

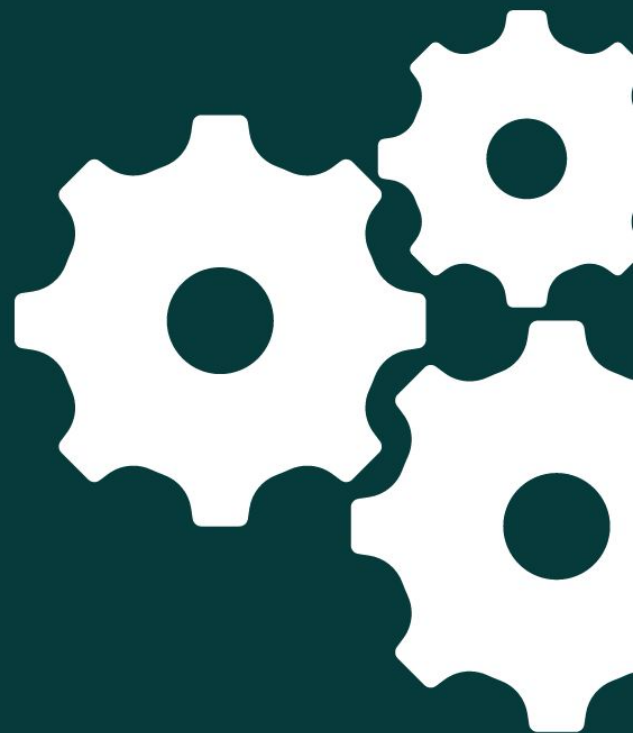
MACHINE LEARNING OPERATIONS

2022/2023

https://github.com/daila10/MLOps_project

DAILA ALEXANDRE 20191182
DIOGO SILVA 20221393
LUÍS FERNANDRES 20221649

PROFESSOR: NUNO ROSA



Index

1. Introduction	3
2. Project Planning.....	3
3. Data Selection and Exploration	3
4. Objective and Success metrics	4
5. Data Engineering and Modelling.....	4
5.1. Data Engineering	4
5.2. Data Science	5
5.3. Model Evaluation	5
6. Quality Assurance and Testing	5
6.1. Data Unit Test	5
6.2. Data Drift	6
6.3. pytest	7
7. Production Implementation and Technology Considerations	7
8. Conclusion	8
 Figure 1 – Metrics Performance Plot	5
Figure 1 - Data- Drift Report	6

1. Introduction

The aim of this project was to develop a machine learning model for predicting machine failure and deploying it into production environment.

The project was divided into three main phases. In the first phase, data collection and exploration were conducted to gain insights into the underlying patterns and characteristics of the dataset. This phase helped to identify important features and understand the data's potential for predictive modeling. The second phase focused on data preparation, model building, and result evaluation. In the final phase, a strong emphasis was placed on ensuring data quality and detecting data drift. Unit data tests and pytests were designed and implemented to verify the integrity of the data throughout the project. Additionally, the evaluation of data drift aimed to detect any significant changes in the data distribution over time, which could impact the model's performance in a real-world scenario.

To facilitate the development and management of the project, Kedro, a robust data pipeline framework, was used. Kedro provided a structured and scalable infrastructure, enabling seamless integration of various project components and ensuring reproducibility. Furthermore, the integration of MLflow within Kedro allowed for efficient tracking and management of multiple model iterations and combinations, simplifying the comparison and evaluation process.

2. Project Planning

The project was divided in three main phases: Data Selection and Exploration, Data Engineering and Modelling and finally, Quality Assurance and Testing. The planning was done as follows:

- Sprint 1: One day - Start and finish the first main phase Data Selection and Exploration.
- Sprint 2: One day - Build essential pipelines to build a Machine Learning model (data_engineering, data_science and model_evaluation).
- Sprint 3: Two days - Debug and improvement of pipeline built in Sprint 2.
- Sprint 4: Three days - Build pipeline tests to assure data, functions and pipelines quality (data_unit_tests, data_drift and elaborating all the pytest functionalities).
- Sprint 5: One day - Improve model performance.

3. Data Selection and Exploration

A search was conducted for open-source datasets that tackled compelling problems. The [Machine Predictive Maintenance Classification Dataset](#) from Kaggle was chosen due to its initial simplicity, serving as a foundational step towards more complex scenarios.

A notebook *Exploration* was created with the purpose of getting insights in order to build functions and pipelines suitable for the data.

The dataset has ten features including an index (*UDI*) and two target features *Target* that is a binary feature indicating the occurrence of failure and *Failure Type* specifying the type of failure among five existents. It also has 10000 observations without any missing values or duplicates.

A summary statistics table was created and further insights from it can be found in the notebook.

A correlation matrix was also created for the independent numeric features using the Pearson coefficient. It can be observed that there is a high correlation between *Air Temperature* and *Process Temperature* (0.88), this was expected as process temperature is always related with the air temperature, however in the occurrence of a failure, the equipment will get even hotter, therefore these features are not going to be dropped, instead a new feature using a combination of both will be created later in the project. The features *Torque [Nm]* and *Rational speed [rpm]* also present a high negative correlation, nonetheless, both features are important for the study as the rotational speed might represent a situation where the machine is working in a high demand or power (not necessarily faulty), and torque might represent the effort of the machine. It is important to note that there is a

variable representing the quality of the machine (L – low, M – Medium, H – High). The ‘Low quality’ type is the predominant with 60% of the sample.

Assessing the dependent variables, these show a sizable unbalance, for both features, the class representing machine failure (0 in *Target* and ‘No Failure’ in *Failure Type*) represents 96.61% of the dataset. This information is especially important in order to choose a classifier and a performance metric that takes the imbalance problem into account.

Among the numeric features, only *Torque [Nm]* and *Rational speed [rpm]* showed possible outliers. The ones in the *Torque [Nm]* boxplot are not far from the remaining observations unlike *Rational speed [rpm]*, however, *Rational speed [rpm]* only has upper outliers, leading to the idea that they may correspond to machines that failed and therefore, later in the process, they will not be removed.

4. Objective and Success metrics

The primary objective of the project was to develop a machine learning model capable of accurately predicting machine failures. The target variable was defined as the binary outcome of whether a machine would fail or not (*Target* feature).

Due to the unbalanced nature of our target variable, the defined success metrics of the project include F1-score, and AUC to evaluate the performance of the machine learning model in predicting machine failures. These scores will be used to evaluate the ability of the model to identify different if the machine failed. Ensuring consistency in data quality checks (through pytest) and achieving a high pass rate in unit tests are also crucial indicators of data reliability. Finally, a successful implementation of the pipeline that is reproducible in all environments is a key measure of the project's success.

5. Data Engineering and Modelling

Moving on to the second phase, this was divided into three subparts that are also three separated pipelines: Data Engineering, Data Science and Model Evaluation.

5.1. Data Engineering

The data engineering pipeline for this project consists of three main steps – also designated nodes: `clean_data`, `data_engineered`, and `data_split`.

During the data cleaning phase, the raw data is processed to remove unnecessary columns for modelling (‘UDI’, ‘Product ID’ and ‘Failure Type’) and rows with missing values. Additionally, the temperature columns are converted from Kelvin to Celsius as it is the metric unit used in our country, and any negative torque values are discarded. This cleaning process ensures that the dataset is consistent, complete, and suitable for further analysis and modeling.

Following data cleaning, feature engineering is performed to derive new meaningful features from the cleaned dataset. Binary variables are created to represent different product quality variants (low, medium, and high) as it was already mentioned in the exploration. Due to its high correlation, the difference between air temperature and process temperature is calculated to capture the temperature delta as a relevant feature. The torque values are normalized by dividing them by the maximum torque, facilitating fair comparison across different samples. Moreover, considering the distribution values identified during the exploration, categorical variables are created to classify rotational speed into low, medium, and high ranges. These engineered features provide valuable insights and enhance the model's ability to capture patterns and relationships within the data.

Once the feature engineering step is completed, the data is split into training (80% of the data) and test (20% of the data) sets.

5.2. Data Science

The data_science pipeline is mainly dedicated to train a classifier to conduct the task of predicting machine failure. It starts by ingesting the preprocessed train data in the train_model node where it trains a selected machine learning model, in this case Random Forest Classifier. The model's artifacts are saved in the catalog and it is passed to the next node where it makes predictions for the test data. Those predictions in turn are saved in the data catalog and feed to the next step.

5.3. Model Evaluation

The model evaluation was made using several metrics and always having in consideration that the data is unbalanced. The pipeline model_evaluation is only composed by one node evaluate_model that plots and saves the ROC curve and the F1-Score. The Random Forest Classifier was tested several times with different parameters in an attempt to improve performance, all these attempts can be assessed in mlflow experiments where the pickle file of the model and the plots can be found. Furthermore, to explore and learn different technologies, the metrics were also kept track in Kedro experiments.

The plot of the best model achieved can be observed in the following image.

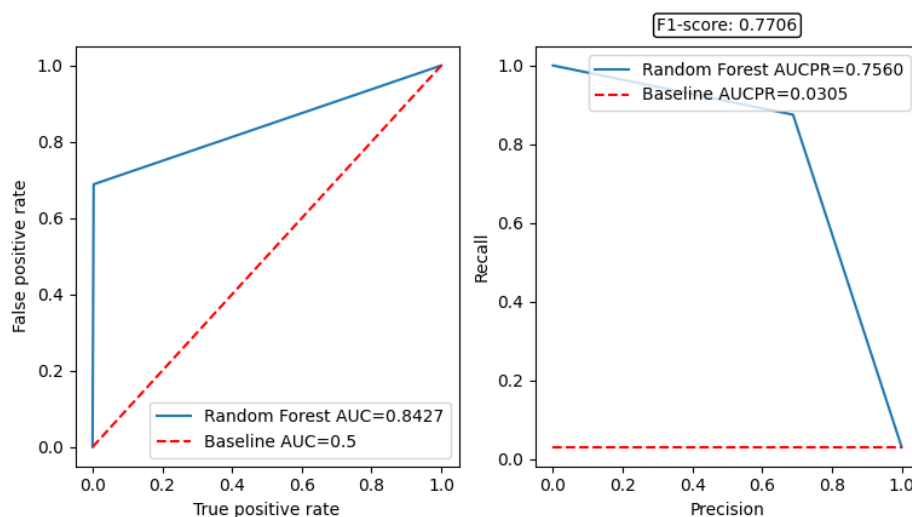


Figure 1 – Metrics Performance Plot

The classifier achieved an F1-Score of 0.77, indicating a reasonably balanced performance in correctly classifying positive and negative instances. The AUC of 0.84 means a strong ability to distinguish between positive and negative instances, while the AUCPR of 0.75 demonstrates a good trade-off between precision and recall. Overall, the model performs well in the given task, exhibiting a balanced classification performance, effective discrimination between classes, and reasonable precision-recall balance.

6. Quality Assurance and Testing

The last phase of the project was dedicated to ensuring the quality of every component of the project.

6.1. Data Unit Test

To perform data unit tests, a pipeline was created using Great Expectations. It performs validation checks to ensure that the raw data meets the expected types for specific columns.

Each validation check returns a success status, and the node function asserts that all checks are successful. If any of the checks fail, an AssertionError is raised.

The purpose of these unit tests is to validate the data integrity and consistency based on the expected column types. It provides an automated way to ensure that the data conforms to the defined schema, helping to identify any potential data quality issues early in the pipeline.

6.2. Data Drift

The `data_drift` pipeline focuses on detecting data drift between a reference dataset and an analysis dataset. The pipeline applies various statistical tests and generates a data drift report using the Evidently AI library.

In this case, the reference dataset is the one used in the project after all the cleaning and all the engineering pipelines. As at the moment of the elaboration of the project there is no other dataset with new data for a new temporal period, a new dataset was fabricated for testing the functionality of the data drift pipeline only. This new dataset for testing purposes was created from the original dataset, filtering only for *Rotational Speed [rpm]* over 2000 – which can be a real industry scenario, where in a high demand period, the machines are also working always in a higher demand.

A "ConstantThreshold" class from the NannyML library was then set to a range between 0.3 and 0.7 and is applied to the Jensen-Shannon statistical test. Many thresholds were tested, verifying that as low the threshold is, as quickly and accurate is to identify drift.

Next, the "UnivariateDriftCalculator" is initialized to perform univariate data drift calculations on selected columns. The columns used for the drift analysis are *Air temperature [C]*, *Process temperature [C]*, *Rotational speed [rpm]*, *Torque [Nm]*, and *Tool wear [min]*. Only those columns were selected as they are the only ones that have the potential to change their value over time. The previously defined threshold is used for the Jensen-Shannon test.

The univariate drift calculations are performed by fitting the calculator with the reference data and then calculating the drift using the analysis data. To generate a data drift report, the function creates an instance of the "Report" class from the Evidently library. The report applies the Kolmogorov-Smirnov test to numeric features with a significance threshold of 0.05.

With the fabricated dataset, the drift report indicates that drift is detected in 80% of the selected columns. This conclusion demonstrates the effectiveness of the `data_drift` pipeline in identifying changes in data distribution.

The variables with drift scores of 3.89, 2.35, 0.09, and 0.069 are the ones where drift was detected. This is because their calculated drift scores exceeded the specified threshold.

Drift is detected for 80.0% of columns (4 out of 5).


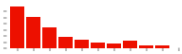






Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
> Rotational speed [rpm]	num			Detected	Wasserstein distance (normed)	3.89088
> Torque [Nm]	num			Detected	Wasserstein distance (normed)	2.354441
> Air temperature [C]	num			Detected	Wasserstein distance (normed)	0.091179
> Process temperature [C]	num			Detected	Wasserstein distance (normed)	0.069967
> Tool wear [min]	num			Not Detected	Wasserstein distance (normed)	0.047516

Figure 2 - Data- Drift Report

Furthermore, it is possible to observe in the annexes 1 and 2, a difference between the distribution of a variable where drift was detected (Process temperature) and another one where drift was not detected (Tool wear). The variable Process Temperature shows a significant shift or deviation in the distribution over time, where the variable Tool Wear remains relatively stable over time. Other variables and graphs can be analyzed accessing the drift report on the project folder.

6.3. pytest

In the test module, several pytest have been implemented to validate different aspects of the data engineering pipeline in the project.

- The **test_clean_data_type()** function verifies that the output of the `clean_data` function is a pandas DataFrame. This test ensures that the data cleaning process successfully transforms the input DataFrame into a cleaned DataFrame.
- The **test_clean_data_column_types()** ensures that the column types in the preprocessed DataFrame (engineered data) match the expected types after cleaning.
- The **test_clean_data_null()** function ensures that there are no null values in the DataFrame after the cleaning process.
- The **test_feature_engineering_torque()** function validates that the *Torque [Nm]* values in the final DataFrame are always positive.
- The **test_clean_data_columns()** ensures that the transformation process retains the expected number of columns in the output DataFrame.
- The **test_binary_variables_values()** function specifically targets the binary variables in the feature-engineered DataFrame. It checks that the unique values in the binary columns are limited to 0 and 1 only. This confirms that the binary variables are correctly encoded and contain the expected values.

Running these tests as part of the testing pipeline helps in maintaining the quality and reliability of the project.

7. Production Implementation and Technology Considerations

The production implementation of the machine learning model for predicting machine failure involved the use of various technologies to ensure a robust and scalable solution. Key packages used in the project included Kedro (v.0.18.9), mlflow (v.2.3.2.), scikit-learn (v.1.2.2.), pytest (v.7.4.0), great-expectations (v.0.16.16), NannyML (v.0.8.6) and evidently (v.0.3.3).

Kedro, as a data pipeline framework, played a crucial role in structuring the project and integrating different components seamlessly. It provided a structured configuration for data engineering, model development, and testing, ensuring reproducibility and maintainability. The integration of mlflow with Kedro facilitated efficient tracking and management of multiple model iterations, simplifying the comparison and evaluation process.

Pytest provides a robust testing framework with a simple syntax and plugin support, enabling comprehensive and efficient testing. Great Expectations offers automated data validation and testing capabilities, ensuring data integrity and consistency. NannyML and Evidently are dedicated to data drift analysis, providing statistical tests and visualizations to detect and analyze changes in data distribution. However, configuring Great Expectations for complex pipelines may require some initial effort, and NannyML may need fine-tuning of parameters.

For further improvement of the project, the prediction of the multiclass target *Failure Type* could be incorporated, along with the inclusion of tracking using Kedro Viz for enhanced visualization and monitoring capabilities.

Implementing the proof of concept in a production environment involves planning for infrastructure, data pipeline, monitoring, scalability, security, and continuous integration. It includes

deploying the machine learning model, managing data processing with a pipeline framework, monitoring performance, optimizing scalability, ensuring data security, and establishing a CI/CD pipeline for updates.

In the case of scaling the pipeline for larger datasets poses risks and challenges. It includes longer processing times, increased resource requirements, memory constraints, complex feature engineering and training, data consistency and quality maintenance, infrastructure scalability and reliability, and monitoring and debugging difficulties.

To solve these possible problems several strategies can be implemented. Besides transitioning to Apache Spark, these include optimizing data pre-processing, simplifying feature engineering, selecting scalable models with tuned hyperparameters, leveraging distributed computing frameworks (e.g., Dask, TensorFlow), using cloud infrastructure for scalability, implementing data parallelism and distributed training, and exploring other scalable machine learning technologies.

8. Conclusion

In conclusion, this project demonstrates the potential of machine learning in addressing real industry problems, particularly in the field of predictive maintenance. By utilizing the power of Kedro and MLflow, it was developed a robust and scalable solution for predicting machine failures.

The insights gained from the data exploration phase highlighted the importance of some variables and identified potential areas of improvement.

Working with pipelines has proven to be a valuable aspect in this project. It guarantees an efficient data processing, reproducibility and version control, selective pipeline execution, automatic file saving, facilitate collaboration and ensure reliability and effectiveness of the developed solution.

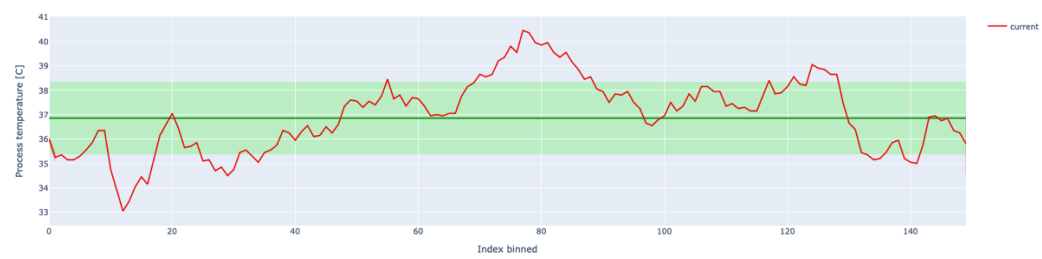
The ability to detect data drift and maintain data consistency adds significant value to the project, ensuring the reliability of predictions over time. The implementation of Pytests and data unit tests also played a crucial role in validating the pipeline and ensuring data integrity. Overall, the model achieved good results, with an F1-score of 0.77, demonstrating its effectiveness in classifying machine failures.

The main challenges were coordinating the project among group members using different operating systems and configurations posed challenges in terms of compatibility, package versions, and file paths.

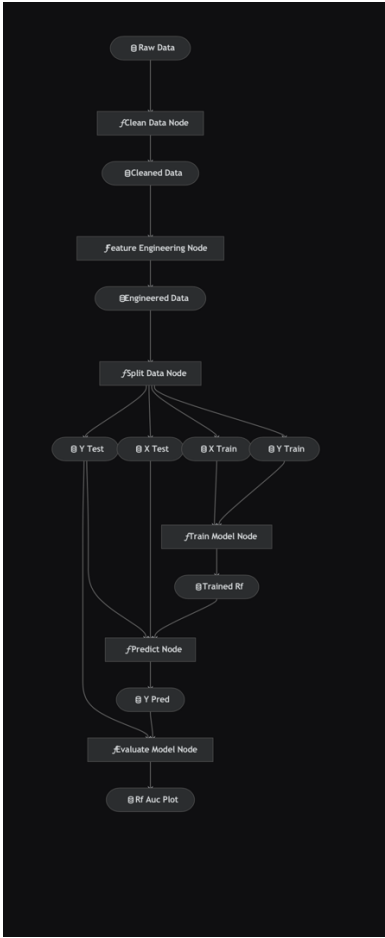
9. Annexes



Annex 1



Annex 2



Annex 3