model_pred_timeseries(t_train_3, y_train_3, y_train_pred_3, t_test_3,
    y_test_3, y_test_pred_3, all=True)
plot_model_pred_timeseries(t_train_3, y_train_3, y_train_pred_3, t_test_3,
    y_test_3, y_test_pred_3, all=False, start_month=1, n_months=3, start_day=1,
    n_days=0)
print(get_RF_feature_imp(best_rf_model_3, X_train_3, plot=True, decimals=4,
    relative_threshold=0.0001))
RF_stats_3 = get_RF_stats(y_train_3, y_test_3, y_train_pred_3, y_test_pred_3)
compare_rf_stats_relative(RF_stats_1, RF_stats_3)
```