

GLOBAL DEATHS PREDICTION CAUSED BY PM AIR POLLUTANTS



IACEC REPORT

MASTER'S DEGREE IN DATA SCIENCE & ENGINEERING
FEUP

AUTHORS: CÁTIA TEIXEIRA (UP200808037) AND SILVIA
TAVARES (UP202204392)

Table of Contents

Table index	3
Figure index.....	3
A1. Business Understanding.....	4
Determine business objective.....	4
Assess situation	6
Determine data mining goals	7
Produce project plan	8
A2. Data Understanding.....	9
Collect initial data.....	9
Describe data	10
Explore data	10
Verify data quality	16
A3. Data Preparation.....	17
Select data.....	17
Clean data.....	17
Construct data.....	18
Integrate data.....	18
Format data.....	18
A4. Modeling	19
Select modeling technique.....	19
Generate test design	25
Build model	26
Assess model	35
A5. Evaluation	36
Evaluate results	36
Review process.....	36
Determine next steps.....	37
References.....	38
Appendix	39

Table index

Table 1 - Project assessment summary table.....	6
Table 2 - PM _{2.5} statistical summary	11
Table 3 - PM ₁₀ statistical summary.....	12
Table 4 - PM deaths statistical summary	13
Table 5 - Years statistical summary.....	14
Table 6 - Summary table for the mean of predictive performance measures for each model ..	28
Table 7 - Summary table for best folds for the best scoring candidate models	28
Table 8 - Random Forest parameter variation for grid search.....	29
Table 9 - Random forest best hyper-parameters.....	29
Table 10 - GBoost parameter variation for grid search	29
Table 11 - GBoost best hyper-parameters.....	30
Table 12 - XG Boost parameter variation for grid search	30
Table 13 - XGBoost best hyper-parameters.....	30
Table 14 - Validation scores for the best hyper-parameters	30
Table 15 - Results with test set	31
Table 16 - Feature importance comparison RF, GB and XGB.....	32
Table 17 - Predictions vs True Values comparison RF, GB and XGB	33
Table 18 – Gboost Results.....	36

Figure index

Figure 1 - PM _{2.5} histogram	11
Figure 2 - PM ₁₀ histogram	12
Figure 3 - PM deaths histogram	13
Figure 4 - Pairs scatter plots.....	14
Figure 5 – Correlation Matrix	16
Figure 6 - Data before and after standardization.....	18
Figure 7 - Test design process flow	26
Figure 8 - Validation results for R ² in different models with standard hyper-parameters	27
Figure 9 - Validation results for Root Mean Square Error in different models with standard hyper-parameters	27
Figure 10 - Mean decrease in impurity (MDI) vs Permutation based importance in Random Forest	32
Figure 11 - Mean decrease in impurity (MDI) vs Permutation based importance in GBoost.....	32
Figure 12 - Mean decrease in impurity (MDI) vs Permutation based importance in XGBoost...	32
Figure 13 - RF Predictions vs True Values	33
Figure 14 - GB Predictions vs True Values.....	33
Figure 15 - XGB RF Predictions vs True Values.....	33
Figure 16 - RF distribution of residuals	34
Figure 17 - GB distribution of residuals.....	34
Figure 18 - XGB distribution of residuals.....	34

A1. Business Understanding

“Around the world, nine out of every ten people breathe unclean air. We need dramatic and systemic change. Reinforced environmental standards, policies and laws that prevent emissions of air pollutants are needed more than ever.” – António Guterres

Determine business objective

Air pollution is one of the main reasons for a high number of premature deaths and increased morbidity rates in the world. According to World Health Organization (WHO), in 2012 seven million premature deaths were caused by air pollution (WHO, 2014). A number that has not decreased from 2012 to this day (WHO, 2022). This includes exposure to both indoor and outdoor contaminants.

Particulate matter (PM) is one of the main constituents of air pollution and may cause a vast number of negative consequences. There is consistent evidence that the levels of fine particulate matter in the air are associated with the risk of death from all causes and from cardiovascular and respiratory illness. (Jonathan M. Samet, 2000). A number of adverse effects on public health, especially on children and the elderly may arise in the exposure to this air pollutant. Respiratory tract complications like bronchitis, pneumonia and allergic reactions are some of the consequences associated with PM (Raaschou-Nielsen, 2010). It has become of special importance the study of PM implications on public health.

Before going further, it is important to mention the meaning of Particulate Matter. PM is common physical classification of particles found in the air, such as dust, dirt, soot, smoke or liquid droplets. Since most particles are not spherical, PM diameter are often described using equivalent diameter, i.e. the diameter of a sphere that would have the same fluid properties (Vallero, D., 2007).

PM_{2.5} refers to particles with an equivalent diameter below or equal to 2.5 µm. The widths of the larger particles in the PM_{2.5} size range would be about thirty times smaller than that of a human hair. There are outdoor and indoor sources of these particles. Outside, fine particles primarily come from car, truck, bus and off-road vehicle (e.g., construction equipment, snowmobile, locomotive) exhausts, other operations that involve the burning of fuels such as wood, heating oil or coal and natural sources such as forest and grass fires. Fine particles also form from the reaction of gases or droplets in the atmosphere from sources such as power plants. These chemical reactions can occur miles from the original source of the emissions, since such small particles can be suspended in the air and travel long distances (Vallero, D., 2007).

PM_{2.5} is also produced by common indoor activities. Some indoor sources of fine particles are tobacco smoke, cooking (e.g., frying, sautéing, and broiling), burning candles or oil lamps, and operating fireplaces and fuel-burning space heaters (e.g., kerosene heaters).

PM₁₀ describes particles with diameters that are equal or smaller to 10 µm. These particles are inhalable and considered to pose the greatest risk to health as they manage to penetrate deep into the lungs and reach the pulmonary alveoli, causing disturbances in the respiratory system. They can be emitted directly into the air (primary particles) or be formed in the atmosphere by gaseous precursors such as sulfur dioxide, nitrogen oxide, ammonia and volatile non-methane organic components (secondary particles). PM₁₀, with prolonged exposure, accumulate in the pulmonary alveoli causing a reaction of pulmonary overload and a decrease in respiratory capacity (Vallero, D., 2007).

PM₁₀ has the ability to act as an obstacle to the diffusion of oxygen across the pulmonary membrane. Some possible sources of PM₁₀ are combustion plants, agricultural activities, wood treatment, metallurgy, civil construction, refineries, mining, industrial activities, etc (EPA, 2022).

Recent studies even suggest there is a link between PM concentration and COVID-19's morbidity and mortality spread and increase. PM mechanism of action on cells is to trigger an inflammatory state which could lead to a more complicated course of the SARS-CoV-2 pathology. There is an apparent positive correlation between COVID-19, the PM concentration and the high mortality rate in some polluted areas (Comunian, S. et al, 2020).

Scientific studies, like the ones referenced above follow an empirical method to acquire knowledge based on observational and experimental data. Data is therefore, the basis of all scientific knowledge.

The analysis of data is the science to analyze crude data and to extract patterns from them. Data science is concerned with the creation of models able to extract patterns from complex data and the use of these models in real-life problems, with the support of suitable technologies (Moreira, J. et al, 2018).

The current computerization of our society as fast development of powerful data collection led to the explosive growth of available data in the past decades. This widely available and gigantic body of data makes this time truly the data age. It urges to turn this data into organized knowledge, by the use of versatile tools that can uncover valuable information from tremendous amounts of data (Han, J. et al, 2012).

Given the information above, the objective of this project is by the use of suitable machine learning and data mining techniques and methodologies, arrive at a model that can

effectively predict the number of deaths caused by PM pollutants, with using airborne PM concentration data.

Assess situation

The assessment for this project resulted in the following outputs:

Inventory of resources

The data repository for choosing the relevant datasets was Kaggle.

For this project, mainly Jupyter Notebook software was used (Python language). Also, an experimental phase was conducted using Rapid Miner but turns out that due the innumerable number of models that we tried to apply it was visibly not easy to identify the relationship between each node so we decided to perform the project by doing everything in Python.

Project assessment

The following table describes the project assessment summary.

Table 1 - Project assessment summary table

Requirements	Assumptions
<ul style="list-style-type: none">• Develop a data mining project by following an appropriate methodology;• Clearly understand a processing methods and learning algorithms;• Clearly understand model evaluation methodologies and evaluation measures;• Present the results in a clear synthetic way.	<ul style="list-style-type: none">• Create a schedule time for the project and estimate a start and end time in order to perform the CRISP-DM steps in a more efficient way to complete the project on time;• Evaluate carefully the quality of the prediction made by each model in order to choose the best one.

Constrains	Risks and Contingency
<ul style="list-style-type: none"> • Availability of quality data; • Size of dataset; • Usability of data; • Computational limitations for time consuming hyper parameters' grid search. 	<ul style="list-style-type: none"> • Project might take longer than anticipate; • Data can have poor quality or a lot of noise; • Data not available within the expected timeline for all the instances; • The result provided by each model for this specific dataset might not be the best.

Costs and benefits

Estimated Costs:

- By implementing different algorithms, it can take a lot of time to find the best hyper-parameters;
- Programming knowledge is required to build the models;
- Theoretical knowledge is also required to analyze the results and evaluate the errors.

Estimated Benefit:

- Good results about the predictions;
- Additional insights about the data exploration and understanding;
- Better understanding on the tools available of Python to perform predictive model;
- Generate action on the day by day in order to increase air quality.

Determine data mining goals

By predicting the number of deaths caused by PM pollutants, with using airborne PM concentration data and total deaths by air pollution the goal is to make people aware of the consequences of air pollution and to encourage the whole community to adopt solutions to avoid these deaths. Investing in cleaner forms of heating, mobility, agriculture and industry brings better health, productivity and quality of life to all, and especially the most vulnerable.

These investments save lives and accelerate progress towards carbon neutrality and robust biodiversity.

In order to assure that the prediction is good, we decided to establish a success criterion that is based in R^2 minimum of 0.65.

Since R^2 statistic can be misleading if the case of nonlinear models (Aggarwal, C., 2015), additional predictive performance measures were used:

- Mean Absolute error (MAE) that represents the average of the absolute difference between the actual and predicted values in the dataset. It measures the average of the residuals in the dataset;
- Mean Absolute Percentage Error (MAPE), also called the mean absolute percentage deviation (MAPD) measures accuracy of a forecast system. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. The mean absolute percentage error (MAPE) is the most common measure used to forecast error, probably because the variable's units are scaled to percentage units, which makes it easier to understand;
- Mean Squared Error (MSE) that is the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals;
- Root Mean Squared Error (RMSE) that refers to the square root of Mean Squared error. It measures the standard deviation of residuals.

The lower value of MAE, MSE, and RMSE implies higher accuracy of a regression model. However, a higher value of R square is considered desirable.

Produce project plan

The implementation plan to achieve our goal starts by the data extraction where we check the availability of the data and the timeframe / years that we have in the different datasets. We intend to use 15% of the total time on this task.

After extracting the data the plan is to load the dataset in python by importing pandas and applying instructions to read the files and get a first insight of all the dataset by checking on the missing data, noisy and outlier detection. Next, we will combine the datasets, perform calculations until arriving to the final structure. Following that, starts the calculation of the

descriptive statistics of all the instances by importing from pandas profiling that automatically gives information such as:

- Quantile Statistics - including Minimum Value, Q1, Median, Q3, Maximum, Range, Interquartile Range;
- Descriptive Statistics - Mean, Mode, Standard Deviation, Sum, Median Absolute Deviation, Coefficient of Variation, Kurtosis, Skewness ;
- Most frequent and extreme value;
- Histogram;
- Correlations – Spearman, Pearson, Kendall, Phik;
- Missing Value – Calculations through counts, matrix and heatmaps.

The goal is to apply 10% of the time on the task above.

The next step is to start building the different Models, find the best hyper-parameters and evaluate the performance-based R^2 , Mean Squared Error, Mean Absolute Percentage Error, Root Mean Squared Error and Mean Absolute Error. These tasks will take 75% of the time.

A2. Data Understanding

Collect initial data

In order to create our final dataset we start by first collecting total Air pollution deaths from 1990 to 2017 in Kaggle. After, we decide to look after more indicators about air pollutants.

Our second dataset has indicators about the level of air pollution around the globe. In the dataset about death, as our main focus in this study is to predict deaths by particular matter, we select the in this dataset the instance representing deaths due to particulate matter, total deaths, country, and year from 2000 until 2017 so it matches to the time frame in the dataset about particular matters. PM deaths and total deaths are given as a rate of the number of deaths per 100,000 in each country.

On the second dataset that contains the level of air pollution we originally had 10 instances named:

- Country Name – that includes name of the countries;
- City – name of the town;

- Year – the year in which the data is;
- PM2.5 ($\mu\text{g}/\text{m}^3$) – the particular matter 2.5 pollution level in micrograms per cubic meter;
- PM10 ($\mu\text{g}/\text{m}^3$) – the particular matter 10 pollution level in micrograms per cubic meter;
- NO2 ($\mu\text{g}/\text{m}^3$) – the nitrogen dioxide in micrograms per cubic meter;
- PM2.5 temporal coverage (%) – the particular matter 2.5 pollution level in percentage;
- PM10 temporal coverage (%) – the particular matter 10 pollution level in percentage;
- NO2 temporal coverage (%) – the nitrogen dioxide level in percentage;
- Updated year – year in which the data was updated.

The dataset about level of air pollution indexes had also redundant information that is not related to our predictive goal. So, for that we select only the year, country, PM2.5 ($\mu\text{g}/\text{m}^3$) and PM10 ($\mu\text{g}/\text{m}^3$).

Describe data

After, we combined both dataset, we kept with year from 2000 to 2017, total deaths, total deaths due to ambient particular matters, average of PM2.5 ($\mu\text{g}/\text{m}^3$) in all the cities of each country, average of PM10 ($\mu\text{g}/\text{m}^3$) in all the cities of each country and the country itself. We use the average of the PM_{2.5} and PM₁₀ by country (average of all cities) given the dataset one only had deaths by country. This step was mandatory so we can merge the two datasets into one. So, the final dataset contains 333 observations and 5 attributes, where country is nominal and the rest are numerical data.

Explore data

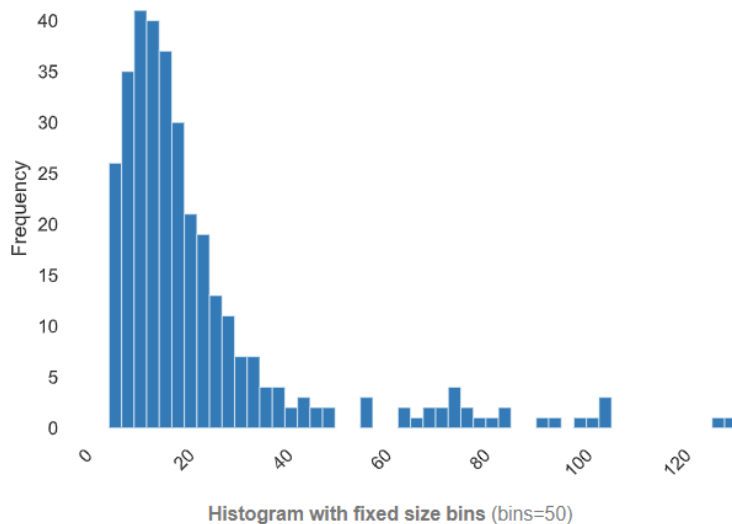
PM2.5

In order to know more about the values of PM2.5 that we had in our dataset, the calculate some statistics measures that are in the table below.

Table 2 - $PM_{2.5}$ statistical summary

Quantile statistics		Descriptive statistics	
Minimum	4.5785	Standard deviation	21.3632
5-th percentile	6.5673	Coefficient of variation (CV)	0.9189
Q1	11.2567	Kurtosis	6.9289
Median	16.46	Mean	23.1410
Q3	25	Median Absolute Deviation (MAD)	6.435
95-th percentile	75.04	Skewness	2.5395
Maximum	132	Sum	7705.98
Range	127.4215	Variance	452.1261
Interquartile Range (IQR)	13.743	Monotonicity	Not mono.

Also, we created a histogram to visualize the distribution of $PM_{2.5}$. On the x-axis we have the bins that we created based on the frequency values.

Figure 1 - $PM_{2.5}$ histogram

From the above visualization we can see the data is highly right skewed, with the biggest concentration of frequencies in the smallest values of x-axis.

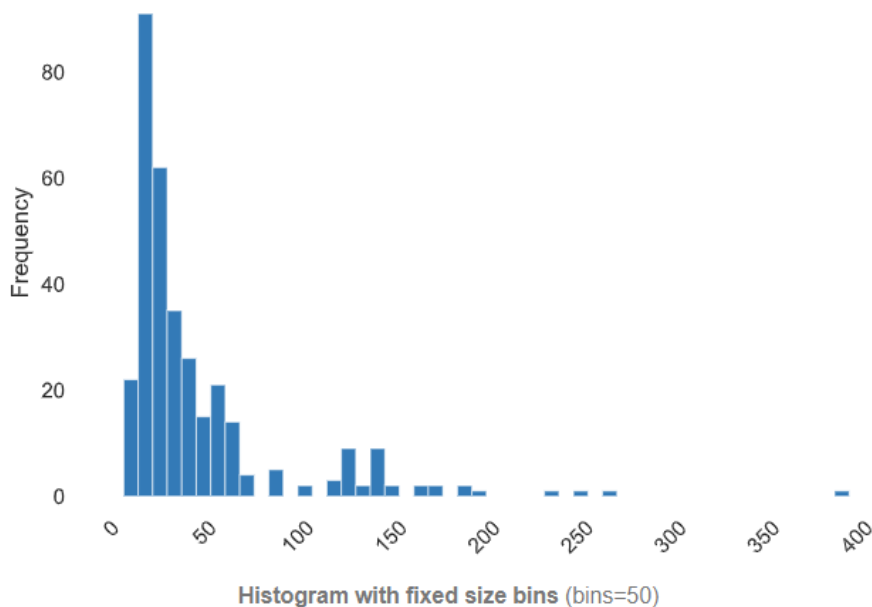
PM10

In order to have a better understanding on the statics of this instances, we created the following table.

Table 3 - PM_{10} statistical summary

Quantile statistics		Descriptive statistics	
Minimum	6.03	Standard deviation	46.6947
5-th percentile	13.057	Coefficient of variation (CV)	1.0282
Q1	19.1267	Kurtosis	12.8001
Median	27.8625	Mean	45.4146
Q3	51.3	Median Absolute Deviation (MAD)	10.8754
95-th percentile	143.124	Skewness	3.0183
Maximum	395.3333	Sum	15123.1
Range	390.3033	Variance	210.4009
Interquartile Range (IQR)	32.1733	Monotonicity	Not mono.

Also, we created a histogram to visualize the distribution of PM_{10} .

Figure 2 - PM_{10} histogram

From the above visualization we can see the data is highly right skewed, with the biggest concentration of frequencies in the smallest values of x-axis.

PM deaths

PM deaths represent the total death by particular matters and has the following statistics measures.

Table 4 - PM deaths statistical summary

Quantile statistics		Descriptive statistics	
Minimum	7.5444	Standard deviation	14.3139
5-th percentile	9.1631	Coefficient of variation (CV)	0.5468
Q1	16.0333	Kurtosis	0.5866
Median	22.4306	Mean	26.1778
Q3	33.5053	Median Absolute Deviation (MAD)	8.6160
95-th percentile	54.0200	Skewness	1.0411
Maximum	73.7233	Sum	8717.208
Range	66.1789	Variance	204.8877
Interquartile Range (IQR)	17.4719	Monotonicity	Not mono.

Also, we created a histogram to visualize the distribution of PM deaths.

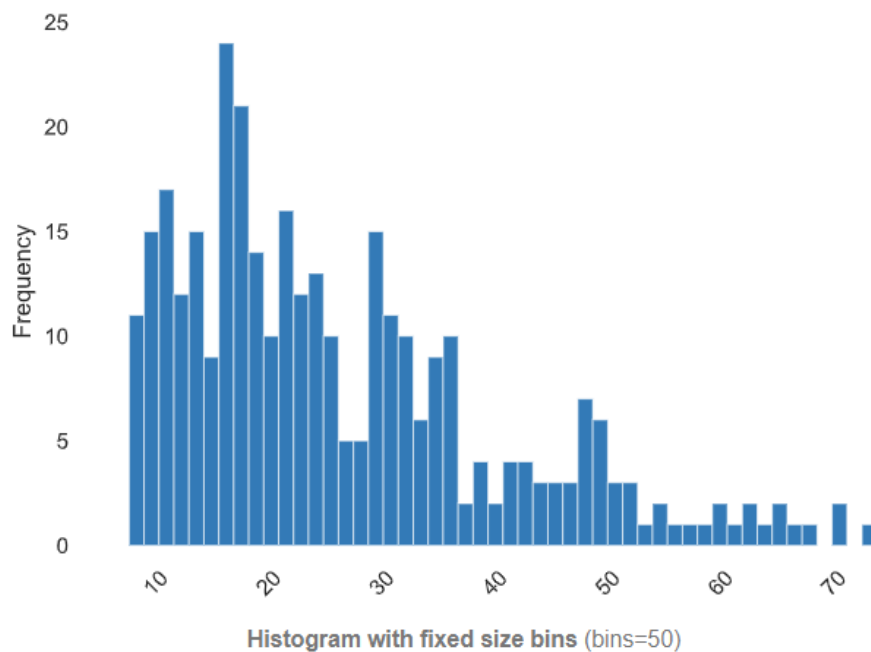


Figure 3 - PM deaths histogram

From the above visualization we can see the data is slightly right skewed, with the majority concentration of frequencies in the smallest values of x-axis.

Years

The following table as the descriptive statistic for the year.

Table 5 - Years statistical summary

Quantile statistics		Descriptive statistics	
Minimum	2010	Standard deviation	2.1961
5-th percentile	2010	Coefficient of variation (CV)	0.00109
Q1	2013	Kurtosis	-0.70832
Median	2014	Mean	2014.180
Q3	2016	Median Absolute Deviation (MAD)	2
95-th percentile	2017	Skewness	-0.534855
Maximum	2017	Sum	670722
Range	7	Variance	4.82285
Interquartile Range (IQR)	3	Monotonicity	Not mono.

Interactions

In order to have an overall look of the PM values we create a scatter matrix that is a pairwise scatter plot of those variables presented in a matrix format. It can be used to determine whether the variables are correlated and whether the correlation is positive or negative. According to the figure bellow we can see that we have only positive correlation and also that there's no perfect correlation. The variance also increases as the x values increases. The data becomes more sparse as the values increase.

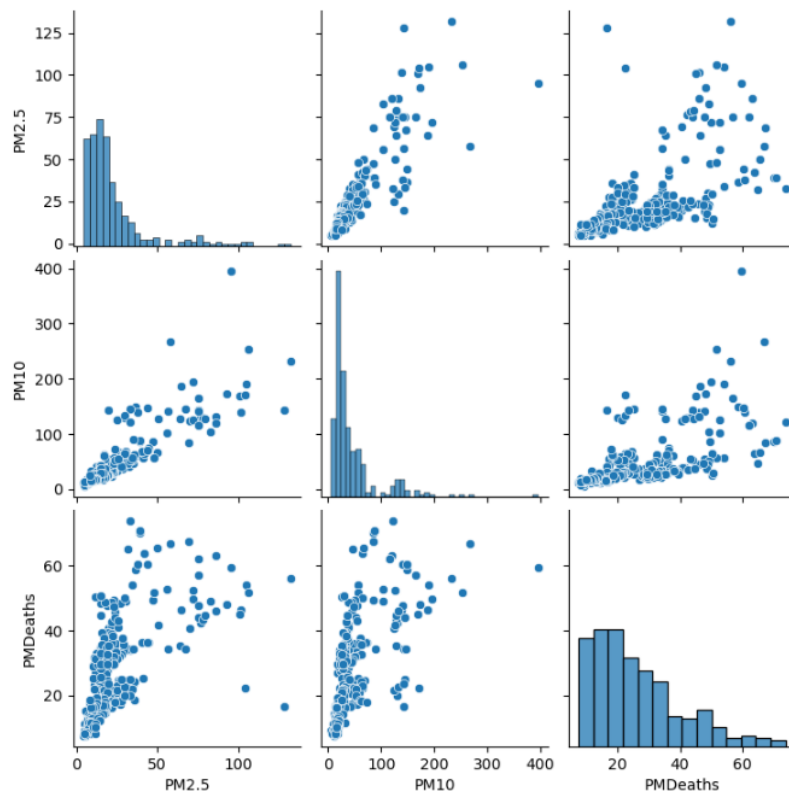


Figure 4 - Pairs scatter plots

Correlations

We created a correlation matrix that shows the coefficients between variables. It enables to see the degree to which a pair of variables are linearly related.

Spearman's ρ

The Spearman's rank correlation coefficient (ρ) measures the monotonic correlation between two variables, and is therefore better in catching nonlinear monotonic correlations than Pearson's r . This is a non-parametric test. In the correlation matrix bellow we can see that lies between -1 and +1, -1 indicating total negative monotonic correlation, 0 indicating no monotonic correlation and 1 indicating total positive monotonic correlation.

Pearson's r

The Pearson's correlation coefficient (r) measures the linear correlation between two variables. Like Spearman, the values lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation. This is a parametric test. Furthermore, r is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect r .

Kendall's τ

Similarly to Spearman's rank correlation coefficient, the Kendall rank correlation coefficient (τ) measures ordinal association between two variables. It's value lies between -1 and +1, -1 indicating total negative correlation, 0 indicating no correlation and 1 indicating total positive correlation.

Phik ϕ_k

Phik (ϕ_k) is a new and practical correlation coefficient that works consistently between categorical, ordinal and interval variables, captures non-linear dependency and reverts to the Pearson correlation coefficient in case of a bivariate normal input distribution. (Baak, M. et al, 2020).

By analyzing all the correlation coefficient, we can see that $PM_{2.5}$ and PM_{10} have a positive strong correlation which makes sense because they are all PM pollutants. Additionally, $PM_{2.5}$ can affect PM_{10} i.e. increase the diameter and became a PM_{10} . Besides that, they also have a positive correlation that is around 0,6 with PM deaths. Year has a slight negative correlation with the other variables.

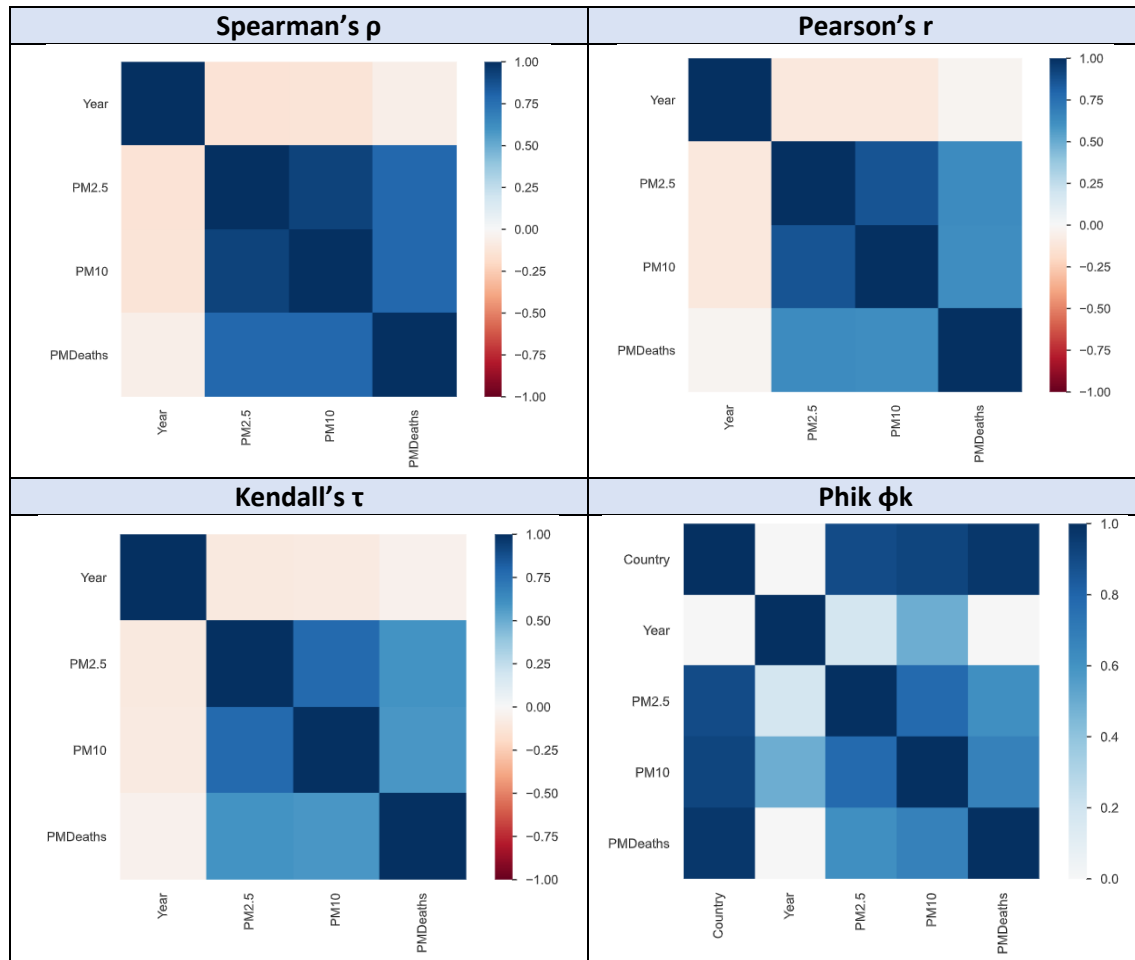


Figure 5 – Correlation Matrix

Verify data quality

In order to verify quality of our dataset we started to look for missing data. The missing data was overcome, since the PM concentration average of all cities was calculated. This resulted in no missing data in country wise PM concentrations. We proceeded our analysis and check for potential outliers. Although in this case the outliers that we found they contributed for a better result in the prediction. They represented extreme values but erroneous values.

A3. Data Preparation

Data preprocessing is a very important step in improving data quality since the obtained raw data is usually inconsistent, incomplete, and noisy. Before training our algorithm with the training data, it is important to make sure that it is in the proper form. Data preprocessing is the key step to identifying the missing key values, inconsistencies, and noise, containing errors and outliers.

Select data

Like mentioned before, we selected two datasets from Kaggle. One with deaths and another one with air pollution instances. Deaths by air pollution original dataset can be found at <https://www.kaggle.com/datasets/pavan9065/air-pollution> , the air pollution instances can be found at <https://www.kaggle.com/datasets/totoro29/air-pollution-level> .

On the deaths dataset only deaths by particulate matter, year and country was selected while on the air pollution levels only $PM_{2.5}$, PM_{10} , city, year and country was selected.

Clean data

Like mentioned previously, the dataset about air pollution levels had some “Nan” values in some cities, so we decided to drop them and stay only with the city that had values. As our goal was to have the information about the year and not the city we performed the average of all cities. After, we check both datasets and as expected, there were no missing values.

We found some outliers that, as mentioned before contributed to the prediction so they were not removed.

Given the majority of our attributes were highly right skewed as seen previously, they were standardized using z-score. Also because of the outliers, Z-Score was the best choice.

The following Figure 6 illustrates data before and after standardization.

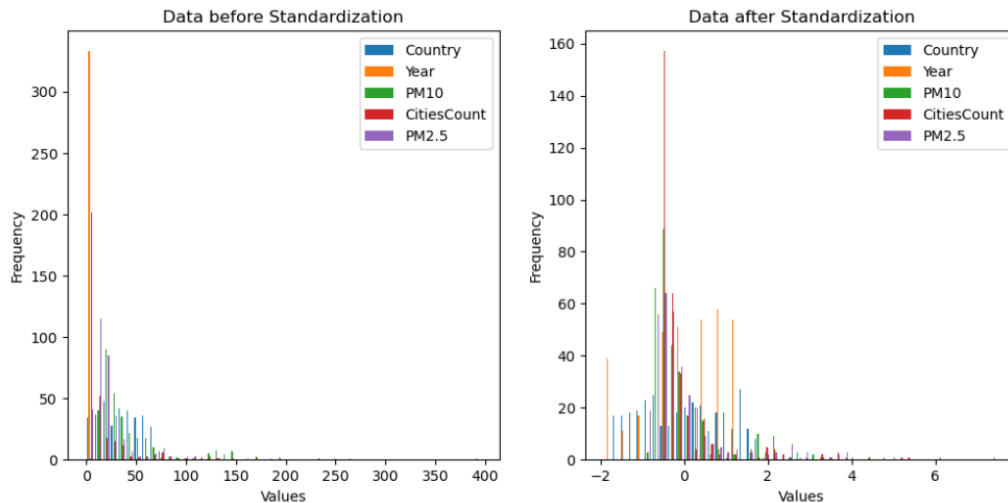


Figure 6 - Data before and after standardization

Construct data

On the dataset about air pollution levels we created a new instance: CitiesCount. It refers to number of not null values in the dataset. In other words, the number of the cities used in the average calculation.

Total deaths was not selected to the final dataset because the aim of the project was to predict deaths due to PM deaths using PM concentration values.

Integrate data

Since we had two datasets it was necessary to combine them and stay with one final dataset. The merge was performed using the year and the country – the necessary keys to identify the levels on PM_{2.5} and PM₁₀.

Format data

Originally, some of the columns had names that were not easy to work with so we decided to change them. On the Air pollution levels:

- “Country Name” becomes “Country”;
- “PM2.5 (µg/m3)” becomes “PM2.5”;
- “PM10 (µg/m3)” becomes “PM10”.

On the deaths dataset:

- “Entity” becomes “Country”

- “Deaths - Ambient particulate matter pollution - Sex: Both - Age: Age-standardized (Rate)” becomes “PMDeaths”.

With all this taken into account, we could then consider that the data was ready to be used in the Modelling step.

A4. Modeling

Our goal in this machine learning project is to predict deaths due to particulate matter. This is a regression problem meaning we will use Supervised Learning techniques.

Supervised Learning uses a training set to teach models to yield the desired output. The training dataset includes inputs and correct outputs, which allow the model to learn over time. Regression is used to understand the relationship between dependent and independent variables.

Select modeling technique

In order to arrive to the best prediction, we experimented 11 different algorithms and hyper-parameters. In the following step we will go deeper in each one of them.

Linear Regression

Modeling Technique:

Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.

Assumptions:

- First, linear regression needs the relationship between the independent and dependent variables to be linear. It requires to check for outliers since linear regression is sensitive to outlier effects;
- Secondly, the linear regression analysis requires all variables to be multivariate normal. When the data is not normally distributed a non-linear transformation might fix this issue;

- Thirdly, linear regression assumes that there is little or no multicollinearity in the data. Multicollinearity occurs when the independent variables are too highly correlated with each other;
- Fourthly, linear regression analysis requires that there is little or no autocorrelation in the data. Autocorrelation occurs when the residuals are not independent from each other;
- The last assumption of the linear regression analysis is homoscedasticity. The assumption of homoscedasticity (meaning “same variance”) is central to linear regression models. Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.

Ridge Regression

Modeling Technique:

Ridge regression is a type of linear regression technique that is used to reduce the overfitting of linear models. It is used when there are multiple variables that are highly correlated. It helps to prevent overfitting by penalizing the coefficients of the variables. It also reduces the overfitting by adding a penalty term to the error function that shrinks the size of the coefficients. It is like ordinary least squares regression, but the penalty term ensures that the coefficients do not become too large. This can be beneficial when there is a lot of noise in the data, as it prevents the model from being too sensitive to individual data points.

Assumptions:

- The assumptions of ridge regression are the same as that of linear regression: linearity, constant variance, and independence. However, as ridge regression does not provide confidence limits, the distribution of errors to be normal need not be assumed;
- Another requirement is to standardize the variables. This causes a challenge in notation since we must somehow indicate whether the variables in a particular formula are standardized or not. As far as standardization is concerned, all ridge regression calculations are based on standardized variables. When the final

regression coefficients are displayed, they are adjusted back into their original scale. However, the ridge trace is on a standardized scale;

- The downside of ridge regression is that it can be computationally intensive and can require more data to achieve accurate results.

Lasso Regression

Modeling Technique:

Lasso regression is a regularization technique. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models. This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

Lasso Regression uses L1 regularization technique. It is used when we have more features because it automatically performs feature selection.

Assumptions:

- The first is sparsity, i.e. only a small number of variables may actually be relevant;
- Another assumption is that the irrepressible condition must hold, this condition may look very technical but it only says that the relevant variable may not be very correlated with the irrelevant variables;
- Regularization is also an important concept that is used to avoid overfitting of the data, especially when the trained and test data are much varying.

KNN Regressor

Modeling Technique:

Ridge K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.

Assumptions:

- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other;
- In order to work with KNN, the number of K nearest neighbor needs to be specified in advance.

Random Forest Regressor

Modeling Technique:

Random forest is another flexible supervised machine learning algorithm. The "forest" references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions. It is an ensemble learning method that operates by constructing a multitude of decision trees at training time. For regression tasks, the mean or average prediction of the individual trees is returned.

Assumptions:

- There should be some actual values in the feature variables of the dataset, which will give the classifier a better chance to predict accurate results, rather than provide an estimation. Missing values should be handled from training the model;
- The predictions from each tree must have very low correlations.

Model Trees

Model Trees is a decision tree learner for regression task which is used to predict values of numerical response variable, which is a binary decision tree having linear regression functions at the terminal (leaf) nodes, which can predict continuous numerical attributes.

*With Linear Regression**Modeling Technique:*

Linear model trees combine linear models and decision trees to create a hybrid model that produces better predictions and leads to better insights than either model alone. A linear model tree is simply a decision tree with linear models at its nodes. This can be seen as a piecewise linear model with knots learned via a decision tree algorithm.

*With Lasso**Modeling Technique:*

Like Linear model trees, Lasso Model tree is a combination of Lasso models and decision trees to create a hybrid model that produces better predictions and leads to better insights than either model alone.

*With Ridge**Modeling Technique:*

Ridge Model tree is the application of Ridge Regression in decision trees. The idea is similar to the two mentioned above. A hybrid model that produces better predictions and leads to better insights.

Assumptions:

- Models Tree don't require particular assumptions about the input data - no data preparation, no constraints on the features types.

Multilayer Perceptron Regressor (MLP)

Modeling Technique:

The Multilayer Perceptron is a neural network where the mapping between inputs and output are non-linear. It has input and output layer, and one or more hidden layer with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function. Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer. In order for the algorithm to learn it uses Backpropagation that is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

Assumptions:

- The best part about a Multilayer Perceptron we don't assume any underlying pattern for the data. We are forcing the patterns to be captured using a almost exhaustive search. Activation functions are necessary because some of the functions are nonlinear example a circle which can be achieved through activation functions alone.

Gradient Boosting Regressor (GBoost)

Modeling Technique:

Gradient Boosting is a technique for repeatedly adding decision trees so that the next decision tree corrects the previous decision tree error. It is powerful enough to find any nonlinear relationship between your model target and features and has great usability that can deal with missing values, outliers, and high cardinality categorical values on your features without any special treatment.

Assumptions:

- Independence of observations;
- The selected loss function must be differentiable;
- Assumptions of the weak learner models, used to build the ensemble, need to be considered. For our purposes here, this means we need to consider the assumptions behind Decision Tree Regressors.

Extreme Gradient Boosting (XGBoost)

Modeling Technique:

XGBoost is an extension to gradient boosted decision trees (GBM) and is well known for its speed and performance. Predictions are made based on combining a set of simpler, weaker models - these models are decision trees that are created in sequential form. These models make predictions based on evaluating other decision trees through if-then-else true/false feature questions, using these to assess and estimate the probability of producing a correct decision.

Assumptions:

- The XGBoost may assume that encoded integer values for each input variable have an ordinal relationship;
- XGBoost assume that your data may not be complete (i.e. it can deal with missing values). When working with tree based algorithms, missing values are learned during the training phase. This then leads to the fact that: XGBoost can handle sparsity;

- XGBoost manages only numeric vectors, therefore if there's categorical variables they will need to be converted into numeric variables.

Generate test design

In order to represent better all the steps that were performed, we create the following process flow that illustrates the relationships between major tasks performed in the project.

The data set was split with 90% for training and validation and 10% reserved for final test. On this 90% , “K” subgroups was created in the K-Fold Cross Validation.

It is important to specify that in order to split our dataset between training, test and validation we used K-Fold Cross Validation because the dataset is randomly split up into ‘k’ groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group has been used as the test set. The error estimation is averaged over all k trials to get total effectiveness of our model. Every data point gets to be in a validation set exactly once and gets to be in a training set k-1 times. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method.

In order to evaluate the best number of folds, several trials were performed from K equal to 2 until 20. The best result was obtained using K equals to 7 in our top 3 models that will be explained in the following step.

The disadvantage of this method is that the training algorithm must be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.

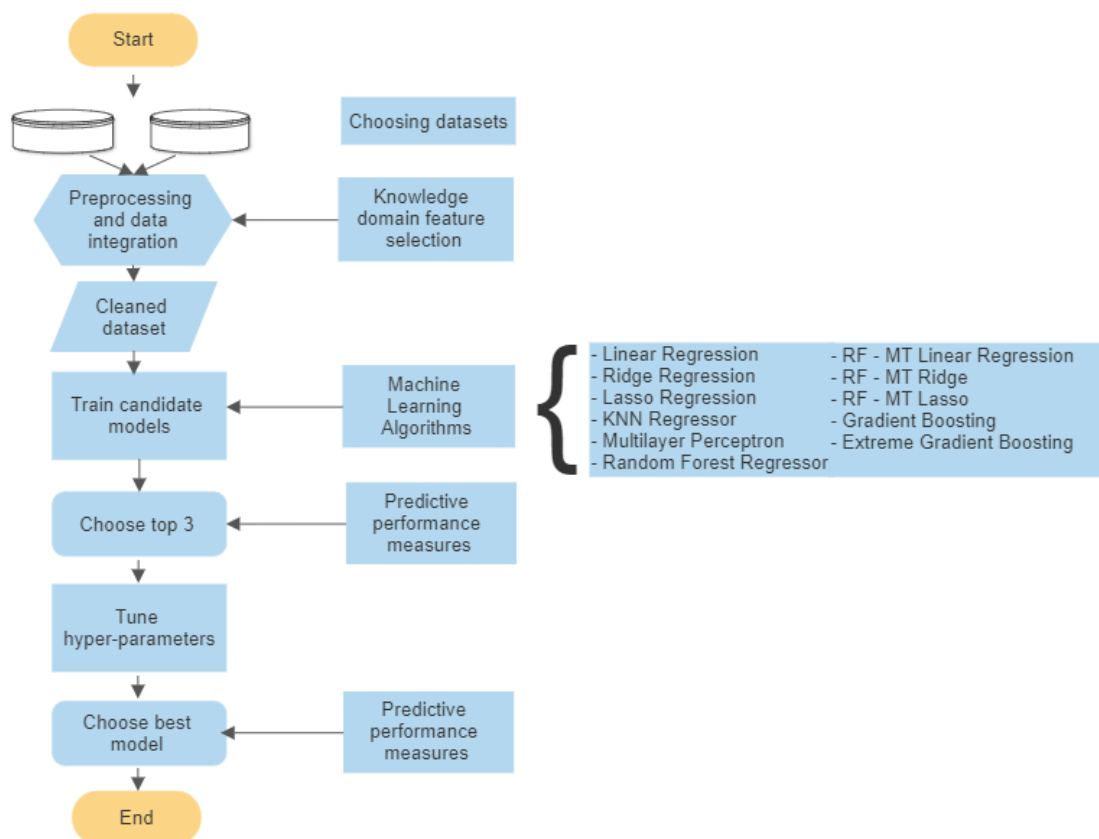


Figure 7 - Test design process flow

Build model

In order to get a diversification of the prediction, we decided to use several different algorithms and then choose the best using MAE, MAPE, MSE, RMSE and R^2 . Additional visualization were also used to support our decision.

Train candidate models

First iteration consists of multiple trials with different models using various k-folds (from 2 to 20). In this first trial, the standard hyper-parameters of each model were considered. This totaled a number of 24 948 trials.

On the figure below, we can see the results based on R^2 score of the different models.

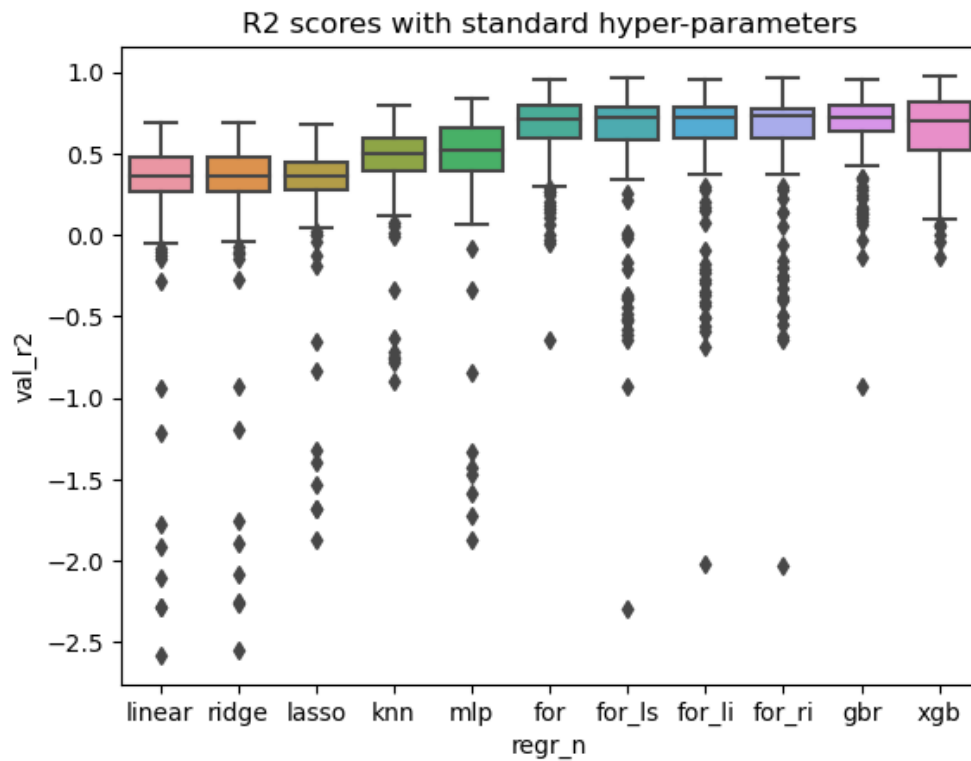


Figure 8 - Validation results for R^2 in different models with standard hyper-parameters

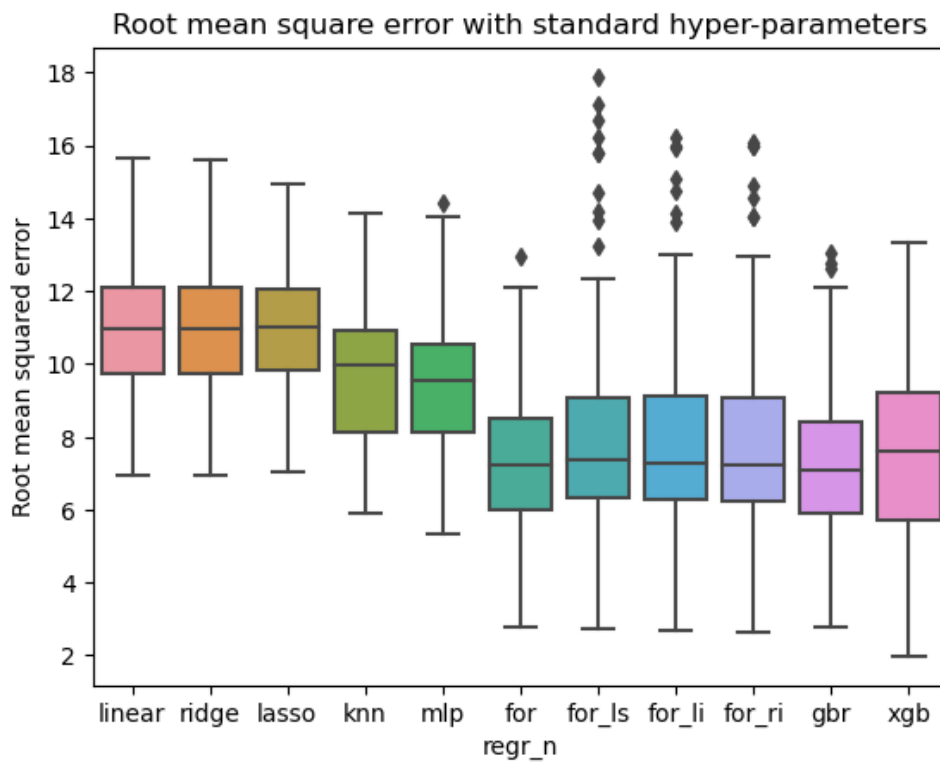


Figure 9 - Validation results for Root Mean Square Error in different models with standard hyper-parameters

On the graphs we can have a first approach of the first best models. Random Forest Regressor (for) obtained higher value of R^2 and lower errors. Also, Gradient Boost and Extreme Gradient Boost had a good results in terms of R^2 and errors estimation.

In order to compare all the Errors estimation and the R^2 , the following table was created. Here we can see that MAE ranges between 4,6 and 8,5. The lower MAE and MAPE was obtained by XGBoost. Although, by analyzing the rest of the evaluation criterion, we can see that Random Forest had better results of MSE, RMSE and R^2 . With puts RF in a position of the best model so far.

Table 6 - Summary table for the mean of predictive performance measures for each model

	MAE	MAPE	MSE	RMSE	R^2
Linear Regression	8.408373	0.400177	123.852352	10.972897	0.272985
Ridge Regression	8.405513	0.400272	123.615101	10.963979	0.274790
Lasso Regression	8.560176	0.413195	123.488375	10.987572	0.290719
KNN Regressor	7.285750	0.317703	96.248377	9.655994	0.453058
MLP	6.818322	0.288481	92.056601	9.410421	0.446731
Random Forest Regressor	4.675507	0.193537	55.880526	7.252989	0.673062
RF - MT Linear Regression	4.864658	0.197382	66.223319	7.782174	0.599209
RF - MT Ridge	4.859223	0.197202	65.936510	7.765317	0.601074
RF - MT Lasso	4.921704	0.201479	69.249181	7.913175	0.582127
GBoost	4.872267	0.201391	55.843214	7.230265	0.671832
XGBoost	4.636762	0.180261	62.493635	7.526831	0.641797

Although Random Forest had in overall the best results. It was to soon to choose as the best model. So, we decided to pick the best 3 models that are Random Forest Regressor, Gradient Boosting Regressor and Extreme Gradient Boosting Regressor.

Of these three models, the best results were obtained using 7 folds and they are described in the following table:

Table 7 - Summary table for best folds for the best scoring candidate models

	k-folds	MAE	MAPE	MSE	RMSE	R^2
Random Forest Regressor	7	4.721765	0.197854	55.487580	7.379438	0.709416
GBoost	7	4.889424	0.203691	55.444981	7.364126	0.706552
XGBoost	7	4.574761	0.184108	55.976407	7.383448	0.705462

Hyper-parameters' tuning

For each of the selected candidate a grid search was performed in order to find the most suitable hyper-parameters. This means that for each technique, all combinations of the given values are trained and evaluated, using k-fold cross validation. The number of folds was determined in the previous step.

Random Forest

For Random Forest, the following parameter variations were experimented:

Table 8 - Random Forest parameter variation for grid search

	Candidate 1	Candidate 2	Candidate 3	Candidate 4	Candidate 5
Number of trees	400	1000	2000	4000	-
Max depth	5	10	50	100	None

This resulted in a total of 140 trials, fitting 7 folds for each 20 candidate combination.

The set of candidates that obtained a best score (considering R^2) were the following:

Table 9 - Random forest best hyper-parameters

Number of trees	Max depth
1000	50

Gradient Boost

For Gradient Boost, the following parameter variations were experimented:

Table 10 - GBoost parameter variation for grid search

	Candidate 1	Candidate 2	Candidate 3	Candidate 4	Candidate 5
Number of boosting rounds	400	800	1000	2000	4000
Learning Rate	0.0005 to 1 with 50 step				
Sub sample	0.1	0.25	0.5	0.75	1.0
Max depth	5	10	50	100	None

This resulted in 43750 trials, fitting 7 folds for each of 6250 candidate combinations.

The set of candidates that obtained a best score (considering R^2) were the following:

Table 11 - GBoost best hyper-parameters

Number of boosting rounds	Learning rate
4000	0.04129591836734694
Sub sample	Max depth
0.5	5

XG Boost

For Extreme Gradient Boost, the following parameter variations were experimented:

Table 12 - XG Boost parameter variation for grid search

	Candidate 1	Candidate 2	Candidate 3	Candidate 4	Candidate 5
Number of boosting rounds	400	800	1000	2000	4000
Learning Rate	0.0005 to 1 with 50 step				
Sub sample	0.1	0.25	0.5	0.75	1.0
Max depth	5	10	50	100	None

This resulted in 43750 trials, fitting 7 folds for each of 6250 candidate combinations.

The set of candidates that obtained a best score (considering R^2) were the following:

Table 13 - XGBoost best hyper-parameters

Number of boosting rounds	Learning rate
4000	0.02089795918367347
Sub sample	Max depth
0.75	5

The results for the best hyper-parameters obtained with the validation sets were the following:

Table 14 - Validation scores for the best hyper-parameters

	Random Forest	GBoost	XG Boost
Best score (R^2)	0.709499	0.753666	0.745593

Test set results

After hyper- parameter tuning we can finally use the test set (split at the beginning) to calculate our improved models results. This helps us to see how the model will fit when applying an unseen data.

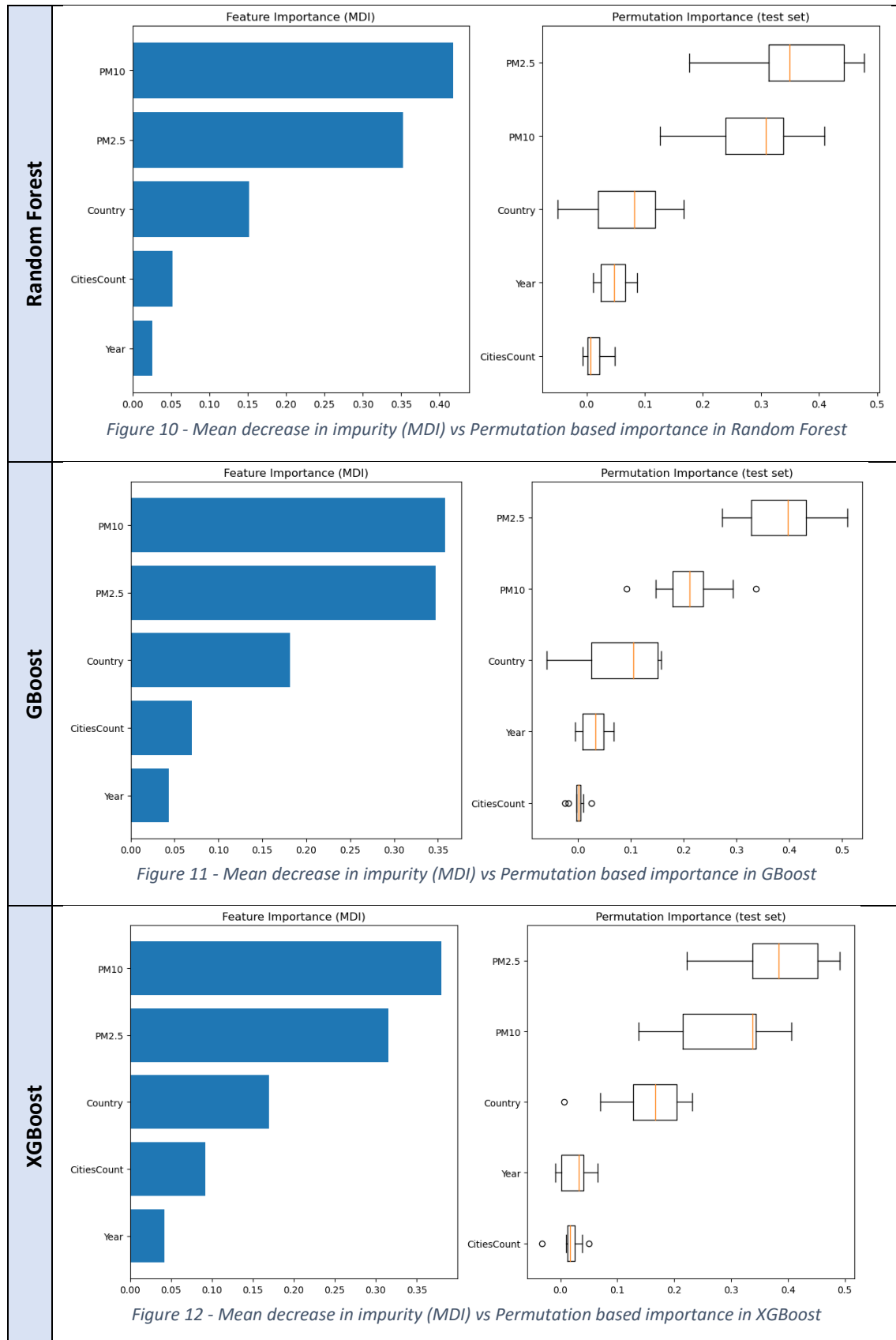
Table 15 - Results with test set

	MAE	MAPE	MSE	RMSE	R ²
Random Forest Regressor	7.561012	0.298011	99.361033	9.968000	0.657429
GBoost	7.007454	0.280956	87.554636	9.357063	0.698134
XGBoost	7.032485	0.261984	90.682497	9.522736	0.687350

Additional comparisons are presented in the next sub-chapters.

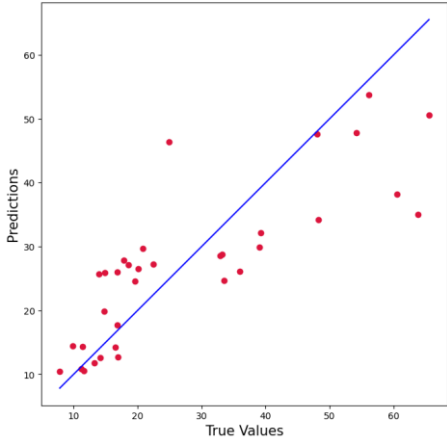
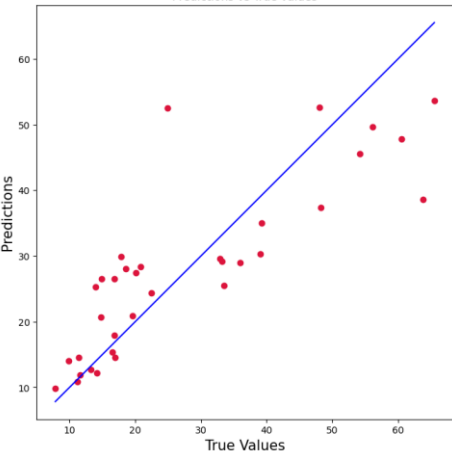
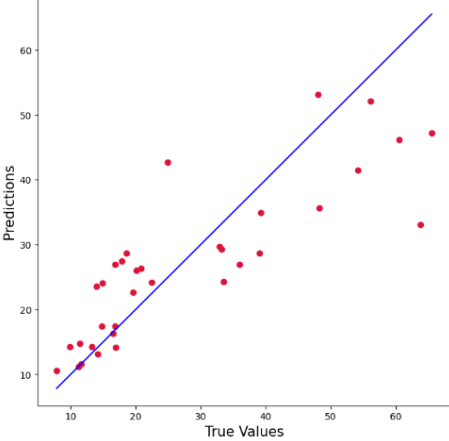
Feature Importance

Table 16 - Feature importance comparison RF, GB and XGB

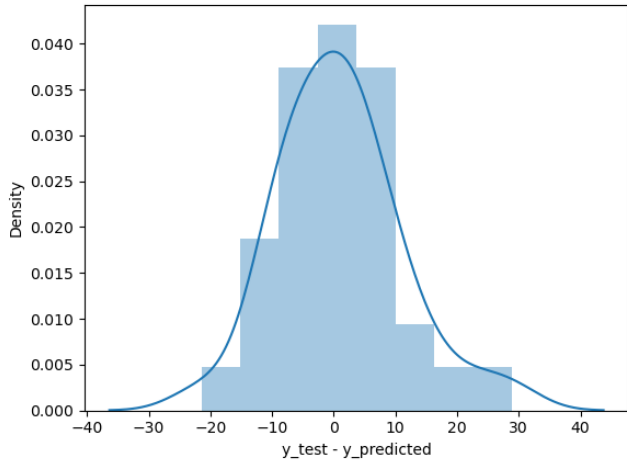
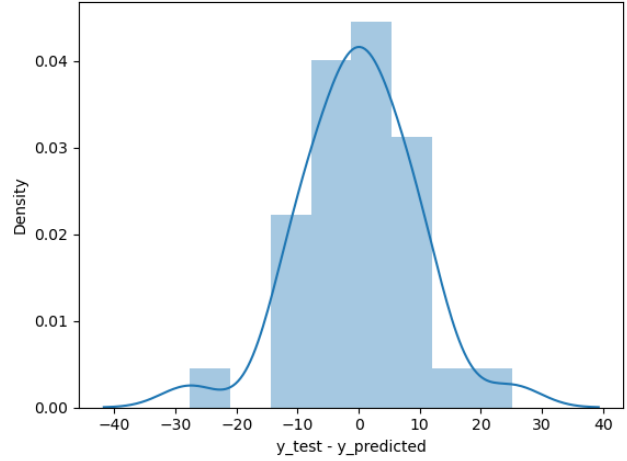
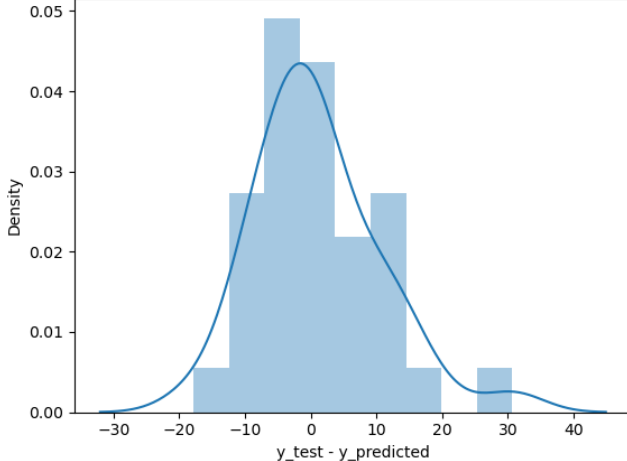


Predictions vs True Values

Table 17 - Predictions vs True Values comparison RF, GB and XGB

Random Forest	<div></div> <p>Figure 13 - RF Predictions vs True Values</p>
GBoost	<div></div> <p>Figure 14 - GB Predictions vs True Values</p>
XGBoost	<div></div> <p>Figure 15 - XGB RF Predictions vs True Values</p>

Distribution of residuals

Random Forest	 <p data-bbox="638 728 1005 761"><i>Figure 16 - RF distribution of residuals</i></p>
GBoost	 <p data-bbox="638 1288 1005 1321"><i>Figure 17 - GB distribution of residuals</i></p>
XGBoost	 <p data-bbox="638 1848 1005 1881"><i>Figure 18 - XGB distribution of residuals</i></p>

Assess model

Before going deeply in the model assess, it is important to mention that the figures below represent the feature importance on the predictions.

Tree-based models provide an alternative measure of feature importance's based on the mean decrease in impurity (MDI). Impurity is quantified by the splitting criterion of the decision trees (Gini, Log Loss or Mean Squared Error). However, this method can give high importance to features that may not be predictive on unseen data when the model is overfitting. Permutation-based feature importance, on the other hand, avoids this issue, since it can be computed on unseen data.

Furthermore, impurity-based feature importance for trees are strongly biased and favor high cardinality features (typically numerical features) over low cardinality features such as binary features or categorical variables with a small number of possible categories. Permutation-based feature importance do not exhibit such a bias (scikit-learn, 2001).

In our model higher permutation importance was obtained by PM2.5 followed by PM10 and Country. Lower importance was obtained by Year and CitiesCount. We therefor can conclude that our model predict PMdeaths mainly by the first three. Year and CitiesCount could be excluded and trial again the models without them. This step was not considered in this project due to time constraints.

Now going back to ours models, the results obtained from the test set can be ranked according to the evaluation criterion.

Gradient Boost (GB) model obtained an overall better performance, followed by Extreme Gradient Boost (XGB) and Random Forest Regressor (RF) in last place.

GB and XGB have very similar results on the final test set, however GB obtained lower values for errors and higher R^2 score. Also, the best scores in the training set after the hyper-parameter tuning was also obtained by GB, followed by XGB and RF for last.

This is also supported by the distribution of residuals. We can see GB residuals are centered in zero whereas XGB tends slightly to the left. RF residual distribution although centered in the zero as well, present a peak that is not as slim and the rest of the candidates, meaning that there is a higher frequency of residuals that are not zero when compared to GB and XGB.

Another important thing to be mentioned is that if we use the graphs (table 17) to compare the results, we can see that there is not much difference between the observed and

the predicted values on the three models. This visualization is not very helpful to make a decision given the fact that our dataset is small, that why it is important to analyzed additional predictive measures.

A5. Evaluation

Evaluate results

The results that we obtained satisfied the business objective. We were able to predict based in our data the deaths due to Particular Matters. Putting this result in the real-world scenario we can see that we are able to predict and see how the future might look like and what actions we need to take.

Based on the Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R^2 we ranked 3 main Models as mentioned before. Then we decided to choose GBoost baised on the reasons that we already discuss.

Table 18 – Gboost Results

	MAE	MAPE	MSE	RMSE	R^2
GBoost	7.007454	0.280956	87.554636	9.357063	0.698134

Review process

First, this project was very interesting to do. It is very satisfying to see how we come from a simple dataset to a whole set of trials and models. A lot of knowledge was acquired during this period of time. The necessary codes were very complex to be developed but at the end it was worth the dedication.

By reviewing all the performed steps, we think that a better time organization and distribution among the phases would be beneficial. The creation of the algorithms in Python were very complex and time consuming. The hyper-parameter tuning also was a challenging step as we had to perform a lot of trials. We also tried to step out and include visualizations to represent the data and monitor the performance of the models.

Determine next steps

After doing this work we arrived to the conclusion that more information about particular materials can also improve our prediction. In this case we could include PM_1 (particles with an aerodynamic diameter smaller than $1\text{ }\mu\text{m}$) and $PM_{0.1}$ (particles with an aerodynamic diameter smaller than $0.1\text{ }\mu\text{m}$). The two of them are considered ultrafine particles.

Ultra-fine dust is the most damaging variant of fine particles because the particles penetrate directly through the lungs into the bloodstream and are thus spread to the organs.

Also, we believe that there are more variable that can be added in order to provide a better understanding about this topic.

Besides that, we also believe that there might be other predictive algorithm that we could use in this prediction task.

After performing the feature importance, we saw that CitiesCount and Year were not so important for the prediction so removing them and trial the models again is also considered to be a future task.

The hyper-parameters can also be something to improve in order to have a better prediction. Meaning that test further different combinations could help us find an even better model. However, it is importance to take into consideration that the hyper-parameter tuning is very time consuming.

Given that our dataset had a small number of instances an alternative approach could be to experiment using Oversampling technique in the data partition and compare if better results are obtained.

References

- Aggarwal, C. (n.d.). *Data Mining: The Textbook*. 2015: Springer.
- Baak, M., Koopman, R., Snoek, H., & Klous, S. (2020). A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristicsA new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics. *Computational Statistics & Data Analysis*. doi:<https://doi.org/10.1016/j.csda.2020.107043>
- Comunian, S., Dongo, D., Milani, C., & Palestini, P. (2020). Air Pollution and COVID-19: The Role of Particulate Matter in the Spread and Increase of COVID-19's Morbidity and Mortality. *International Journal of Environmental Research and Public Health*, 17, no. 12: 4487. Retrieved from <https://doi.org/10.3390/ijerph17124487>
- EPA. (2022, 12 2). *Particulate Matter (PM10) Trends*. Retrieved from United States Particulate Matter (PM10) Trends: <https://www.epa.gov/air-trends/particulate-matter-pm10-trends>
- Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.
- Jonathan M. Samet, F. D. (2000). Fine Particulate Air Pollution and Mortality in 20 U.S. Cities, 1987–1994. *The New England Journal of Medicine*, 1742-1749. Retrieved from <https://doi.org/10.1056/NEJM200012143432401>
- Moreira, J., Carvalho, A., & Horvath, T. (2018). *A General Introduction to Data Analytics*. Wiley-Interscience.
- Raaschou-Nielsen, O. H. (2010). Long-term exposure to indoor air pollution and wheezing symptoms in infants. *Indoor Air*, 159-167. Retrieved from <https://doi.org/10.1111/j.1600-0668.2009.00635.x>
- scikit-learn. (2001). *Permutation feature importance*. Retrieved from scikit-learn documentation: https://scikit-learn.org/stable/modules/permutation_importance.html
- Vallero, D. (2007). *Fundamentals of Air Pollution*. American Press. doi:<https://doi.org/10.1016/B978-0-12-373615-4.X5000-6>
- WHO. (2014). *Burden of disease from Household Air Pollution 2012 - Summary of results*. Geneva: World Health Organization.
- WHO. (2022, November 28). *Air Pollution - Impact*. Retrieved from World Health Organization: https://www.who.int/health-topics/air-pollution#tab=tab_2

Appendix

IACEC_script

December 5, 2022

```
[73]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
!jupyter nbextension enable --py widgetsnbextension
from pandas_profiling import ProfileReport
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression, LinearRegression, Ridge, \
    Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, \
    GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.model_selection import train_test_split, cross_val_score, \
    cross_validate, GridSearchCV, KFold
from sklearn.utils import resample
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, \
    mean_absolute_percentage_error
from lineartree import LinearForestRegressor
from sklearn.inspection import permutation_importance
```

Enabling notebook extension jupyter-js-widgets/extension...
- Validating: ok

```
[2]: ap = pd.read_csv('../01. Dataset/Air Pollution.csv')
ap.head()
```

```
[2]: Country Name    City    Year    PM2.5 (g/m3)    PM10 (g/m3)    NO2 (g/m3) \
0    Afghanistan    Kabul    2019         119.77         NaN         NaN
1      Albania    Durres    2015          NaN         17.65        26.63
2      Albania    Durres    2016         14.32         24.56        24.78
3      Albania    Elbasan    2015          NaN         NaN         23.96
4      Albania    Elbasan    2016          NaN         NaN         26.26
```

```
PM25 temporal coverage (%)    PM10 temporal coverage (%) \
```

```

0          18.0          NaN
1          NaN          NaN
2          NaN          NaN
3          NaN          NaN
4          NaN          NaN

```

```

      N02 temporal coverage (%) Updated Year
0          NaN          2022
1      83.961187          2022
2      87.932605          2022
3      97.853881          2022
4      96.049636          2022

```

```
[3]: ap.isnull().sum()
```

```

[3]: Country Name          0
City                    0
Year                    0
PM2.5 (g/m3)          17143
PM10 (g/m3)           11082
NO2 (g/m3)             9991
PM25 temporal coverage (%) 24916
PM10 temporal coverage (%) 26810
NO2 temporal coverage (%)  12301
Updated Year           0
dtype: int64

```

```
[4]: ap.describe()
```

```

[4]:
      count      Year  PM2.5 (g/m3)  PM10 (g/m3)  NO2 (g/m3)  \
count  32191.000000  15048.000000  21109.000000  22200.000000
mean    2015.579354    22.920320    30.533252    20.619336
std         2.752654    17.925906    29.312756    12.133388
min     2000.000000     0.010000     1.040000     0.000000
25%     2014.000000    10.350000    16.980000    12.000000
50%     2016.000000    16.000000    22.000000    18.800000
75%     2018.000000    31.000000    31.300000    27.160000
max     2021.000000   191.900000   540.000000   210.680000

      PM25 temporal coverage (%)  PM10 temporal coverage (%)  \
count          7275.000000          5381.000000
mean           90.794096           90.583500
std            14.872681           13.816311
min              0.000000            2.568493
25%            88.595890           87.945205
50%            97.000000           96.039000
75%            99.000000           98.938000

```


max	100.000000	100.000000
	N02 temporal coverage (%)	Updated Year
count	19890.000000	32191.000000
mean	93.696804	2021.744214
std	10.451751	1.051897
min	1.923077	2016.000000
25%	93.207763	2022.000000
50%	96.369863	2022.000000
75%	98.926941	2022.000000
max	100.000000	2022.000000

```
[5]: ap_s = ap.loc[:, ['Country Name', 'City', 'Year', 'PM2.5 (g/m3)', 'PM10 (g/
    <math>m^3)</math>']]
    ap_s = ap_s.rename(columns = {
        'Country Name': 'Country',
        'Year': 'Year',
        'PM2.5 (g/m3)': 'PM2.5',
        'PM10 (g/m3)': 'PM10',
    })
    ap_s.isnull().sum()
```

```
[5]: Country      0
    City          0
    Year          0
    PM2.5      17143
    PM10       11082
    dtype: int64
```

```
[6]: ap_s = ap_s.dropna(how='any', subset=['PM2.5', 'PM10'], axis=0)
    ap_s.isnull().sum()
```

```
[6]: Country      0
    City          0
    Year          0
    PM2.5         0
    PM10          0
    dtype: int64
```

```
[7]: dp = pd.read_csv('../01. Dataset/death-rates-from-air-pollution.csv')
    dp.head()
```

```
[7]:      Entity Code  Year  \
0  Afghanistan  AFG  1990
1  Afghanistan  AFG  1991
2  Afghanistan  AFG  1992
3  Afghanistan  AFG  1993
```

4 Afghanistan AFG 1994

	Deaths - Air pollution - Sex: Both - Age: Age-standardized (Rate) \
0	299.477309
1	291.277967
2	278.963056
3	278.790815
4	287.162923

	Deaths - Household air pollution from solid fuels - Sex: Both - Age: Age-standardized (Rate) \
0	250.362910
1	242.575125
2	232.043878
3	231.648134
4	238.837177

	Deaths - Ambient particulate matter pollution - Sex: Both - Age: Age-standardized (Rate) \
0	46.446589
1	46.033841
2	44.243766
3	44.440148
4	45.594328

	Deaths - Ambient ozone pollution - Sex: Both - Age: Age-standardized (Rate)
0	5.616442
1	5.603960
2	5.611822
3	5.655266
4	5.718922

```
[8]: dp_s = dp.loc[:, ['Entity', 'Year',
                      'Deaths - Air pollution - Sex: Both - Age: Age-standardized (Rate)',
                      'Deaths - Ambient particulate matter pollution - Sex: Both - Age: Age-standardized (Rate)']]
dp_s = dp_s.rename(columns = {
    'Entity': 'Country',
    'Deaths - Air pollution - Sex: Both - Age: Age-standardized (Rate)': 'TotalDeaths',
    'Deaths - Ambient particulate matter pollution - Sex: Both - Age: Age-standardized (Rate)': 'PMDeaths',
})
dp_s.isnull().sum()
```

```
[8]: Country      0
      Year        0
      TotalDeaths 0
      PMDeaths    0
      dtype: int64
```

```
[9]: dp_sg = dp_s.groupby(['Country', 'Year'])
      dp_sg.agg({'TotalDeaths': 'mean', 'PMDeaths': 'mean'}).head()
```

```
[9]:
```

		TotalDeaths	PMDeaths
Country	Year		
Afghanistan	1990	299.477309	46.446589
	1991	291.277967	46.033841
	1992	278.963056	44.243766
	1993	278.790815	44.440148
	1994	287.162923	45.594328

```
[10]: ap_sf = ap_s.merge(dp_s)
      ap_sf.isnull().sum()
```

```
[10]: Country      0
      City        0
      Year        0
      PM2.5       0
      PM10        0
      TotalDeaths 0
      PMDeaths    0
      dtype: int64
```

```
[11]: ap_sg = ap_sf.groupby(['Country', 'Year'])
      pd.set_option('display.max_columns', None)
      pd.set_option('display.max_rows', None)
      #ap_sc.loc[(ap_s['Country'] == 'Australia') & (ap_s['Year'] == 2013)] # For validation
      final_df = ap_sg.agg({'PM2.5': 'mean', 'PM10': 'mean', 'TotalDeaths': 'mean', 'PMDeaths': 'mean', 'City': 'size'})
      final_df.rename(columns={'City': 'CitiesCount'}, inplace=True)
```

```
[12]: # final_df.to_csv('../01. Dataset/cleaned_dataset.csv')
```

```
[13]: profile = ProfileReport(final_df.iloc[:, [0,1,3]], title="Air Pollution Report")
      profile
```

```
Summarize dataset: 0%|          | 0/5 [00:00<?, ?it/s]
```

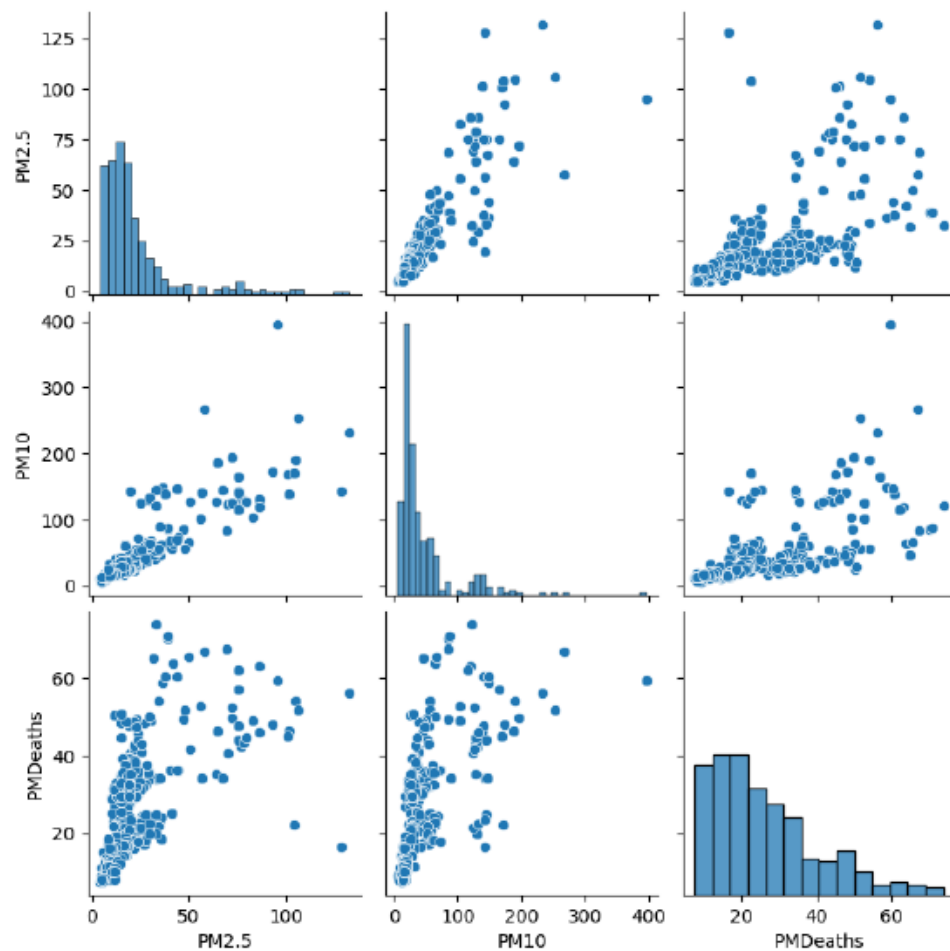
```
Generate report structure: 0%|          | 0/1 [00:00<?, ?it/s]
```

```
Render HTML: 0%|          | 0/1 [00:00<?, ?it/s]
```

<IPython.core.display.HTML object>

[13]:

```
[14]: sns.pairplot(final_df.iloc[:, [0,1,3]])
      plt.show()
```



```
[15]: scoring = [
    "neg_mean_absolute_error",
    "neg_mean_absolute_percentage_error",
    "neg_mean_squared_error",
    "neg_root_mean_squared_error",
    "r2"
```

```

]
train_frac = 0.90
target_columns = [0, 1, 3, 6, 2, 5]
categorical_columns = [0, 1]
r_states = [ 1 ]

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)

[162]: data = final_df.reset_index()
data = data.iloc[:, target_columns]
# data # Final input dataset

[17]: def clean_dataset(dataset, plot=False, remove_ol=False, rs=None, print_f=False):
    X = dataset[dataset.columns[:-1]].values # X origin is first columns
    y = dataset[dataset.columns[-1]].values # y origin is last single column
    X_i, y_i = X, y
    ol = []

    # Convert 'Country' column to label so that it can be used on Regression
    for i in categorical_columns:
        X[:,i] = LabelEncoder().fit_transform(X[:,i])

    if plot:
        fig, ax = plt.subplots(2, 2, figsize=[12, 12])
        ax[0, 0].set_title("Data before Standardization")
        ax[0, 0].hist(X, bins=50, orientation="vertical", label=dataset.
columns[:-1].values.tolist())
        ax[1, 0].set_title("Data with OutLiers")
        ax[1, 0].scatter(X[:, -1], y[:])
        ax[0, 0].set_xlabel('Values')
        ax[0, 0].set_ylabel('Frequency')
        ax[0, 0].legend()

    # Remove Out-Liers
    if remove_ol:
        olf = LocalOutlierFactor()
        #olf = IsolationForest(random_state=rs)
        ol = olf.fit_predict(X)
        mask = (ol != -1)
        if print_f:
            print(mask.sum(), X.shape, mask.sum() / X.shape[0])
        X, y = X[mask, :], y[mask]

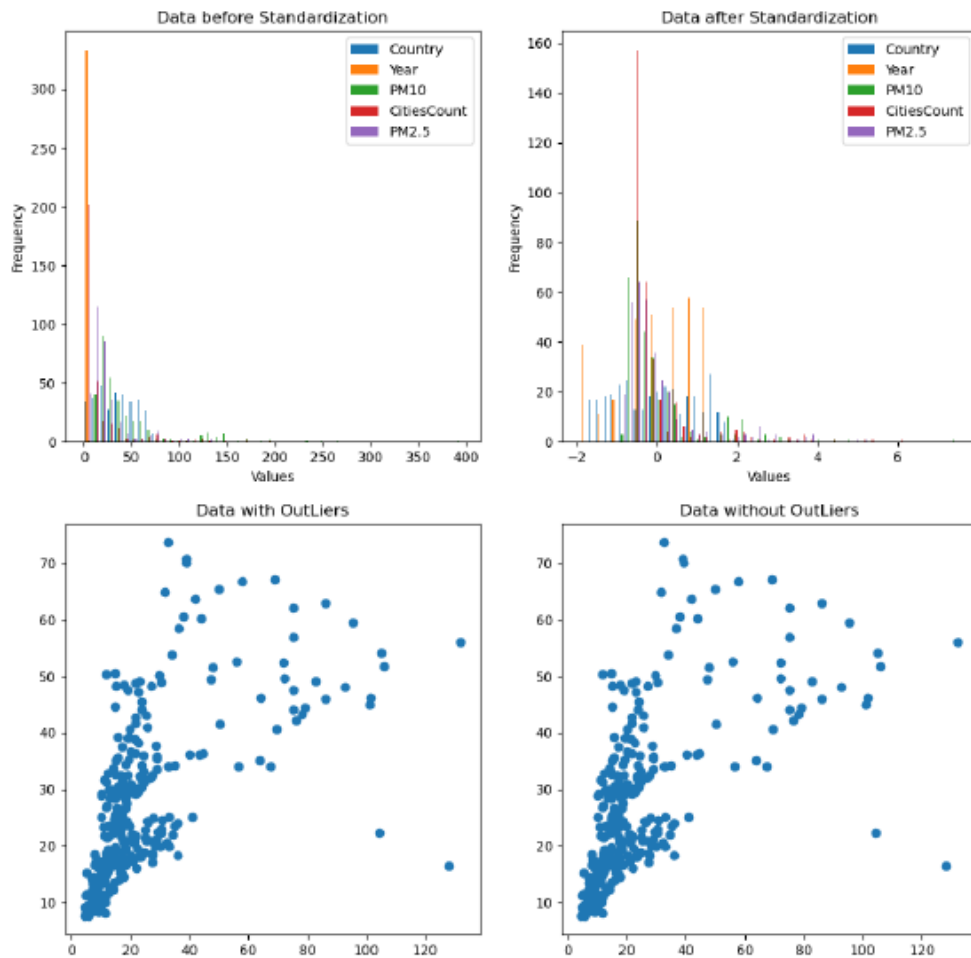
    X = StandardScaler().fit_transform(X, y)

```

```
if plot:
    ax[0, 1].set_title("Data after Standardization")
    ax[0, 1].hist(X, bins=50, orientation="vertical", label=dataset.
columns[: -1].values.tolist())
    mask = (ol == -1)
    ax[1, 1].set_title("Data without OutLiers")
    ax[1, 1].scatter(X_i[:, -1], y_i)
    ax[1, 1].scatter(X_i[mask, -1], y_i[mask], color='r')
    ax[0, 1].set_xlabel('Values')
    ax[0, 1].set_ylabel('Frequency')
    ax[0, 1].legend()

data = pd.DataFrame(np.hstack((X, np.reshape(y, (-1, 1)))))
return data, X, y
```

```
[18]: graph = clean_dataset(data, plot=True)
plt.show()
```



```
[19]: def get_dataset(dataframe, rs=None):
    #ovdf = dataframe.sample(frac=1, random_state=rs)
    #train, test = np.split(ovdf, [int(train_frac*len(ovdf))])

    train, test = train_test_split(dataframe, train_size=train_frac,
    random_state=rs)

    train, X_train, y_train = clean_dataset(train, remove_ol=False, rs=rs)
    test, X_test, y_test = clean_dataset(test, remove_ol=False, rs=rs)

    return X_train, y_train, X_test, y_test
```

```
[51]: global_results = []
for kfolds in range(2, 20):
    for rs in r_states:
        models = [
            # List of models to cross validate
            # Note all of hyper-parameters are fixed, just random_state is
            # changed between trials
            ("linear", LinearRegression()),
            ("ridge", Ridge(random_state=rs)),
            ("lasso", Lasso(random_state=rs)),
            ("knn", KNeighborsRegressor()),
            ("mlp", MLPRegressor(max_iter=3000, random_state=rs)),
            ("for", RandomForestRegressor(random_state=rs)),
            ("for_ls", LinearForestRegressor(Lasso(random_state=rs),
            random_state=rs, max_features=None)),
            ("for_li", LinearForestRegressor(LinearRegression(),
            random_state=rs, max_features=None)),
            ("for_ri", LinearForestRegressor(Ridge(random_state=rs),
            random_state=rs, max_features=None)),
            ("gbr", GradientBoostingRegressor(random_state=rs)),
            ("xgb", XGBRegressor(random_state=rs))

        ]

        for regr_n, regr in models:
            # Get the dataset
            X_train, y_train, _, _ = get_dataset(data, rs=rs)
            # Cross validate the model
            kf = KFold(kfolds, shuffle=True, random_state=rs)
            cv = cross_validate(regr, X_train, y_train, scoring=scoring, cv=kf,
            return_train_score=True)
            # Save score of model for future reference
            for k, v in cv.items():
                for v_i in v:
                    global_results.append({
                        "regr_n": regr_n,
                        "kfolds": kfolds,
                        k: v_i,
                        "random_state": rs,
                        "model": regr,
                    })

# Final results
len(global_results)
```

C:\Users\Asus\anaconda3\lib\site-
packages\sklearn\normal_network_multilayer_perceptron.py:692:


```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.  
warnings.warn(  
C:\Users\Asus\anaconda3\lib\site-  
packages\sklearn\neural_network\_multilayer_perceptron.py:692:  
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3000) reached and  
the optimization hasn't converged yet.
```

[51]: 24948

```
[164]: gr_df = pd.DataFrame.from_dict(global_results)
gr_df_abs= gr_df.groupby(['regr_n', 'kfold']).agg({
    'train_neg_mean_absolute_error': lambda x : abs(x.mean()),
    'test_neg_mean_absolute_error': lambda x : abs(x.mean()),
    'train_neg_mean_absolute_percentage_error': lambda x : abs(x.mean()),
    'test_neg_mean_absolute_percentage_error': lambda x : abs(x.mean()),
    'train_neg_mean_squared_error': lambda x : abs(x.mean()),
    'test_neg_mean_squared_error': lambda x : abs(x.mean()),
    'train_neg_root_mean_squared_error': lambda x : abs(x.mean()),
    'test_neg_root_mean_squared_error': lambda x : abs(x.mean()),
    'train_r2': ['mean'],
    'test_r2': ['mean'],
}).sort_values(by=('test_r2', 'mean'),ascending=False)[0:20]

# gr_df_abs
```

```
[90]: summary_table = gr_df.copy()
summary_table = summary_table.groupby('regr_n').agg({
    'test_neg_mean_absolute_error': lambda x : abs(x.mean()),
    'test_neg_mean_absolute_percentage_error': lambda x : abs(x.mean()),
    'test_neg_mean_squared_error': lambda x : abs(x.mean()),
    'test_neg_root_mean_squared_error': lambda x : abs(x.mean()),
    'test_r2': ['mean']
})

summary_table
```

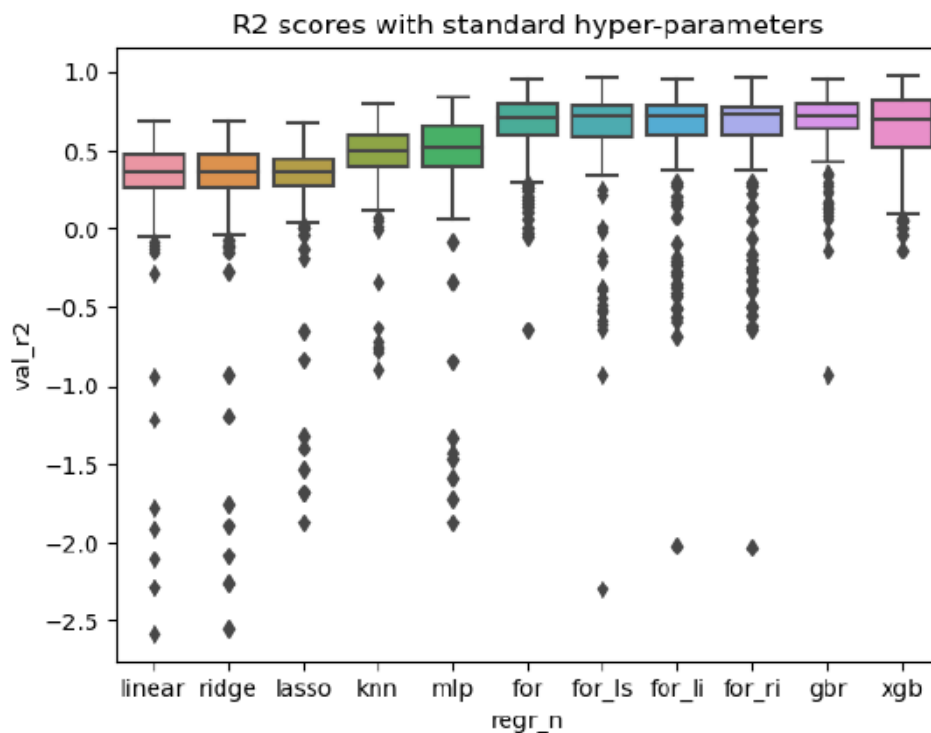
```
[90]:      test_neg_mean_absolute_error test_neg_mean_absolute_percentage_error \
      <lambda>                                <lambda>

regr_n
for      4.675507                        0.193537
for_li    4.864658                        0.197382
for_ls    4.921704                        0.201479
for_ri    4.859223                        0.197202
gbr       4.872267                        0.201391
knn       7.285750                        0.317703
lasso     8.560176                        0.413195
linear    8.408373                        0.400177
mlp       6.818322                        0.288481
ridge     8.405513                        0.400272
xgb       4.636762                        0.180261

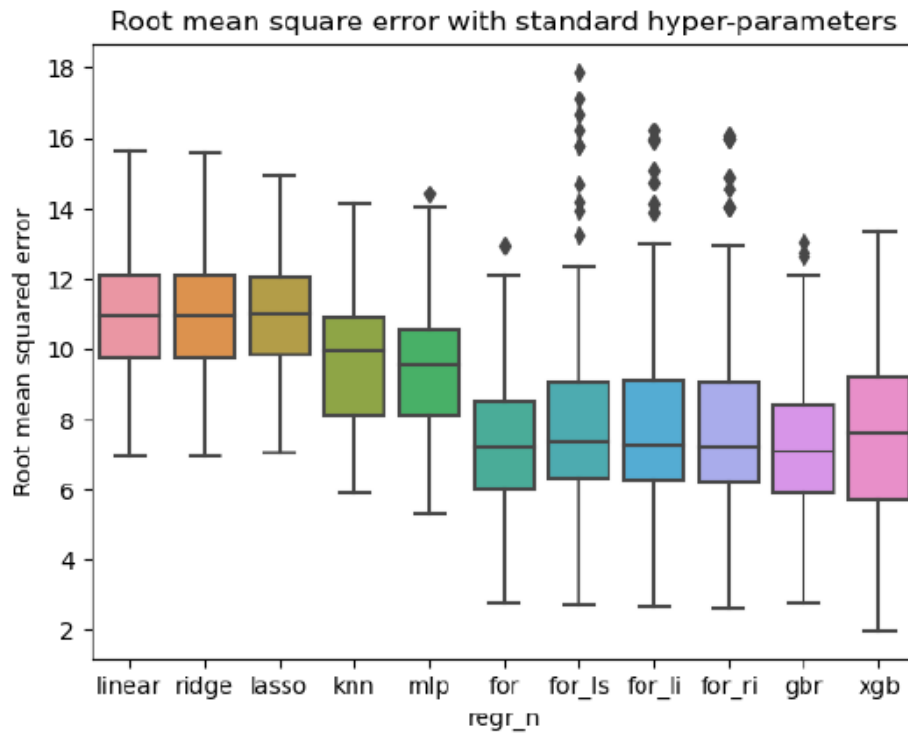
      test_neg_mean_squared_error test_neg_root_mean_squared_error  test_r2
      <lambda>                                <lambda>      mean
regr_n
for      55.880526                        7.252989  0.673062
for_li    66.223319                        7.782174  0.599209
```

for_ls	69.249181	7.913175	0.582127
for_ri	65.936510	7.765317	0.601074
gbr	55.843214	7.230265	0.671832
knn	96.248377	9.655994	0.453058
lasso	123.488375	10.987572	0.290719
linear	123.852352	10.972897	0.272985
mlp	92.056601	9.410421	0.446731
ridge	123.615101	10.963979	0.274790
xgb	62.493635	7.526831	0.641797

```
[56]: ax = sns.boxplot(x="regr_n", y="test_r2", data=gr_df)
plt.title('R2 scores with standard hyper-parameters')
plt.ylabel('val_r2')
plt.show()
```



```
[57]: ax = sns.boxplot(x=gr_df["regr_n"], y=-gr_df["test_neg_root_mean_squared_error"])
plt.title('Root mean square error with standard hyper-parameters')
plt.ylabel('Root mean squared error')
plt.show()
```



```
[107]: cv_results = []

for rs in r_states:
    models = [

        ("for", 7, RandomForestRegressor(random_state=rs), {
            "n_estimators": [400, 1000, 2000, 4000],
            "max_features": [None],
            "max_depth": [5, 10, 50, 100, None],
        }),

        ("gbr", 7, GradientBoostingRegressor(random_state=rs), {
            "n_estimators": [400, 800, 1000, 2000, 4000],
            "learning_rate": np.linspace(.0005, 1, 50),
            "subsample": [0.1, 0.25, 0.5, 0.75, 1.0],
            "max_depth": [5, 10, 50, 100, None],
        }),

        ("xgb", 7, XGBRegressor(random_state=rs), {
            "n_estimators": [400, 800, 1000, 2000, 4000],
```

```

        "learning_rate": np.linspace(.0005, 1, 50),
        "subsample": [0.1, 0.25, 0.5, 0.75, 1.0],
        "max_depth": [5, 10, 50, 100, None],
    }},
]
for regr_n, folds, regr, parameters in models:
    # Get the dataset
    X_train, y_train, X_test, y_test = get_dataset(data, rs=rs)
    # Find best parameters for model
    kf = KFold(folds, shuffle=True, random_state=rs)
    cv = GridSearchCV(regr, parameters, verbose=1, refit="r2", n_jobs=-1,
cv=kf, scoring=scoring)
    cv.fit(X_train, y_train)
    # Get predicted values
    y_pred = cv.best_estimator_.predict(X_test)
    # Save score of model for future reference
    cv_results.append({
        "regr_n": regr_n,
        "random_state": rs,
        "best": cv.best_estimator_,
        "best_params": cv.best_params_,
        "best_score": cv.best_score_,
        "mape": mean_absolute_percentage_error(y_test, y_pred),
        "mae": mean_absolute_error(y_test, y_pred),
        "mse": mean_squared_error(y_test, y_pred),
        "rmse": mean_squared_error(y_test, y_pred, squared=False),
        "r2": r2_score(y_test, y_pred),
    })
    print(cv_results[-1]['mape'], cv_results[-1]['r2'],
cv_results[-1]['best_score'])

# Final results
len(cv_results)

```

Fitting 7 folds for each of 20 candidates, totalling 140 fits

0.2980113015211964 0.6574286986802609 0.709498727118943

Fitting 7 folds for each of 6250 candidates, totalling 43750 fits

C:\Users\Asus\anaconda3\lib\site-

packages\sklearn\model_selection_search.py:978: RuntimeWarning: overflow encountered in square

(array - array_means[:, np.newaxis]) ** 2, axis=1, weights=weights

C:\Users\Asus\anaconda3\lib\site-packages\numpy\core_methods.py:179:

RuntimeWarning: overflow encountered in reduce

ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)

C:\Users\Asus\anaconda3\lib\site-

packages\sklearn\model_selection_search.py:969: UserWarning: One or more of the test scores are non-finite: [-159.87119555 -155.62127016 -152.59258417 ...

```

-393.7107815 -139.55104499
-101.05644582]
warnings.warn(
C:\Users\Asus\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:978: RuntimeWarning: invalid value
encountered in subtract
(array - array_means[:, np.newaxis]) ** 2, axis=1, weights=weights
C:\Users\Asus\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of the
test scores are non-finite: [-12.60813027 -12.44116653 -12.32271746 ...
-19.47700965 -11.52636735
-9.91671617]
warnings.warn(
C:\Users\Asus\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of the
test scores are non-finite: [ 0.17107983  0.19258676  0.2073941 ... -1.07934644
0.26929274
0.46962833]
warnings.warn(
0.2809564476366974 0.6981341214573953 0.7536661018506895
Fitting 7 folds for each of 6250 candidates, totalling 43750 fits
0.2619841416486356 0.6873500593600361 0.7455930781958852

```

[107]: 3

```
[108]: cv_df = pd.DataFrame(cv_results)
cv_df.groupby('regr_n')['best_score'].mean().sort_values(ascending=False)
```

```
[108]: regr_n
gbr    0.753666
xgb    0.745593
for    0.709499
Name: best_score, dtype: float64
```

```
[163]: # cv_df
```

```
[110]: best = cv_df.sort_values(by='best_score', ascending=False)['best'].head(1).
      <values[0]
best
```

```
[110]: GradientBoostingRegressor(learning_rate=0.04129591836734694, max_depth=5,
      n_estimators=4000, random_state=1, subsample=0.5)
```

```
[111]: best_params_gb = cv_df.sort_values(by='best_score',
      <ascending=False)['best_params'].head(1).values[0]
best_params_gb
```

```
[111]: {'learning_rate': 0.04129591836734694,
        'max_depth': 5,
        'n_estimators': 4000,
        'subsample': 0.5}

[112]: second_best = cv_df.sort_values(by='best_score', ascending=False)['best'].
        ↪head(2).values[1]
        second_best

[112]: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
                    colsample_bylevel=1, colsample_bynode=1, colsample_bytrees=1,
                    early_stopping_rounds=None, enable_categorical=False,
                    eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
                    grow_policy='depthwise', importance_type=None,
                    interaction_constraints='', learning_rate=0.02089795918367347,
                    max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
                    max_delta_step=0, max_depth=5, max_leaves=0, min_child_weight=1,
                    missing=nan, monotone_constraints='()', n_estimators=4000,
                    n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=1,
                    ...)

[113]: best_params_xgb = cv_df.sort_values(by='best_score', ↪
        ↪ascending=False)['best_params'].head(2).values[1]
        best_params_xgb

[113]: {'learning_rate': 0.02089795918367347,
        'max_depth': 5,
        'n_estimators': 4000,
        'subsample': 0.75}

[114]: third_best = cv_df.sort_values(by='best_score', ascending=False)['best'].
        ↪head(3).values[2]
        third_best

[114]: RandomForestRegressor(max_depth=50, max_features=None, n_estimators=1000,
                             random_state=1)

[79]: best_params_rf = cv_df.sort_values(by='best_score', ↪
        ↪ascending=False)['best_params'].head(3).values[2]
        best_params_rf

[79]: {'max_depth': 50, 'max_features': None, 'n_estimators': 1000}

[118]: feature_importance = best.feature_importances_
        sorted_idx = np.argsort(feature_importance)
        pos = np.arange(sorted_idx.shape[0]) + 0.5
        fig = plt.figure(figsize=(12, 6))
```

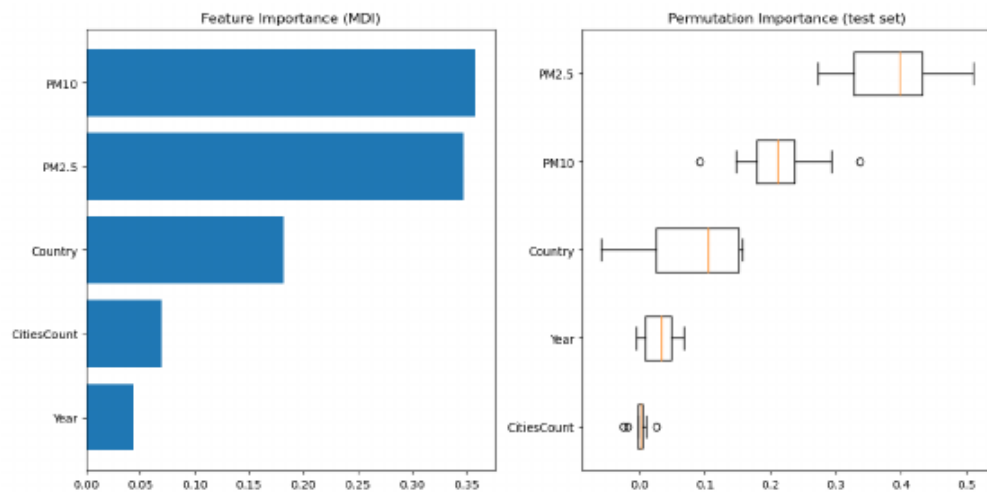


```

plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(data.columns.values)[sorted_idx])
plt.title("Feature Importance (MDI)")

result = permutation_importance(best, X_test, y_test, n_repeats=10,
    random_state=1, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(result.importances[sorted_idx].T, vert=False, labels=np.array(data.
    columns.values)[sorted_idx],)
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()

```



```

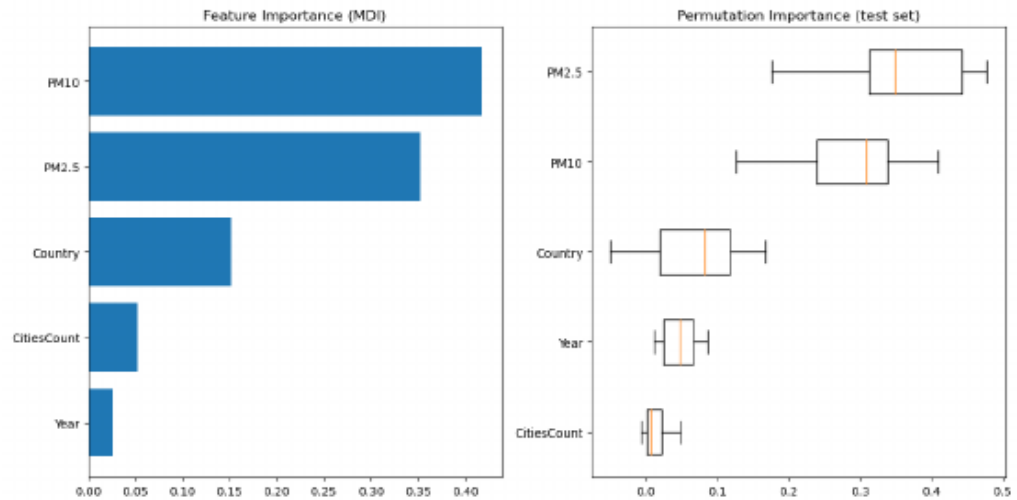
[119]: feature_importance = third_best.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(data.columns.values)[sorted_idx])
plt.title("Feature Importance (MDI)")

result = permutation_importance(third_best, X_test, y_test, n_repeats=10,
    random_state=1, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)

```

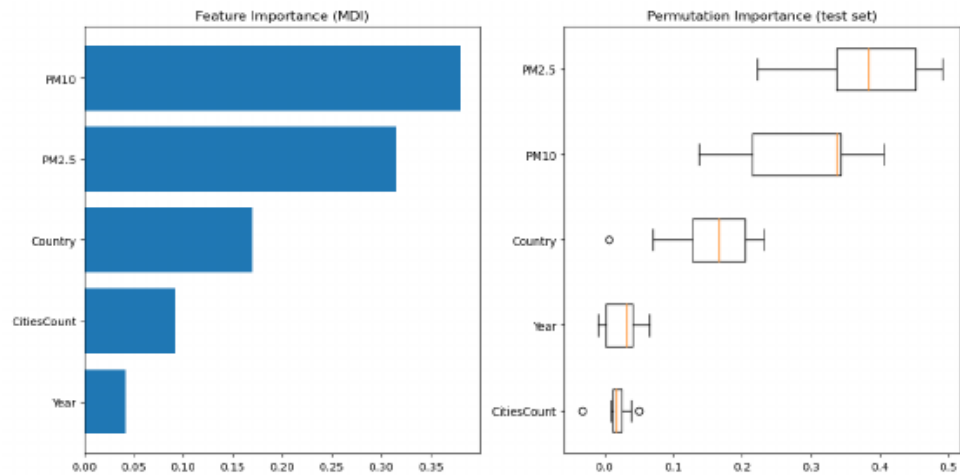


```
plt.boxplot(result.importances[sorted_idx].T, vert=False, labels=np.array(data.
    columns.values)[sorted_idx],)
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
```



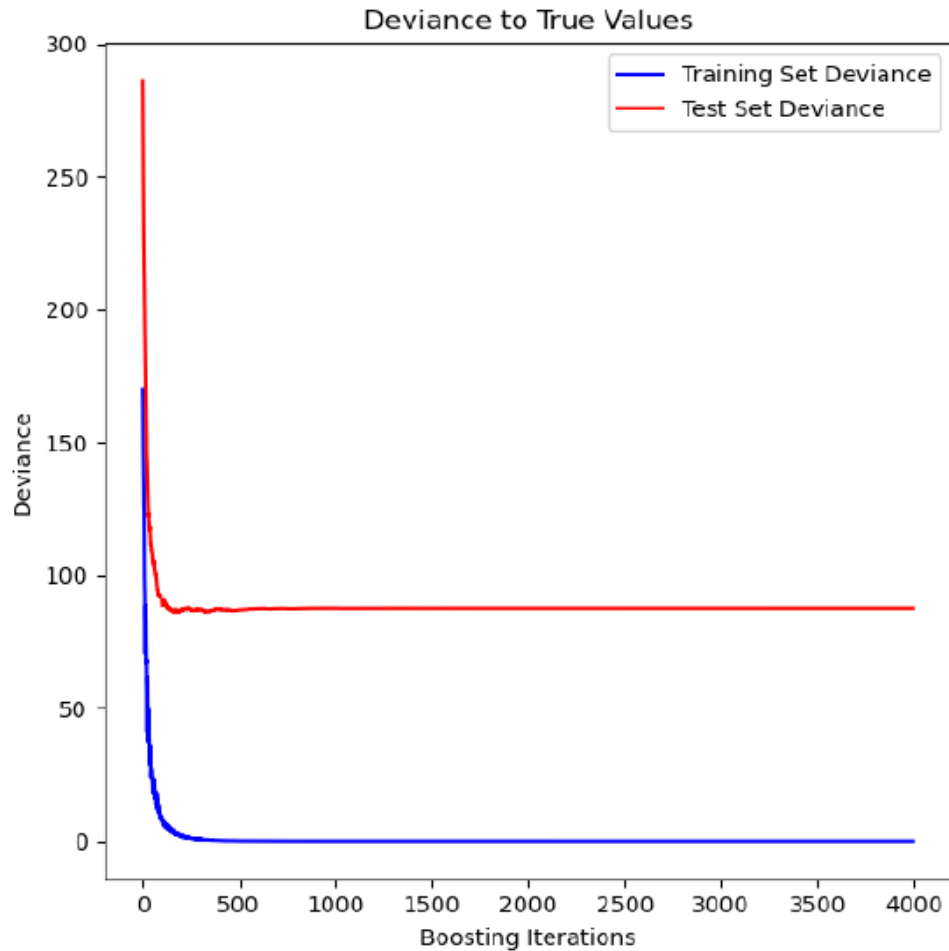
```
[120]: feature_importance = second_best.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(data.columns.values)[sorted_idx])
plt.title("Feature Importance (MDI)")

result = permutation_importance(second_best, X_test, y_test, n_repeats=10,
    random_state=1, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(result.importances[sorted_idx].T, vert=False, labels=np.array(data.
    columns.values)[sorted_idx],)
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
```



```
[121]: test_score = np.zeros((len(best.estimators_),), dtype=np.float64)
for i, y_pred in enumerate(best.staged_predict(X_test)):
    test_score[i] = best.loss_(y_test, y_pred)

fig = plt.figure(figsize=(6, 6))
plt.subplot(1, 1, 1)
plt.title("Deviance to True Values")
plt.plot(
    np.arange(len(best.estimators_)) + 1,
    best.train_score_,
    "b-",
    label="Training Set Deviance",
)
plt.plot(
    np.arange(len(best.estimators_)) + 1, test_score, "r-", label="Test Set Deviance"
)
plt.legend(loc="upper right")
plt.xlabel("Boosting Iterations")
plt.ylabel("Deviance")
fig.tight_layout()
plt.show()
```

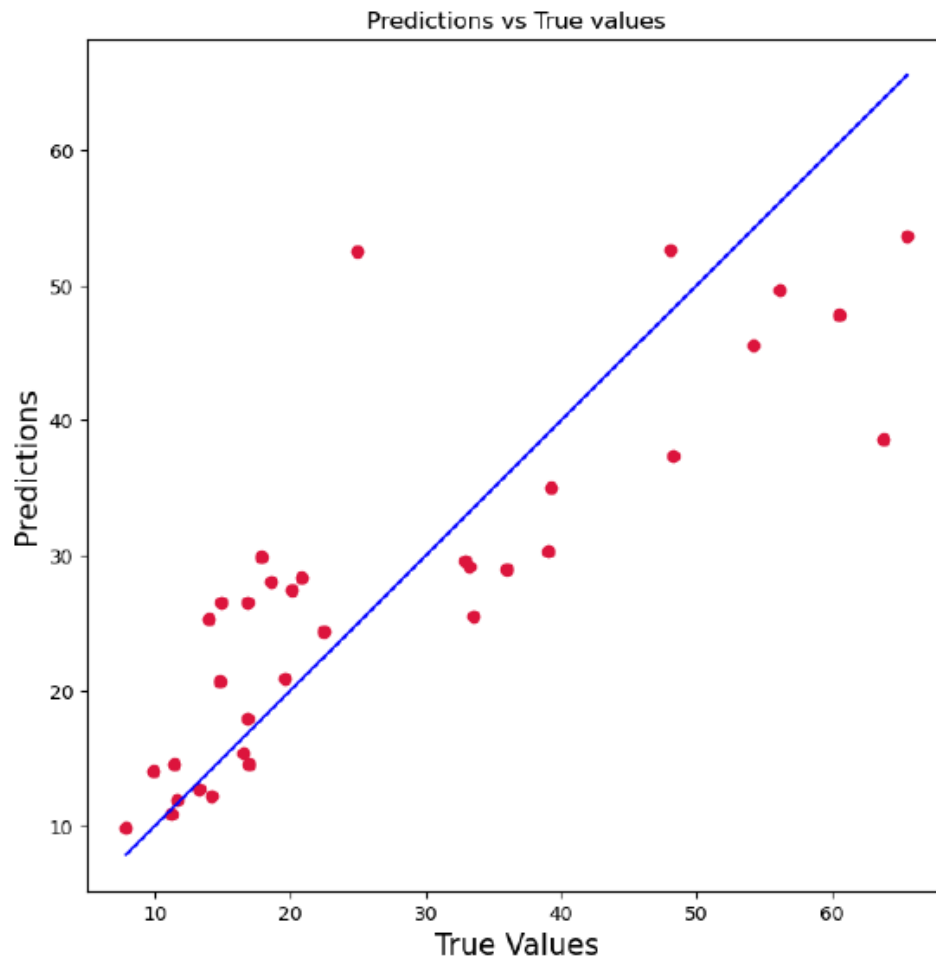


```
[124]: _, _, X_test, y_test = get_dataset(data, rs=rs)
y_pred = best.predict(X_test)

plt.figure(figsize=(8,8))
plt.scatter(y_test, y_pred, c='crimson')
#plt.yscale('log')
#plt.xscale('log')

p1 = max(max(y_pred), max(y_test))
p2 = min(min(y_pred), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
```

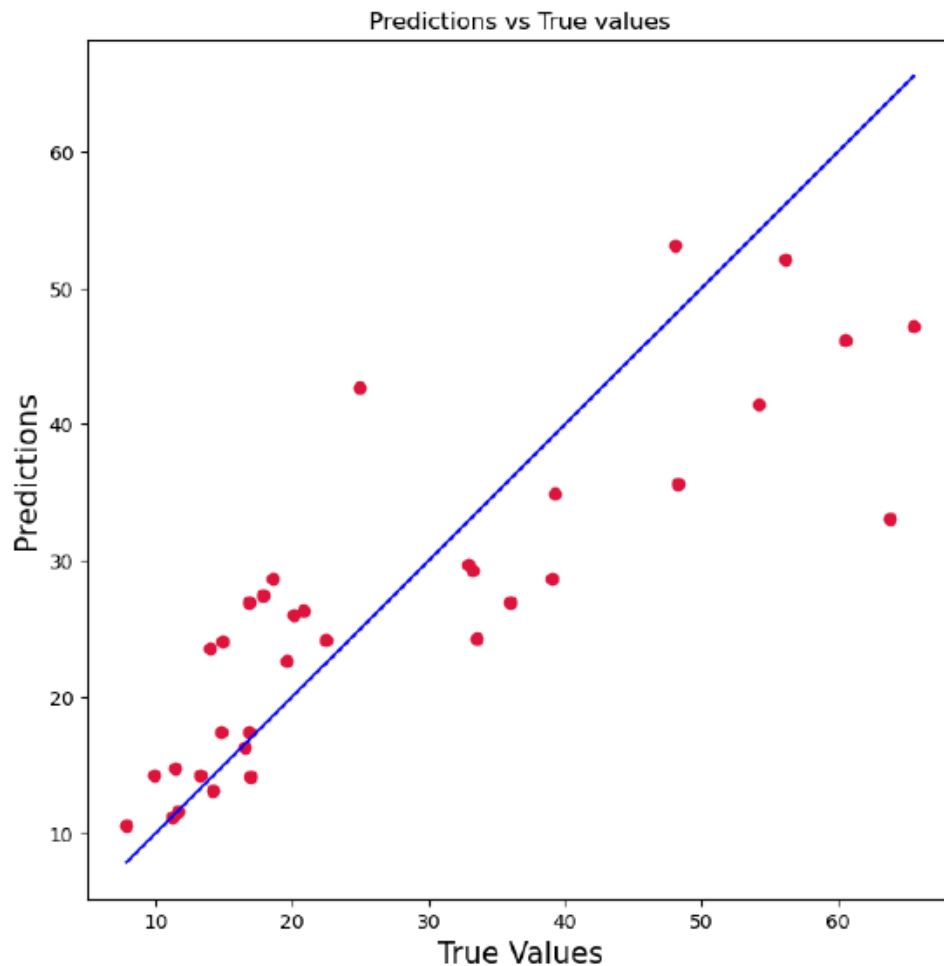
```
plt.axis('equal')
plt.title("Predictions vs True values")
plt.show()
```



```
[125]: _,_, X_test, y_test = get_dataset(data, rs=rs)
y_pred = second_best.predict(X_test)

plt.figure(figsize=(8,8))
plt.scatter(y_test, y_pred, c='crimson')
#plt.yscale('log')
#plt.xscale('log')
```

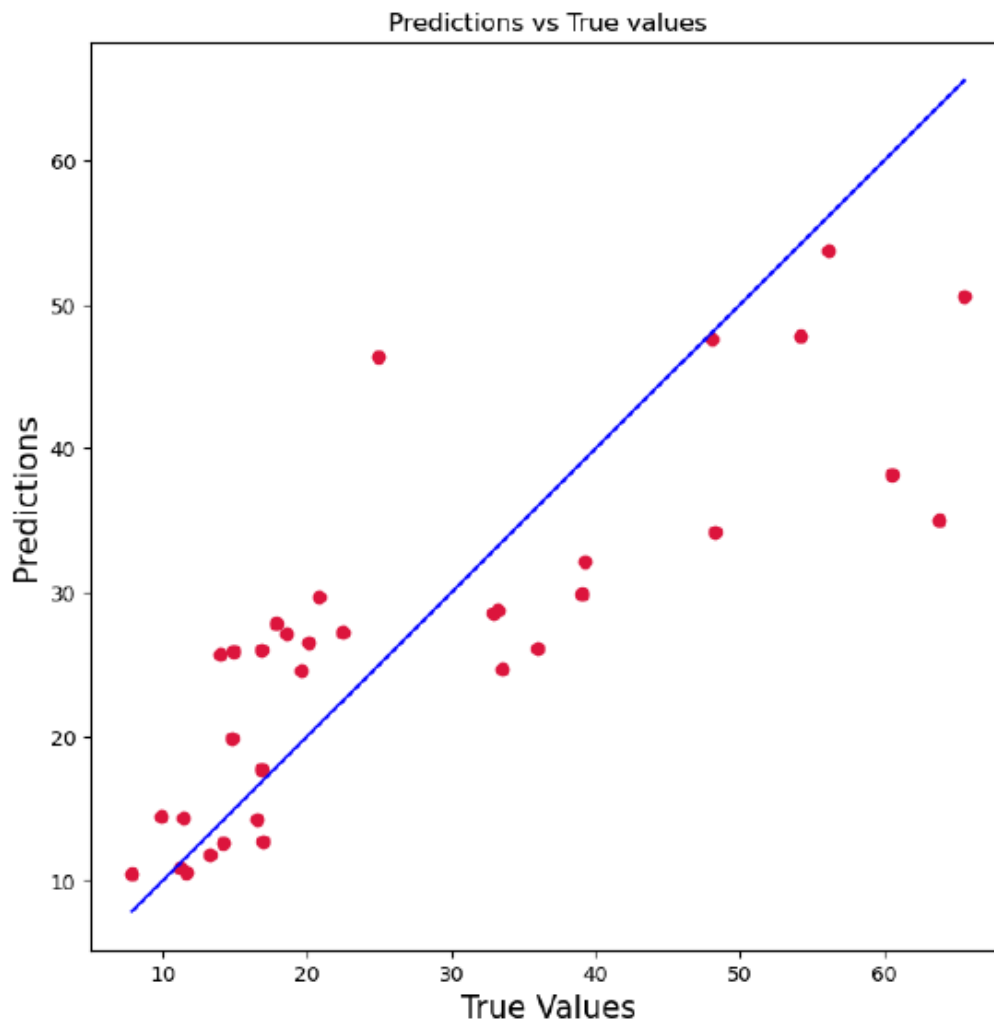
```
p1 = max(max(y_pred), max(y_test))
p2 = min(min(y_pred), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title("Predictions vs True values")
plt.show()
```



```
[127]: _, _, X_test, y_test = get_dataset(data, rs=rs)
y_pred = third_best.predict(X_test)
```

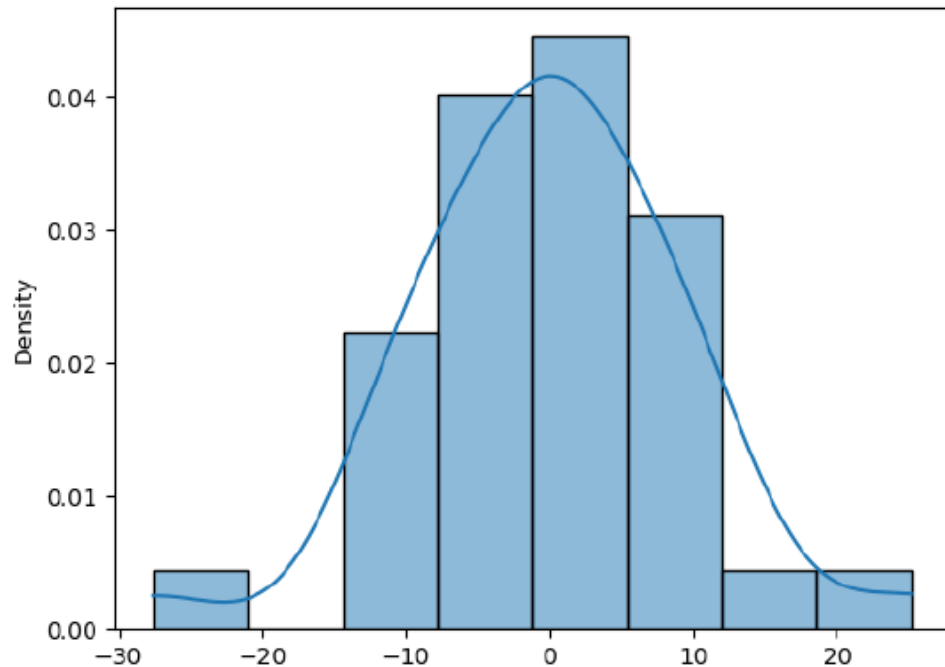
```
plt.figure(figsize=(8,8))
plt.scatter(y_test, y_pred, c='crimson')
#plt.yscale('log')
#plt.xscale('log')

p1 = max(max(y_pred), max(y_test))
p2 = min(min(y_pred), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.title("Predictions vs True values")
plt.show()
```



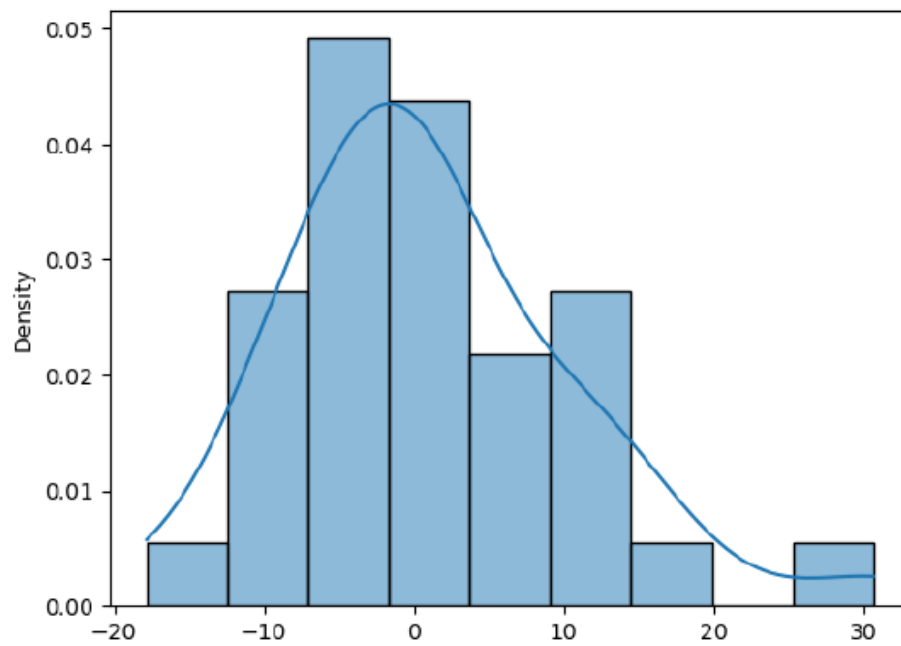
```
[153]: # sns.distplot(y_test-(best.predict(X_test)))
# plt.xlabel('y_test - y_predicted')
# plt.show()

sns.histplot((y_test-(best.predict(X_test))), kde=True, stat="density")
plt.show()
```



```
[154]: sns.histplot((y_test-(second_best.predict(X_test))), kde=True, stat="density")
plt.show()

# sns.distplot(y_test-(second_best.predict(X_test)))
# plt.xlabel('y_test - y_predicted')
# plt.show()
```



```
[155]: # sns.distplot(y_test-(third_best.predict(X_test)))  
# plt.xlabel('y_test - y_predicted')  
# plt.show()  
  
sns.histplot((y_test-(third_best.predict(X_test))), kde=True, stat="density")  
plt.show()
```