Calibration of sensors in an Air Pollution Monitoring Network

Arnau Abella

June 2, 2021

1 Introduction

1.1 Goal

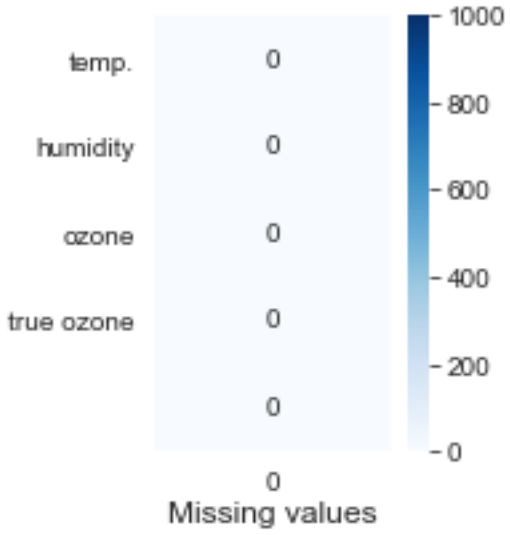
The objective of this project is to calibrate an air pollution sensor in an air pollution monitoring sensor network. We will take the data sampled of one node that accommodates three sensors: a MIC2614 O_3 sensor, a temperature sensor and a relative humidity sensor.

1.2 The data

The dataset has 1000 samples and 5 features:

- Date: UTC
- Temperature: $°C$ ($\mathbb{R}$)
- Humidity: % ($\mathbb{R}$)
- O_3: $K\Omega$ ($\mathbb{R}$)
- True O_3: $\mu gr/m^3$ ($\mathbb{R}$)

Before we start with the exploration data analysis (EDA) we need to clean our dataset and deal with the missing values.

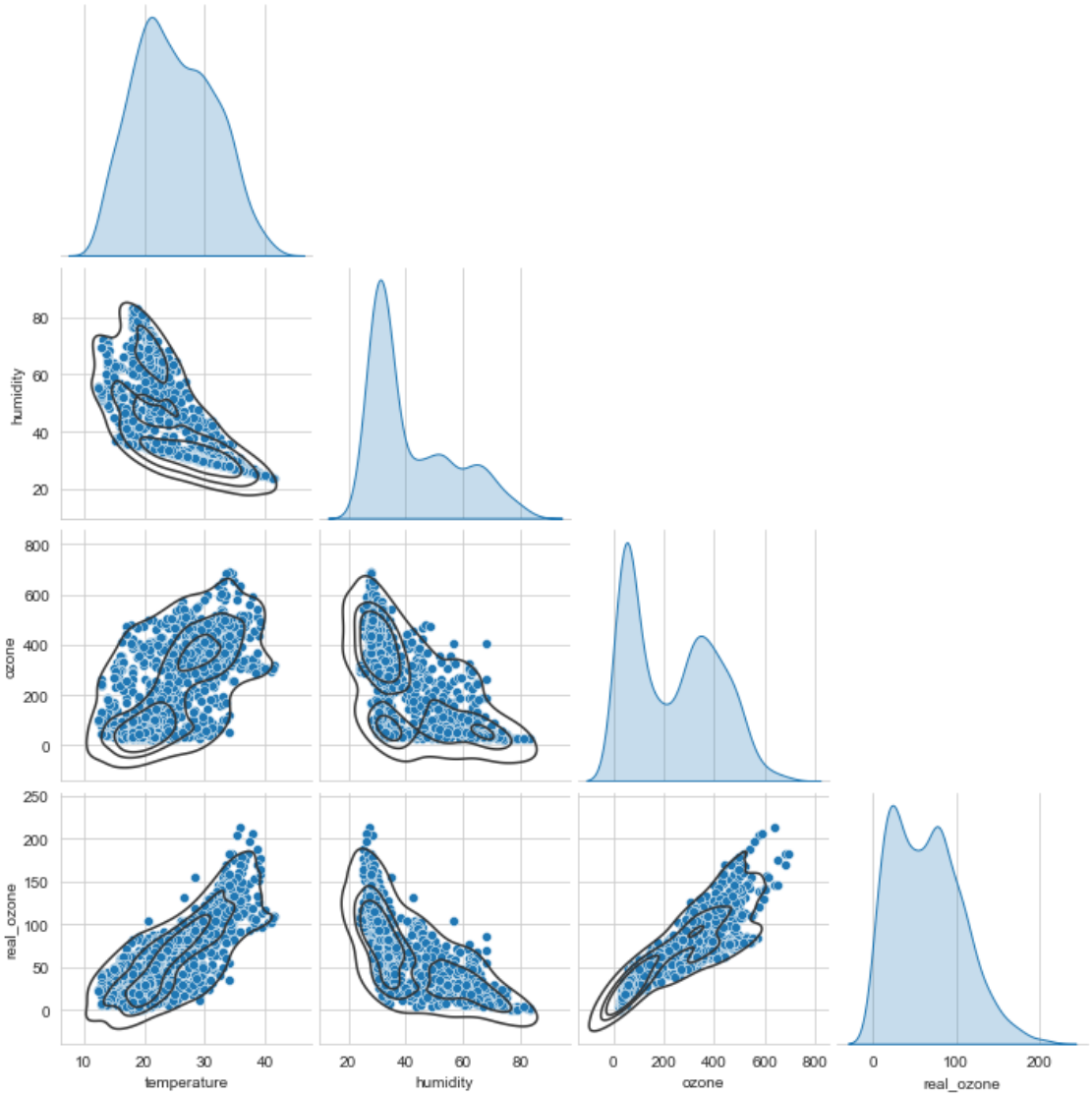


Lucky, the dataset has already been preprocessed in terms of missing values so we can jump into EDA.

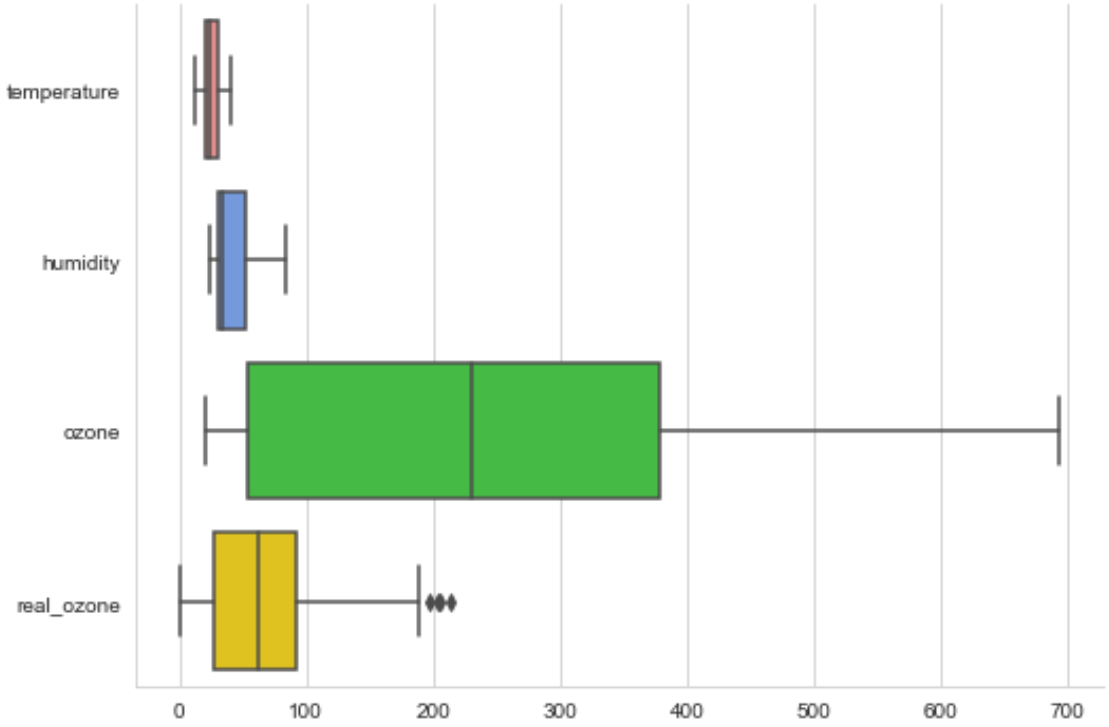
2 Part I: Exploratory Data Analysis

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

First, we are going to use a *pair plot*. A pairs plot allows us to see both distribution of single variables and relationships between two variables. Pair plots are a great method to identify trends for follow-up analysis.



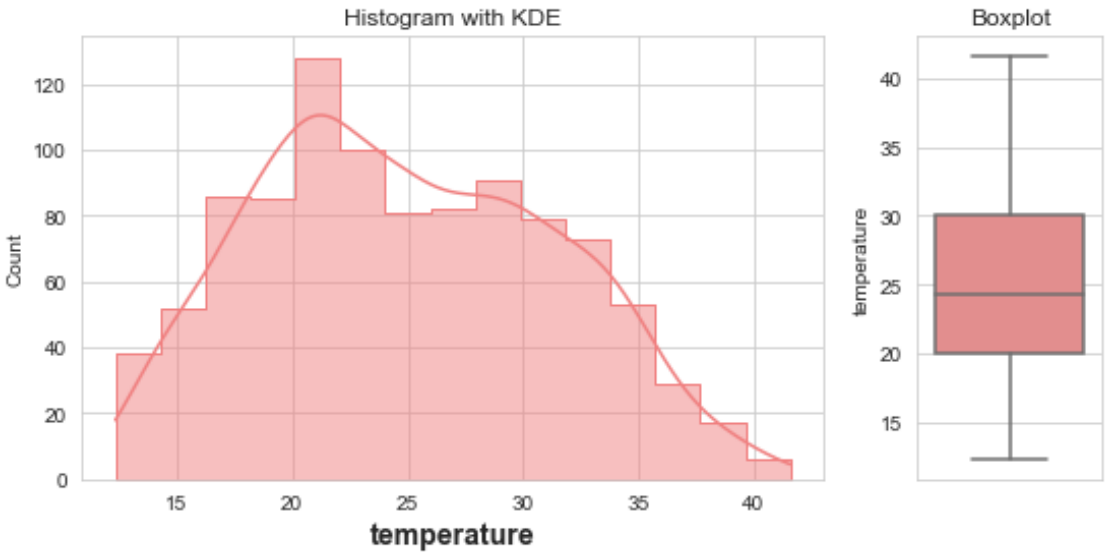
Then, we are going to use a *box plot* to visualize each different feature and its range. The box plot allow us to easily visualize the standard deviation and the outliers of our data.

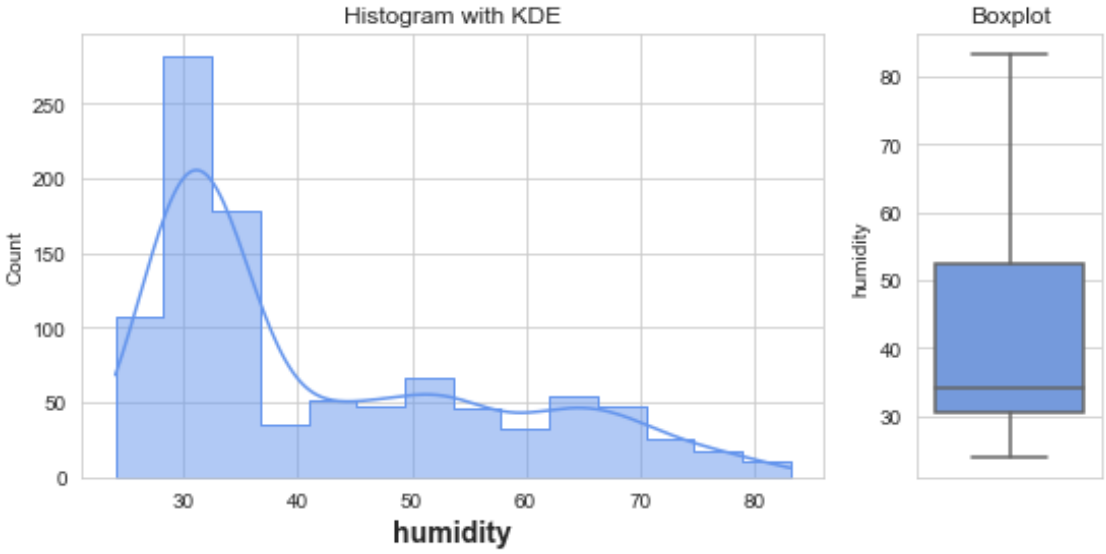


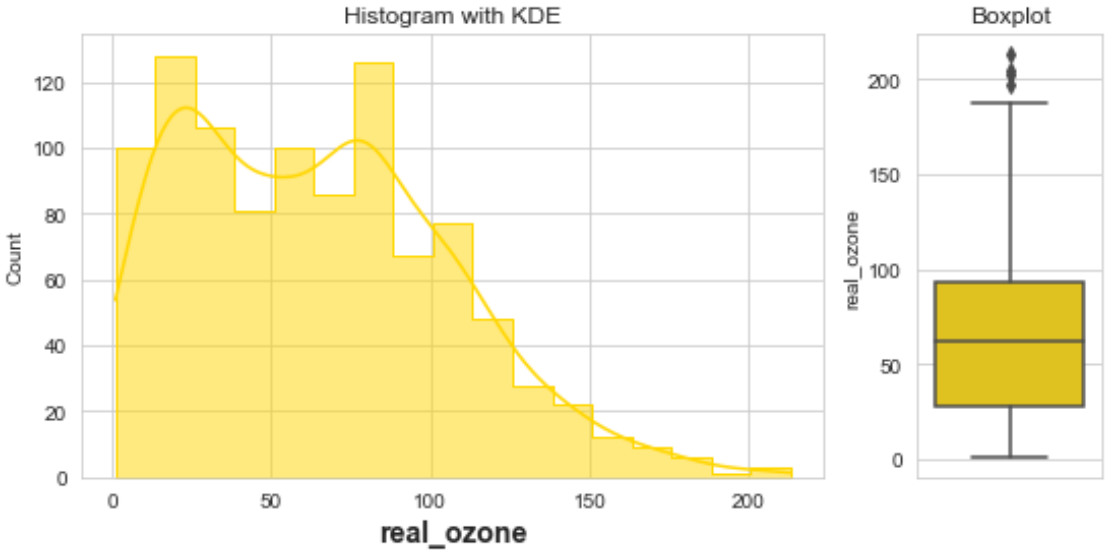
We can observe in the box plot that the data is clearly not *normalized* so we will scale it during the preprocessing.

We also see that O_3 and O_3(true) have some *outliers*.

Let's have a look at the distribution of each feature using a *histogram* and the *kernel density estimation*.


Histogram with KDE
Count
temperature
Boxplot
temperature

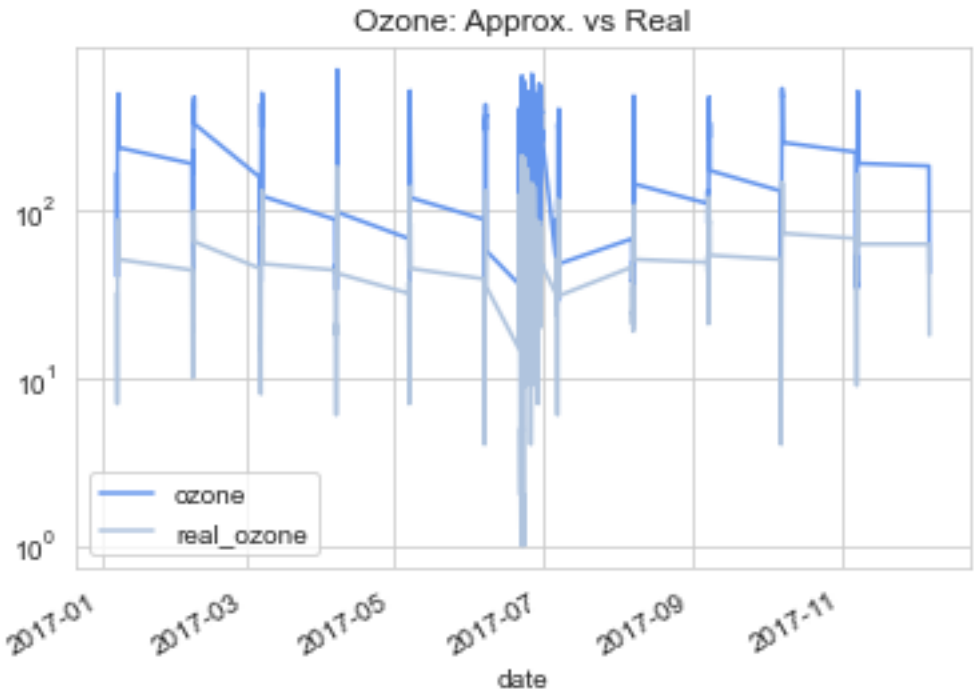

Histogram with KDE
Count
humidity
Boxplot
humidity



All three humidity, O_3, and O_3(true) are *skewed*. We will need to take this into consideration during the modeling since some of our models will assume that each feature follows a *normal distribution*.

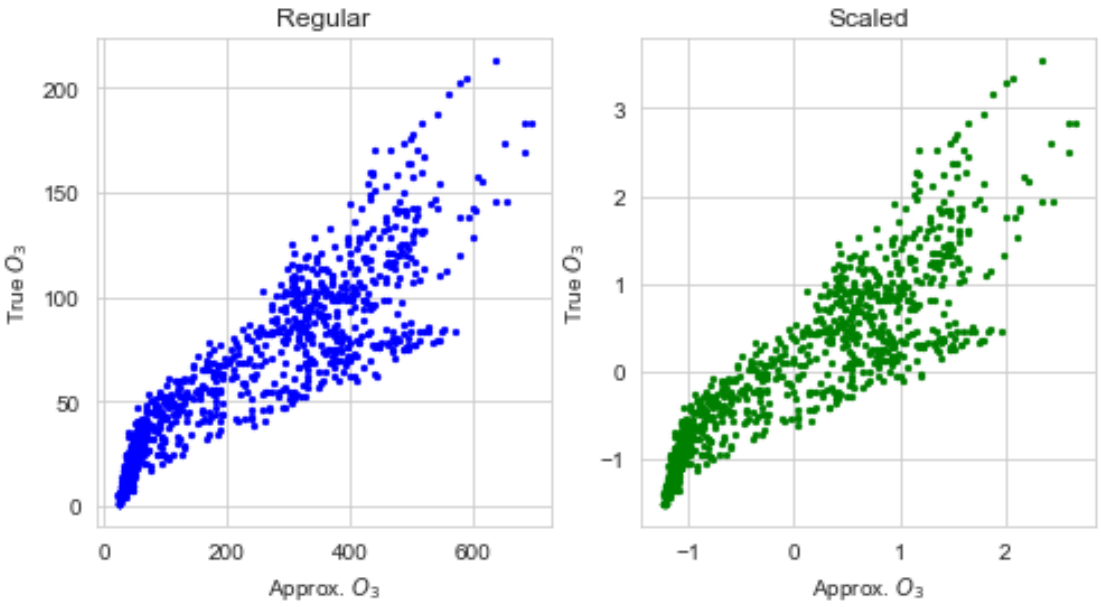
Next, we are going to have a look at the relation between the variables.

The scale of *ozone* and *true ozone* are different. The ozone sensor works as a voltage divisor which is represented as a variable resistor ($k\Omega$) while the reference O_3 is measured in $\mu gr/m^3$.



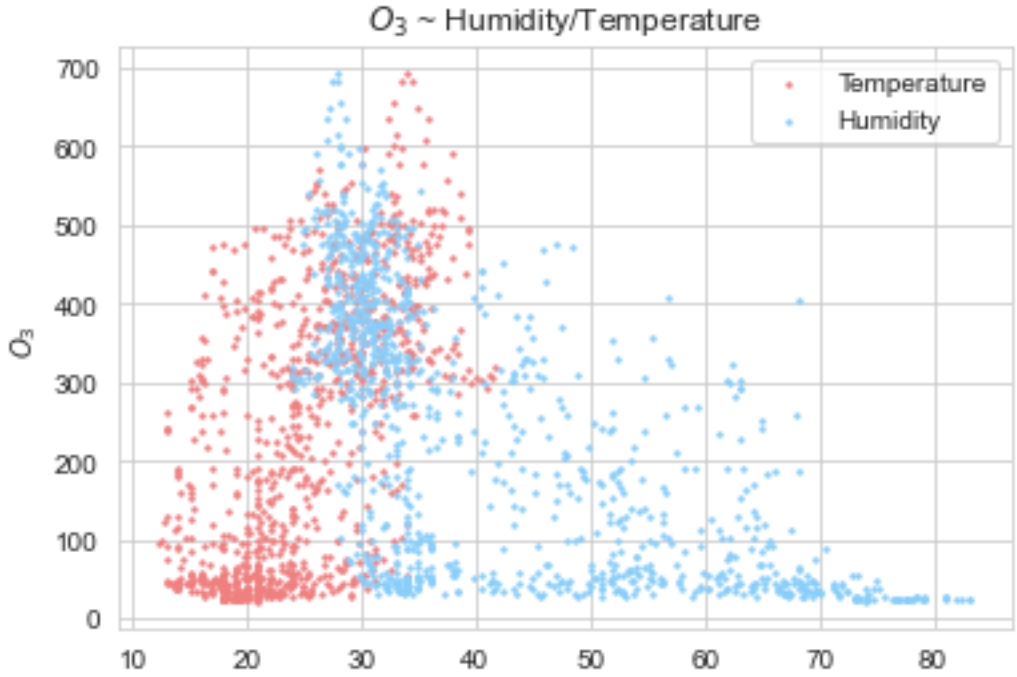
Despite being in different scales, clearly both variables follow the same pattern.

If the MIC2614 O_3 sensor was as good as the reference station, we could use it directly with no calibration. But, as we will see in the next plot, the MIC2614 O_3 is not enough to predict the real O_3: a perfect prediction would lie in the diagonal of the plot which is not the case.



Also, the *standarization* does not change the pattern.

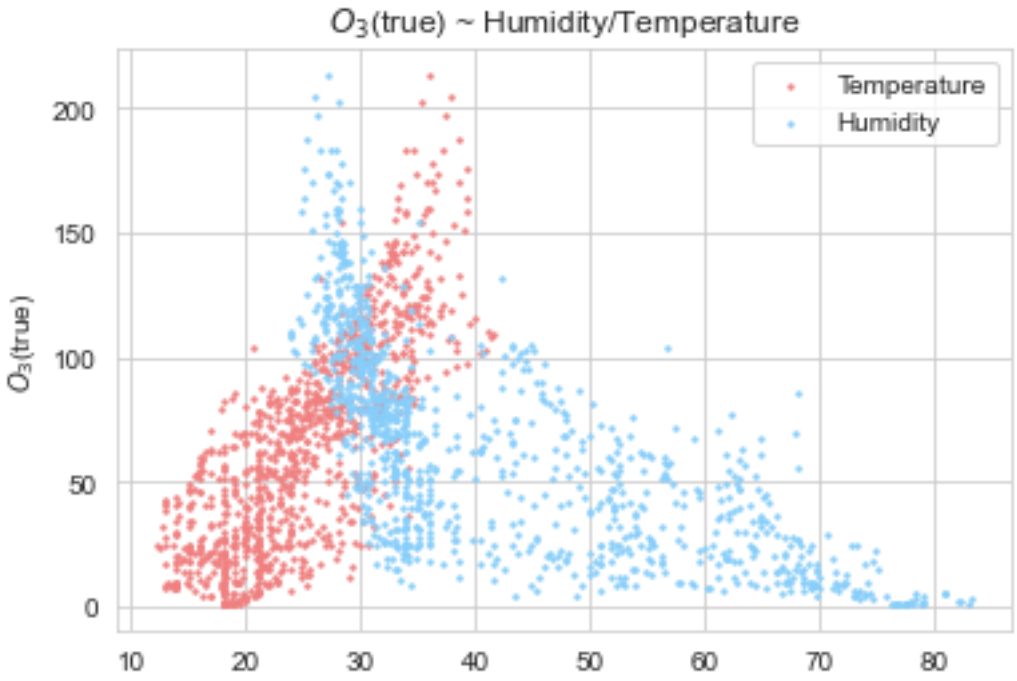
Now, we plot the temperature and the humidity against the O_3 to see if there is any pattern.



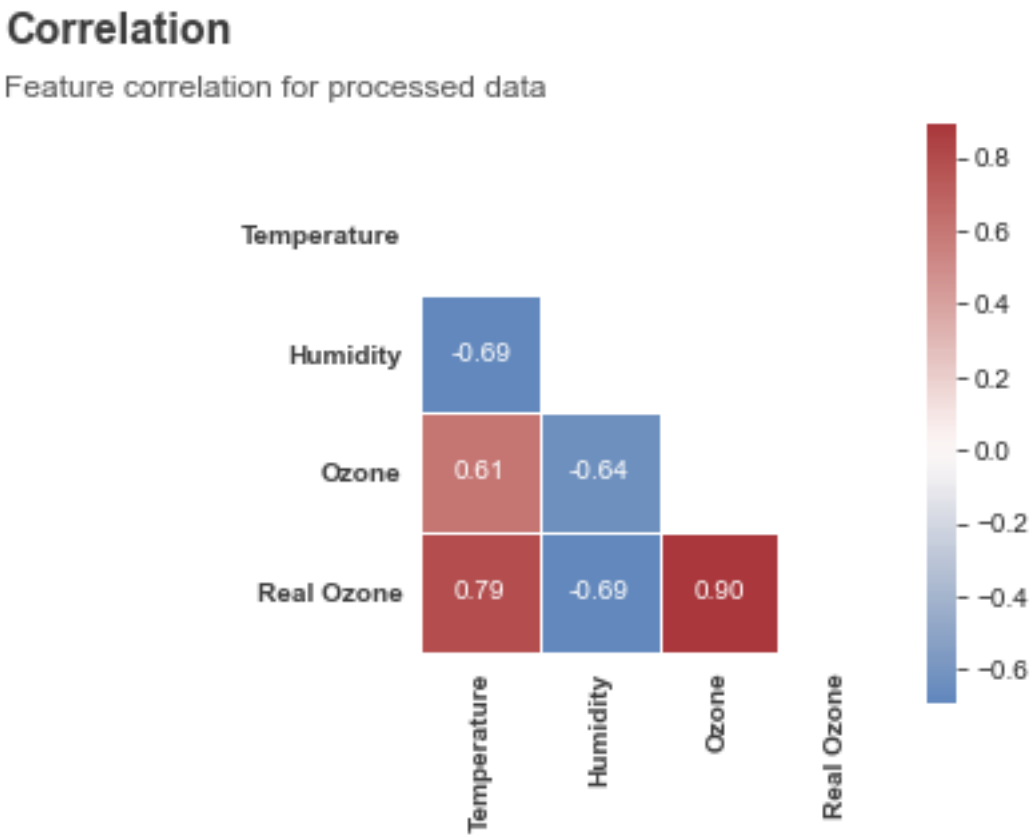
The relation between those variables is the following:

- The higher the temperature, the higher the O_3.
- The lower the humidity, the higher the O_3.

Let's try to plot the same variables against the reference O_3.



These relations are clearer on the correlation plot that we will see next.



- There is high correlation between temperature and O_3.
- There is high negative correlation between humidity and O_3.
- There is very high correlation between O_3 and reference O_3.

3 Part II: Calibration

Once EDA is finished we can start working on the calibration.

3.1 Clean up

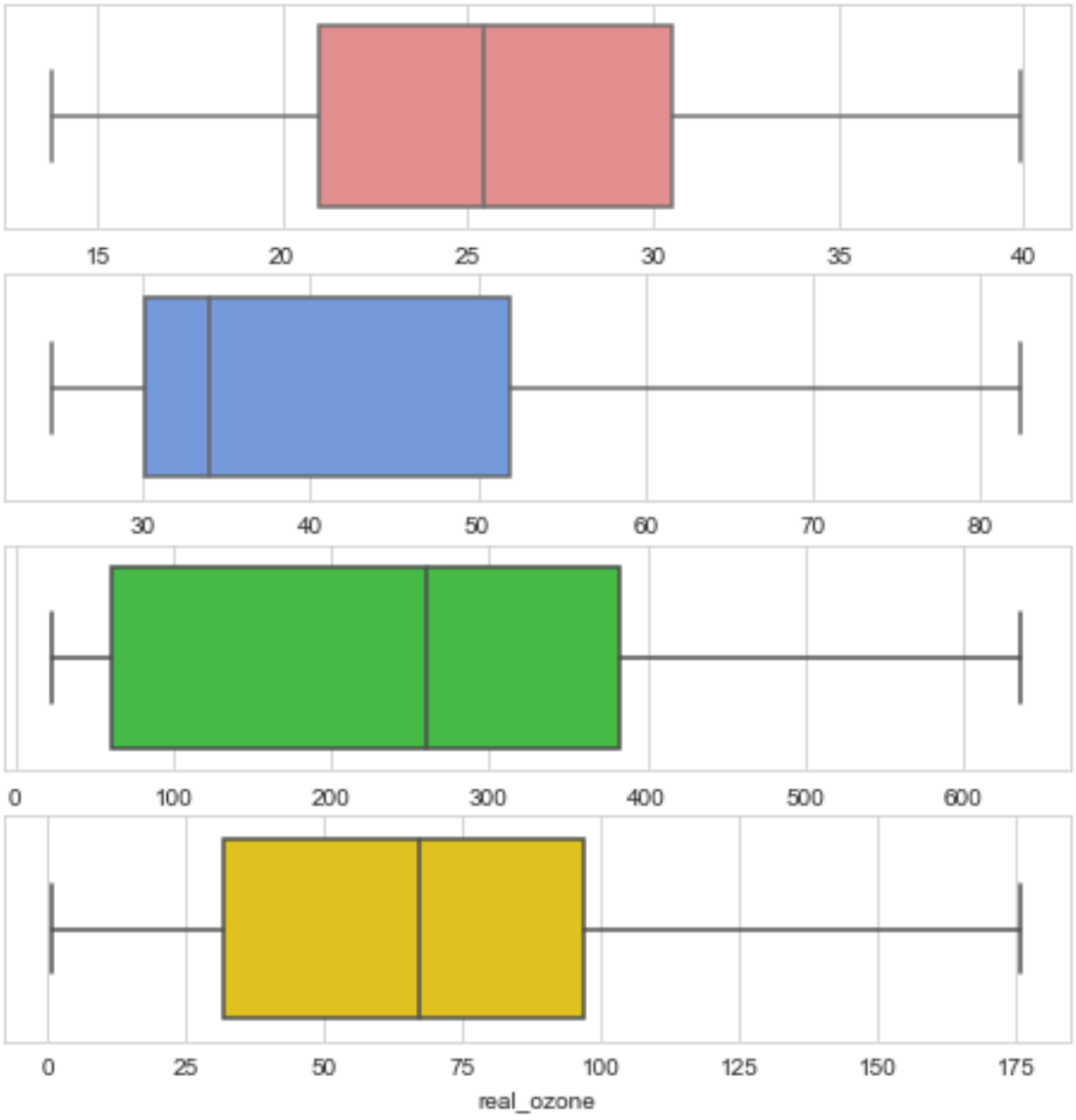
Before jumping into the modeling, we first need to clean up our dataset:

- Remove the outliers.
- Normalize the features.

There are several methods to detect outliers:

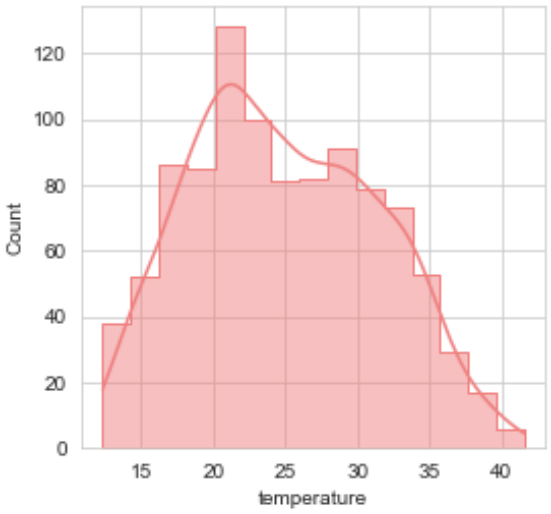
- Robust covariance
- One-Class SVM
- Isolation Forest
- Local Outlier Factor

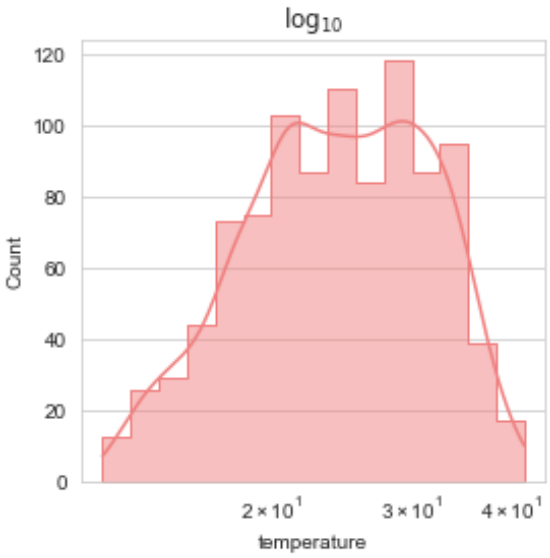
We decided to go with *Local Outlier Factor* since it is efficient and works well in both low and high dimensional datasets.

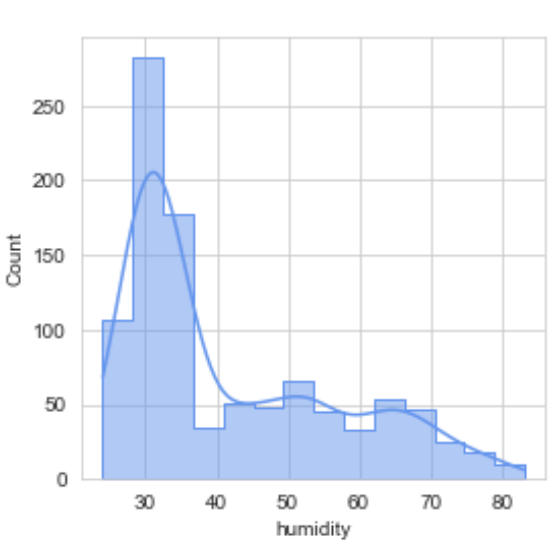


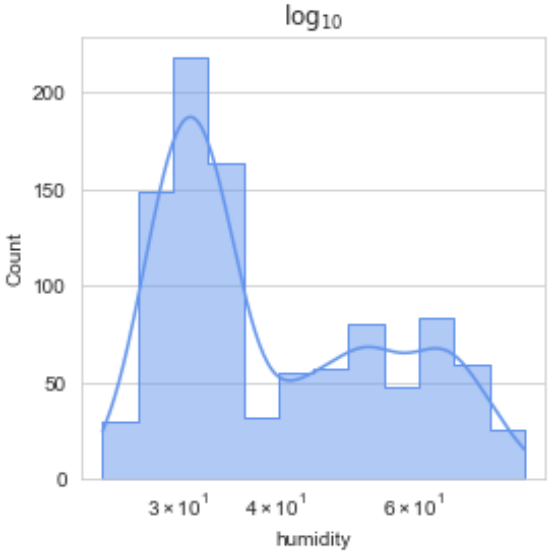
The number of samples after removing *anomalies* (*novelties* and *outliers*) is 789. We will talk later in resampling about how to properly remove anomalies without *leaking* data.

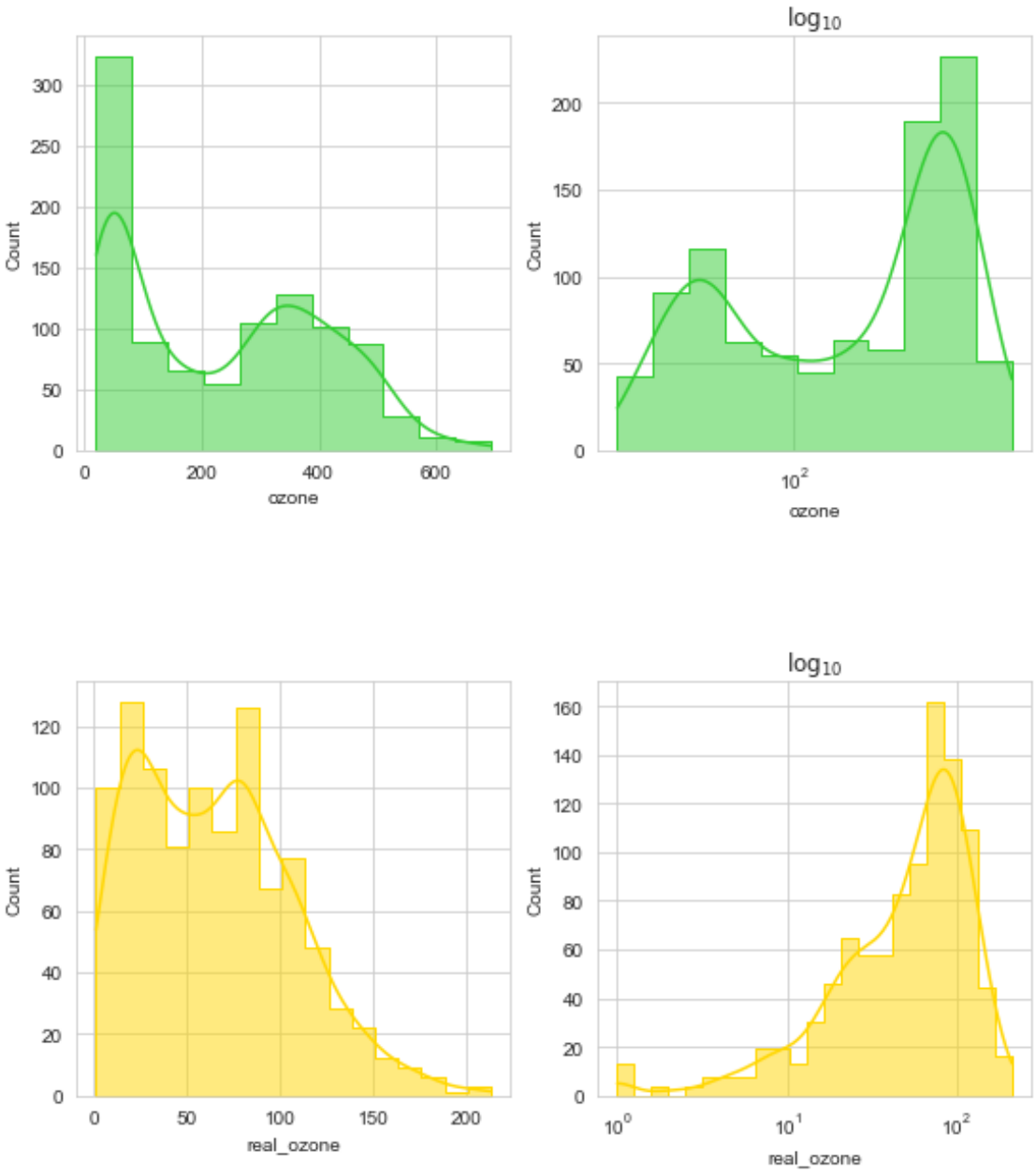
Next, we deal with skewed distributions by applying the *logarithm*, as all values are positive.


Count
120
100
80
60
40
20
0
15
20
25
30
35
40
temperature


log10
Count
120
100
80
60
40
20
0
2×10^1
3×10^1
4×10^1
temperature


Count
250
200
150
100
50
0
30
40
50
60
70
80
humidity


log10
Count
200
150
100
50
0
3×10^1
4×10^1
6×10^1
humidity



The distributions for temperature and true O_3 seem to be fixed but both humidity and O_3 still have this multivariate normal distribution shape. We will see later this reflected in the normality tests.

To finish the clean up, we are going to remove the *date* column. There is a clear dependency between the date and O_3 in order to simplify the analysis we are going to consider both variables independent.

3.2 Modeling

Once the dataset is clean up of anomalies, we are ready to start looking for the best statistical model i.e. the model that best approximates to the real data distribution.

3.2.1 Preprocessing and Resampling

We want our model to be *robust* i.e. to work well for unseen data. If we *overfit* the model to the training data, it will not work well for new data. To prevent overfiting, we are going to split our samples in:

- Training data: 580 samples
- Validation data: 200 samples
- Test data: 200 samples

First, we are going to train our models with the training data and then pick the best one using the validation data. This way, we will avoid picking an overfitted model. Finally, we are going to use the best model and validate it against the test data which is a good approximation of how the model is going to behave in the reality.

To prevent data leakage we are going to preprocess each training/validation/test set independently. The *preprocess* will consist of:

- Remove outliers.
- Standarization of the features.

As a standarization method we will use the *Robust Scaler* which works well in the presence of outliers.

3.2.2 Metrics

We are going to use the following regression metrics:

$$MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i|$$

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}$$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Recall that $MAE \leq RMSE$. If all of the errors have the same magnitude, then $RMSE = MAE$.

$|RMSE - MAE|$ is a good indicator of the presence of anomalies in the data.

3.2.3 Model Selection

We are going to explore the following models:

- Linear Regression, Ridge Regression, and SGD Linear Regression.

- k-Nearest Neighbors Regression
- Random Forest Regression
- Kernel Regression
- Gaussian Process
- Support Vector Regression
- Neural Networks

First, we will evaluate the models with *default parameters*.

Then, we are going to carefully tune the *hyperparameters* to get the best score out of them.

	MAE	RMSE	R^2
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best model with default parameters is the Random Forest followed by K-Nearest Neighbors and Linear Regression.

The coefficient of determination of the first models (≈ 0.95) is really good. This coefficient indicates how good our model fits the data and how well unseen samples are likely to be predicted by the model.

Hypertunning Hypertunning the parameters is a complicated process. You need to know well the model in order to find the best hyperparameters.

There are several techniques to find the best hyperparameters of the models. We are going to combine an exhaustive search with domain knowledge in order to reduce the range of the hyperparameters and make the search computationally feasible.

In each of the followings sections, we are going to optimize a model. Starting from the simplest one, a *Linear Regression*, to the most complex one, a *Neural Network*.

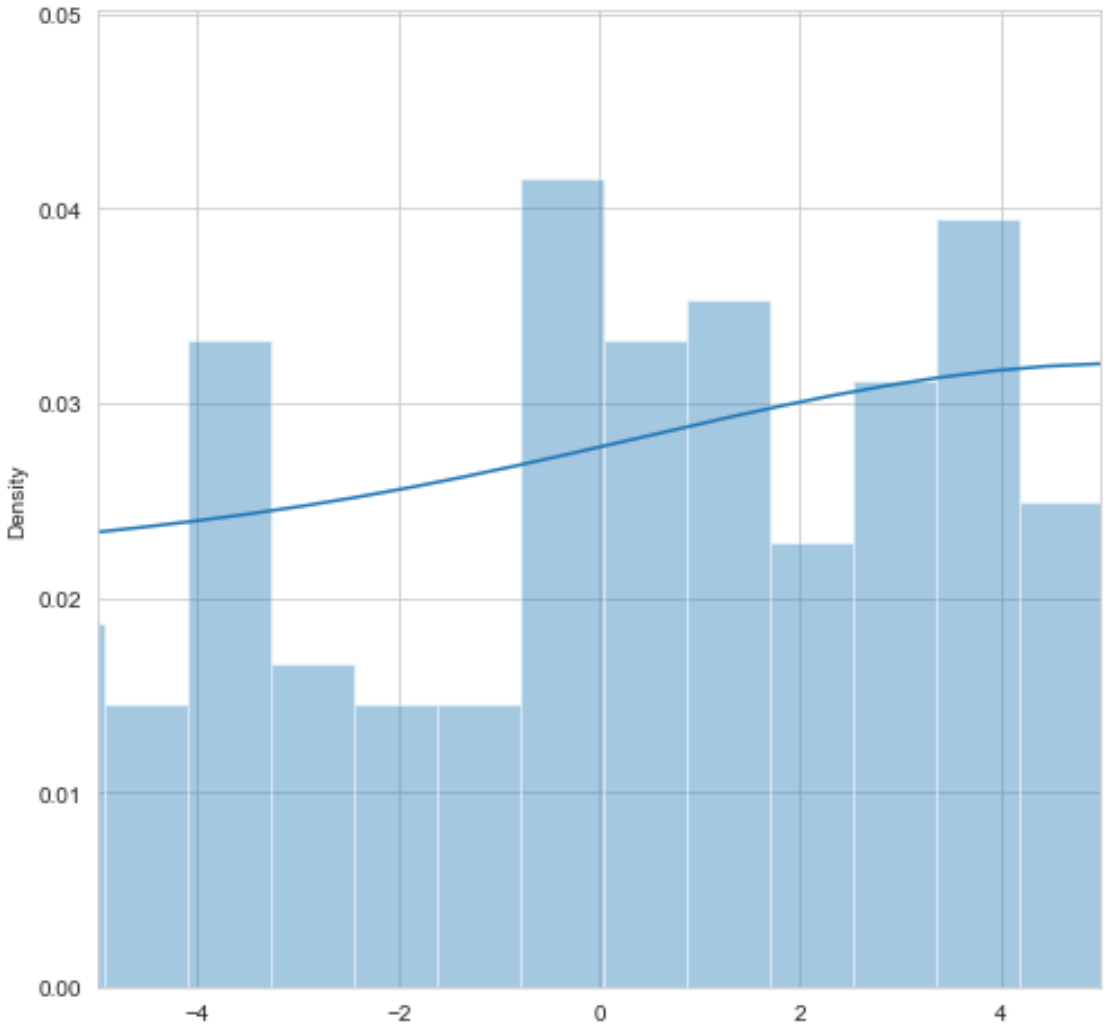
Multiple Linear Regression Before jumping into the optimization, we are going to check the *Linear Regression* assumptions:

- Linear relationship
- Independence of residuals
- Homoscedasticity
- Normality of residuals

There are multiple ways to tests this assumptions. We are going to employ the following:

- Durbin-Watson test
- Jarque-Bera test
- QQ-plot

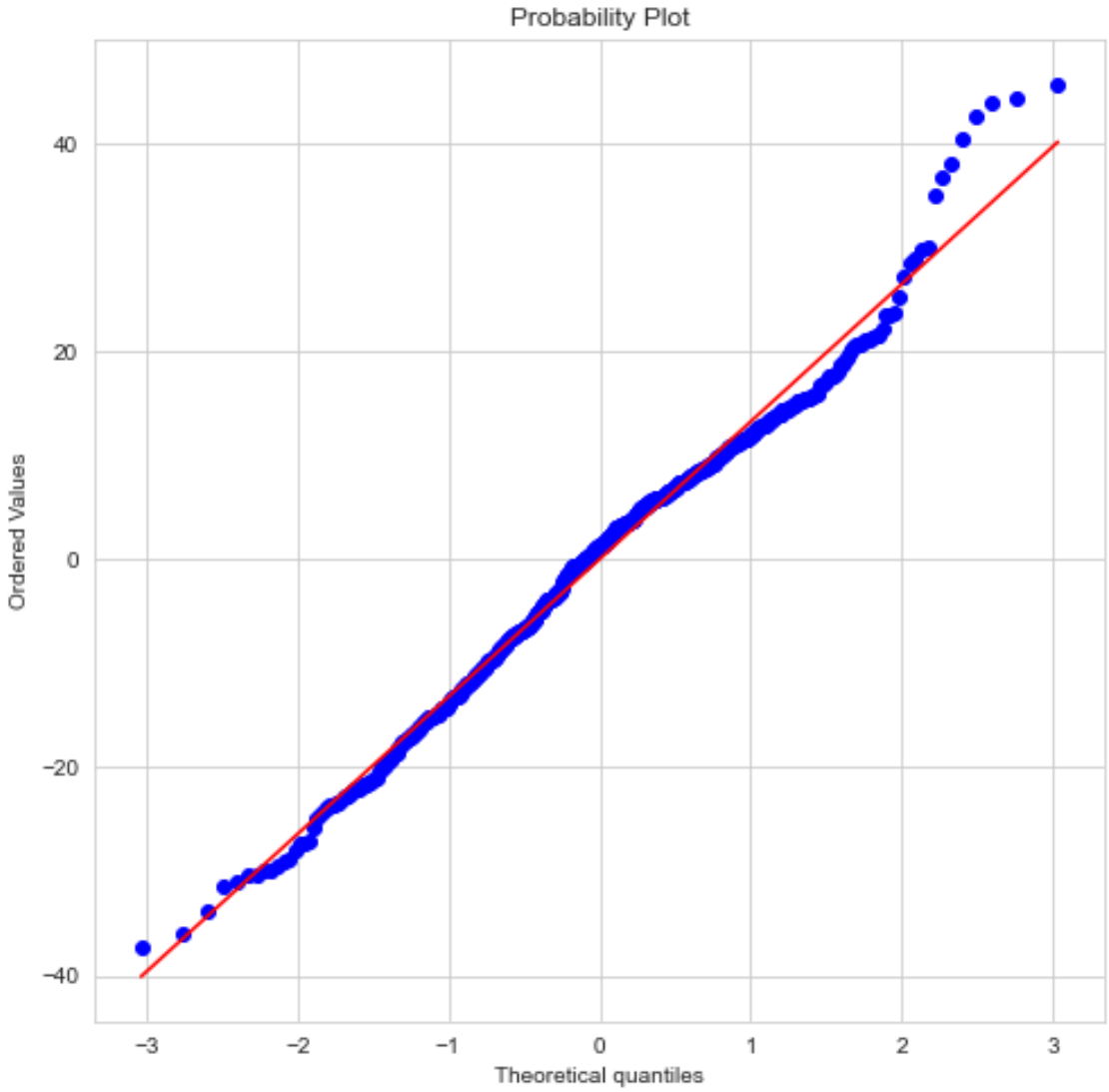
[35]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             real_ozone   R-squared:                       0.892
Model:                            OLS   Adj. R-squared:                  0.892
Method:                 Least Squares   F-statistic:                     1591.
Date:                Wed, 02 Jun 2021   Prob (F-statistic):          3.42e-278
Time:                        19:49:21   Log-Likelihood:                -2320.3
No. Observations:                 580   AIC:                             4649.
Df Residuals:                     576   BIC:                             4666.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         61.5272      0.725     84.815      0.000      60.102      62.952
x1            24.4242      1.228     19.894      0.000      22.013      26.836
x2            -1.1010      1.211     -0.909      0.364      -3.480       1.278
x3            48.5845      1.470     33.049      0.000      45.697      51.472
==============================================================================
Omnibus:                        4.035   Durbin-Watson:                   2.110
Prob(Omnibus):                  0.133   Jarque-Bera (JB):                4.839
Skew:                           0.031   Prob(JB):                       0.0890
Kurtosis:                       3.443   Cond. No.                         3.31
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```



The previous plot should look like a Gaussian distribution in order to validate the assumption of normal distribution of the residuals.

This is an indicator that LR may not perform the best.

We will also run a *QQ-plot* to validate our model.



The *QQ-plot* looks fine.

We start optimizing the *Ridge Regression* (i.e. MLR with regularization). For this model we are going to look for the best α which is the regularization strength.

```
0 minutes, 1 seconds
{
  "alpha": 1
}
```

	param_alpha	mean_test_mae	mean_test_mse	mean_test_r2
4	1	-10.552082	-177.208270	0.886404
3	0.5	-10.555654	-177.187459	0.886387
5	5	-10.558275	-178.206629	0.885999
2	0.1	-10.559142	-177.189949	0.886360

```
1        0.01     -10.559991    -177.192923      0.886352
```

	MAE	RMSE	R^2
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
Ridge(best)	10.43362	13.456077	0.896253
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best model is found when $\alpha = 1$. The model performs best when the regularization is high i.e. the overfitting is low. Probably, because the data from the validation set is very different to the data from the training set.

For the *Stochastic Gradient Descent Linear Regression* we are going to tune:

- The penalty: L1, L2, and Elasticnet.
- The α as in Ridge Regression.
- The learning rate: constant, optimal, and invariant scaling.

```
0 minutes, 0 seconds
{
  "alpha": 0.5,
  "learning_rate": "constant",
  "penalty": "l1"
}
```

	param_penalty	param_alpha	param_learning_rate	mean_test_mae	\
18	l1	0.5	constant	-10.506823	
24	l1	0.5	invscaling	-10.532848	
0	l1	0.01	constant	-10.534928	
2	elasticnet	0.01	constant	-10.535063	
9	l1	0.1	constant	-10.536924	

	mean_test_mse	mean_test_r2
18	-179.356750	0.885268
24	-178.564109	0.885773
0	-178.088059	0.885824
2	-178.856681	0.885618
9	-178.474832	0.885601

	MAE	RMSE	R^2
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253

```
Ridge(best)       10.43362   13.456077   0.896253
LR-SGD(best)      10.88055   14.037045   0.887101
SVR(default)     12.209157     17.7752   0.818962
NN(default)      19.542292   23.670468   0.678964
GP(default)      26.807484  139.389439 -10.132677
KRR(default)     41.903928   48.805811  -0.364844
```

The best model is found when $\alpha = 0.5$, learning rate is constant, and the penalty is L1. Surprisingly, the hypertuned SGD Linear Regression perform worse than the default one. After spending some time thinking about this and checking that everything was correct I found the answer. The model is hypertuned for the real data i.e. while the default one is not. The default model is probably overfitted and will perform bad on the test data.

3.2.4 k-Nearest Neighbors

For the *k-Nearest Neighbors* we are going to tune:

- The number of neighbors: 2, 5, 10, 20, 30.
- The weights: uniform or distance.
- The Minkowski distance.

```
0 minutes, 0 seconds
{
  "n_neighbors": 5,
  "p": 1,
  "weights": "distance"
}
```

```
   param_n_neighbors param_weights param_p  mean_test_mae  mean_test_mse  \
5                  5      distance       1      -7.304946    -101.120084
9                 10      distance       1      -7.312238    -101.161897
7                  5      distance       2      -7.386537    -101.616961
11                10      distance       2      -7.452998    -101.733656
4                  5       uniform       1      -7.586897    -106.575034

    mean_test_r2
5       0.935764
9       0.935946
7       0.935508
11      0.935537
4       0.932272
```

```
                      MAE        RMSE      $R^2$
KNN(best)         7.214225    9.795668    0.94502
RF(default)        7.42645    9.924838    0.94356
KNN(default)         7.465   10.489528   0.936955
LR(default)      10.426426   13.432264    0.89662
LR-SGD(default)  10.429318   13.442404   0.896463
Ridge(default)    10.43362   13.456077   0.896253
Ridge(best)       10.43362   13.456077   0.896253
```

```
LR-SGD(best)      10.88055   14.037045   0.887101
SVR(default)     12.209157     17.7752   0.818962
NN(default)      19.542292   23.670468   0.678964
GP(default)      26.807484  139.389439 -10.132677
KRR(default)     41.903928   48.805811  -0.364844
```

The best model is found when we use 5 neighbors, the manhattan distance and the distance as a weight. We see in the results that our optimized KNN performs better than the default one. The improvement is small.

3.2.5 Random Forest

For the *Random Forest* we are going to tune several hyper-parameters:

- The number of trees: 50, 100.
- The measure of quality of a split: MAE, MSE.
- The maximum depth of the tree.
- The minimum number of samples to split an internal node.
- The minimum number of samples required to be at a leaf node.
- The number of features to consider when looking for the best split.

```
0 minutes, 24 seconds
{
  "criterion": "mse",
  "max_depth": null,
  "max_features": "auto",
  "min_samples_leaf": 2,
  "min_samples_split": 4,
  "n_estimators": 100
}
```

```
   param_n_estimators param_criterion param_max_depth param_min_samples_leaf  \
27                100             mse            None                      2
3                 100             mse             100                      2
1                 100             mse             100                      2
25                100             mse            None                      2
75                100             mae            None                      2

   param_min_samples_split param_max_features  mean_test_mae  mean_test_mse  \
27                       4               auto      -6.576690     -88.698258
3                        4               auto      -6.576690     -88.698258
1                        2               auto      -6.576690     -88.698258
25                       2               auto      -6.576690     -88.698258
75                       4               auto      -6.600129     -88.530604

    mean_test_r2
27      0.943348
3       0.943348
1       0.943348
25      0.943348
```

```
75      0.943766
```

	MAE	RMSE	R^2
KNN(best)	7.214225	9.795668	0.94502
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
Ridge(best)	10.43362	13.456077	0.896253
LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best Random Forest is found when we use 100 trees per forest, MSE as the measure of quality, and maximum depth of 100 to avoid overfitting. The tuned Random Forst perform slightly better than the default one but probably would outperform the default one with the test data.

3.2.6 Kernel Regression

For the *Kernel (Ridge) Regression* we are going to tune:

- The α as in Ridge Regression.
- The kernel: linear, polynomial, RBF, sigmoid.

```
0 minutes, 1 seconds
{
  "alpha": 0.01,
  "kernel": "rbf"
}
```

	param_alpha	param_kernel	mean_test_mae	mean_test_mse	mean_test_r2
2	0.01	rbf	-7.000097	-86.913575	0.944467
1	0.01	polynomial	-7.279903	-92.560738	0.940868
5	0.1	polynomial	-7.327127	-93.483444	0.940317
6	0.1	rbf	-7.411616	-95.050420	0.939293
9	0.5	polynomial	-7.515655	-97.363386	0.937898

	MAE	RMSE	R^2
KRR(best)	6.749738	9.072946	0.952833
KNN(best)	7.214225	9.795668	0.94502
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
Ridge(best)	10.43362	13.456077	0.896253

LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best Kernel Regression is found using the *radial basis function* as the kernel and a small regularization $\alpha = 0.01$. It is surprising how the KRR went from the last position to the first position just by tunning its hyperparameters.

3.2.7 Gaussian Process

For the *Gaussian Process* there is no much tunning to do. We can only change the kernel used. The best model is found when the kernel functions is the RBF as in the Kernel Regression. The optimized Gaussian Process performs way better than the default one but still scores at the middle of the table.

```
0 minutes, 5 seconds
```

	MAE	RMSE	R^2
KRR(best)	6.749738	9.072946	0.952833
KNN(best)	7.214225	9.795668	0.94502
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
GP(best)	10.426458	13.432451	0.896617
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
Ridge(best)	10.43362	13.456077	0.896253
LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

3.2.8 SVR

Support Vector Regression model has lots of hyper-parameters to tune but we are only going to tune a few of them:

- The kernel: linear, polynomial, rbf, and sigmoid.
- The degree and the γ of the kernels.
- The regularization parameter C.

```
0 minutes, 15 seconds
{
  "C": 100,
  "degree": 2,
  "gamma": "scale",
```

```
  "kernel": "rbf"
}
```

```
    param_kernel param_C param_degree param_gamma  mean_test_mae  \
314          rbf     100            7       scale      -6.920229
298          rbf     100            3       scale      -6.920229
290          rbf     100            2       scale      -6.920229
306          rbf     100            5       scale      -6.920229
318          rbf     100            7        auto      -7.365802

     mean_test_mse  mean_test_r2
314     -92.832597      0.940707
298     -92.832597      0.940707
290     -92.832597      0.940707
306     -92.832597      0.940707
318     -97.966851      0.937292
```

	MAE	RMSE	R^2
SVR(best)	6.449151	8.850123	0.955121
KRR(best)	6.749738	9.072946	0.952833
KNN(best)	7.214225	9.795668	0.94502
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
GP(best)	10.426458	13.432451	0.896617
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(default)	10.43362	13.456077	0.896253
Ridge(best)	10.43362	13.456077	0.896253
LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best model is found when we use the RBF as the kernel with a scaling γ and $C = 100$. It is a bit concerning that the regularization parameter is this high since C is inversely proportional to the regularization which is an indicator that the model is overfitting.

3.2.9 Neural Networks

Neural Networks are hard to tune. It requires lot of knowledge and experience to tune them properly. Since our model is so simple the Neural Network doesn't have to be complex, otherwise, it will overfit for sure. From my experience, the Neural Network will overfit anyway but we should give it a try.

We will try the following topologies:

- One hidden layer with 2, 4, 6, or 8 neurons.
- Two hidden layers with 2, 4, 6, or 8 neurons each.

- Three hidden layers with 2, 4, 6, or 8 neurons each.

We will train the Neural Network using the LBFGS solver and ReLU as the activation function.

```
0 minutes, 14 seconds
{
  "alpha": 0.01,
  "hidden_layer_sizes": [
    8,
    8,
    8
  ]
}
```

```
   param_hidden_layer_sizes param_alpha  mean_test_mae  mean_test_mse  \
47                [8, 8, 8]        0.01      -6.844725     -87.899452
55                   [8, 8]         0.1      -6.882325     -89.138209
67                   [8, 8]         0.5      -6.884251     -88.484174
35                [8, 8, 8]       0.001      -6.891949     -87.254180
71                [8, 8, 8]         0.5      -6.899091     -88.478962

    mean_test_r2
47      0.943695
55      0.943247
67      0.943535
35      0.944317
71      0.943576
```

	MAE	RMSE	R^2
SVR(best)	6.449151	8.850123	0.955121
KRR(best)	6.749738	9.072946	0.952833
KNN(best)	7.214225	9.795668	0.94502
NN(best)	7.264912	9.19834	0.95152
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
GP(best)	10.426458	13.432451	0.896617
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(best)	10.43362	13.456077	0.896253
Ridge(default)	10.43362	13.456077	0.896253
LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The best Neural Network has 3 hidden layers with 8 neurons each one. The best model has picked the most complex topology which is an indicator that the model is overfitting.

3.3 Results

	MAE	RMSE	R^2
SVR(best)	6.449151	8.850123	0.955121
KRR(best)	6.749738	9.072946	0.952833
KNN(best)	7.214225	9.795668	0.94502
NN(best)	7.264912	9.19834	0.95152
RF(best)	7.4194	9.750344	0.945527
RF(default)	7.42645	9.924838	0.94356
KNN(default)	7.465	10.489528	0.936955
LR(default)	10.426426	13.432264	0.89662
GP(best)	10.426458	13.432451	0.896617
LR-SGD(default)	10.429318	13.442404	0.896463
Ridge(best)	10.43362	13.456077	0.896253
Ridge(default)	10.43362	13.456077	0.896253
LR-SGD(best)	10.88055	14.037045	0.887101
SVR(default)	12.209157	17.7752	0.818962
NN(default)	19.542292	23.670468	0.678964
GP(default)	26.807484	139.389439	-10.132677
KRR(default)	41.903928	48.805811	-0.364844

The models that achieve the best MAE, $RMSE$ and R^2 scores are:

- Support Vector Regression(SVR)
- Kernel Regression(KRR)
- k-Nearest Neighbor(KNN)
- Neural Networks(NN)
- Random Forest(RF)

All those models did a great job, probably because the problem is too simple. Picking one among them is not easy but we need to be sure that our model is not overfiting since we have very little samples.

Since SVR and KRR used a small regularization constant we are going to discard them. We also know that Neural Networks have overfiting problems so we will also discard them.

We will finally select k-Nearest Neighbors as the model to calibrate our sensors.

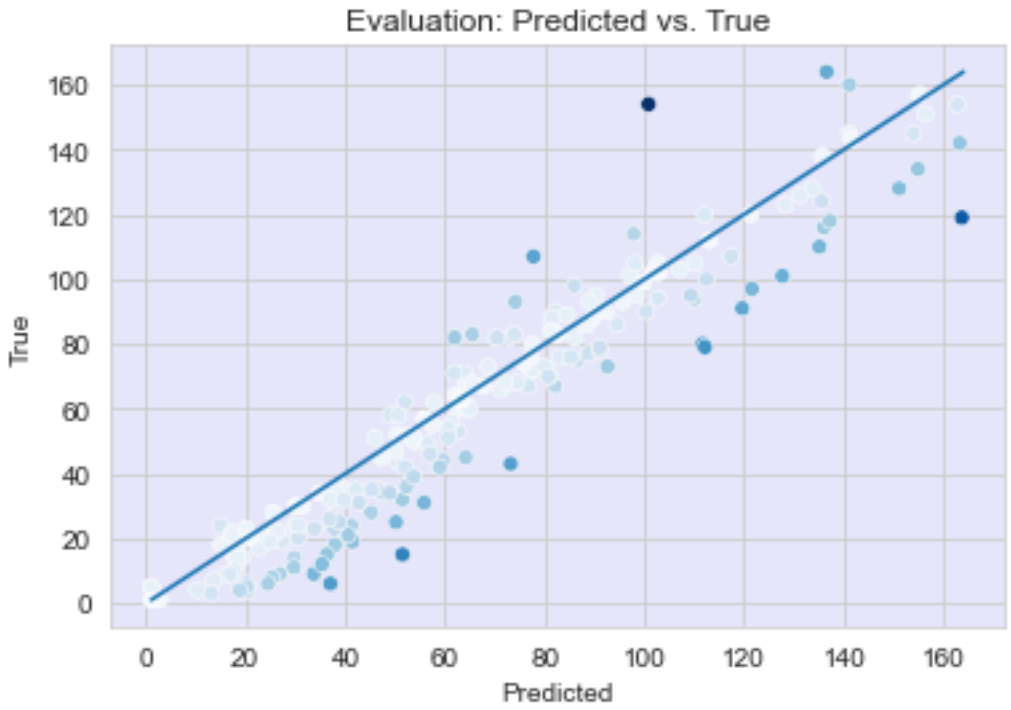
3.4 Model Evaluation

The only task left in our calibration is testing our model. For this task, we are going to use the test dataset which has never been seen by our model (we made sure to avoid data leakage).

The $MAE = 10.19$ shows that our model is not accurate, but it can still make good predictions.

The $RMSE = 13.42$ which shows that we have some outliers.

The $R^2 = 0.89$ which shows that our model fit the data well.



The reader can see that our model predicted very well the true O_3 and we can conclude that our model will work well on the sensors for real data.