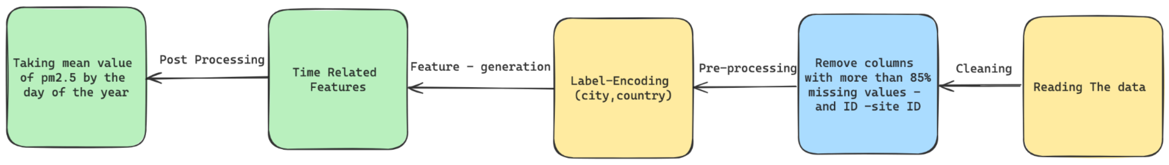
AirQo African Air Quality Prediction

Overview and Objectives:

Thanks for hosting this competition, we had fun optimizing for the given air quality problem and incorporating some useful modeling approaches to our solution. Here we highlight the main chronological steps to develop our solution and relevant thought processes.

Architecture Diagram:


Reading The data
Cleaning
Remove columns with more than 85% missing values - and ID -site ID
Pre-processing
Label-Encoding (city,country)
Feature - generation
Time Related Features
Post Processing
Taking mean value of pm2.5 by the day of the year

ETL Process:

1. About the data: we used the same provided data from the Zindi competition, it is a csv (comma-separated value) format, using the panda's method pd.read_csv, since the data is not quite huge we used pandas for data manipulation.
2. Cleaning and processing: we started by dropping the features with more than 85% of missing values and then headed to do label encoding for the (city and country) features. Then we applied time-related features and dropped (id, site_id, date).

3. Post Processing: We thought postprocessing was crucial because we noticed that the std of the target group by the day of the year was low, especially in the first three quarters of the year, and significantly higher level of variations in the fourth quarter of the year, which resonated with the fact that all factories tend to use their co2 budget at the end of the year, therefore, higher levels of pollutants/pm2_5. and we did it as follows: we took the model prediction and calculated the mean value of it grouped by the day of the year.

Modeling and Validation:

We go for Tree-Based models and we select the LightGBM Algorithm as our winning model. Friendly for **computation efficiently** so that we can try as many empirical experiments as fast as we can, after hitting the problem of collinearity (multiple features have more than 90% correlation which adds redundancy to the data and makes it difficult for our model to extract the underlying pattern, but the tree-based algorithms like LightGBM are not affected by the collinearity, so we decided to settle on it. We didn't use any normalization techniques since tree-based models are not affected by normalizations.

validation: Since the cities in the *test set* are not the same as the *train set*, we decided that the best technique to validate our model is using Group-K Fold which ensures the same group does not appear in both the *testing* and *training* Folds. We grouped the data by the cities.

Inference:

The model deployed inference is derived mainly from the time-related features insights and the variation of the *pm2_5* levels across the days of the years, we only use the predicted functionality and the post-processing technique to test new instances, we can save the model as a pickle or h5 format, and continue training it to new instances while making sure to save the old version using any version control system.

Performance metrics:

1. **Data Processing Time:**
    - **Total ETL Time:** The total time taken to complete the ETL process.
        - **Metric Example:** Total ETL Time = 13.687 seconds
2. **Data Quality:**
    - **Features Dropped:** The number of features dropped due to high missing values.
        - **Metric Example:** Features Dropped = Features with > 85% missing values

Model Training Metrics

1. **Model Training Time:**
    - **Total Training Time:** The total time taken to train the model.
        - **Metric Example:** Model Training Time = 2.93 seconds ± 457 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
        - **Evaluation Metrics:** RMSE and Validation STD
        - **Public - Private Scores: Public** 12.66536524, Private 16.90732355

Run Time:

- Run Time for the Notebook = 13.68726921081543 second
- Run TIme for the model = 2.93 s ± 457 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Error Handling:

Our code and ETL process are free of errors and we did not use any error handling or exceptions

Minatance and Monitoring:

We use metrics to monitor our model, these metrics include

1. **The validation run time:** if our model training process tends to be longer, then means it faces difficulties catching the underlying structure (i.e our data is complex, or hyperparameters are not tuned)
2. **The Standard Deviation of our validation RMSE Score:** after each iteration in the validation folds we save the scores in a list and monitor their standard deviation levels, to detect our model's Stability and Variability as an example of the std validation score is high This could be a sign that the model is overfitting or underfitting certain parts of the data, or that the data itself is heterogeneous.

Strategies for scaling and managing the life cycle:

Before taking any action about what type of scaling and how to scale the solution within the systems, we suggest having a more in-depth analysis of the data's nature as it seems it is hard to understand its behavior with different methodologies and techniques and more empirical experiments can be done to validate the best solution. We can use a formula to create new insight from our sentinel 5p that can be used to boost the modeling performance, for monitoring purposes we think of adding operational metrics to monitor the model performance, like the data latency bandwidth, and memory usage metrics. and how it intersects with our model performance, making sure to avoid concept drift and data drift with our monitoring dashboard.