Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Computer Science (Classe LM-18)

Real Time IoT Analysis with Kestra

Students
De Vitis Fabio Michele
Draibine Adnane
Teghipco Stanislav

Supervisors
De Donato Massimo Callisto
Piangerelli Marco

A.A. 2023/2024

Contents

List of Figures

List of Tables

1. Intoduction

Since its establishment in 1987, Tormatic Srl has specialized in the production of turned parts based on customer designs, marking its prominence in the precision manufacturing industry. The company's machine park, which has seen significant growth over the years, stands as one of the most technologically advanced in Italy. This advancement enables Tormatic Srl to work with a wide range of materials, including every type of steel, stainless steel, brass, aluminum, and plastics. With the capability to produce a vast array of components, even those of high technical difficulty, Tormatic Srl has positioned itself as a leading company in Italy.

Integrating predictive maintenance into Tormatic Srl's operations taps directly into its core strengths—advanced technological capabilities and a commitment to quality and efficiency. By analyzing machine energy consumption and the reasons behind machine stoppages, Tormatic Srl can further enhance its operational efficiency, ensuring that its sophisticated machinery is maintained at peak performance levels. This not only reduces energy consumption and waste but also aligns with the company's ethos of delivering high-quality, precisely engineered components to its clients.

1.1 Objectives

In the journey toward implementing a predictive maintenance strategy, we began with an extensive exploratory data analysis (EDA) phase, using the pandas library in Python. This initial phase was crucial for understanding machine energy consumption and stoppage patterns. Through careful feature engineering, we improved the dataset's quality and relevance to uncover meaningful insights and correlations. We used techniques like t-tests to rigorously examine relationships within the data and ensure our findings were statistically significant.

Feature engineering played a vital role in isolating key factors influencing machine performance and maintenance needs. This step is crucial for predictive modeling as it directly affects the model's ability to learn from the data and make accurate predictions.

Building on this foundation, our main goal is to develop a predictive maintenance regression model using PySpark. We chose PySpark because it allows us to handle big data efficiently, leveraging its distributed computing capabilities to manage and analyze large volumes of data at scale. This approach aims to predict maintenance needs accurately, minimizing downtime and optimizing machine performance across our technological infrastructure. This forward-looking strategy highlights our commitment to innovation and excellence in predictive maintenance, ensuring we remain leaders in Italy's precision manufacturing industry.

1.2 Project Structure

The project is hosted on GitHub[1] and in comprises different sections, from data analysis to Prototypes of the final tool.

The following is the representation of the structure that comprises the most important folders:

```
KestraDataOrchestrator/
├── Analysis_and_Exploration/
│   ├── T_Test.ipynb
│   ├── quality_production_analysis.ipynb
│   └── ⋮
├── Data-Orchestration/
├── Prototype/
└── ⋮
```

The *Analysis_and_Exploration/* folder contains all the files that have been used for feature engineering and for analysis. Inside *Data-orchestration* we will insert the files pertinent to the part of project that will handle real time and batch analysis and prediction of data, while in the *Prototype* folder we will insert prototypes developed along the way.

The other folders contained are self explanatory, to view and inspect them please visit the the link at the footnote that leads to the project.

[1] GitHub project: https://github.com/Staffilon/KestraDataOrchestrator/

2. Dataset Description

In the following sections we will describe how our initial dataset was composed and what kind of feature engineering we did in order to solve some problems we noticed in the dataset.

2.1 File Structure

The files of the project are organized in different folders, depending on which kind of data they contain. The following paragraphs are going to show how data about energy, stoppage registration and produced pieces is structured.

2.1.1 Energy Consumption Tracking

At the heart of our monitoring system are the *energy* and *energy_bak* folders. Here, every piece of specialized machinery is equipped with an energy meter recording the energy consumed at **15-minute intervals**. These readings are encapsulated in files named in the format *location_Tormatic-channel_XXX-register_Ea_Imp_******.csv*, where each file provides a granular view of energy usage, pivotal for analyzing operational efficiency and identifying potential areas for energy conservation.

2.1.2 Production Details and Machinery Identification

Diving deeper into the *energy_bak* folder, we encounter files named *Tormatic_****.csv*, crucial for understanding the production rate of each machinery. These files contain the machinery ID, which serves as a key to cross-reference with its respective energy meter data. Additionally, they offer insights into the type of items produced and the number of pieces manufactured within specific intervals, enabling a comprehensive analysis of productivity and operational dynamics.

2.1.3 Machine Downtime Documentation

In the field of manufacturing, understanding downtime is as critical as monitoring active production. The *Fermate* folder is dedicated to documenting instances when machinery halts—be it for routine maintenance, voluntary stops, or other reasons. Inside it, we have a subfolder named *fermi*, which contains the files in the with the name format *Fermate YYMM.xls*, where *YY* represents the last two digits of the year in which the stoppages were recorded, and *MM* stands for the corresponding month of that year.

Furthermore, the *Fermate* folder contains two document files: *Le macchine sono tutte uguali.docx* which contains some general information about the number and names of stoppages, alongside some information about the machines, and *Significati*

dei fermi.docx, which delves deeper into the meaning of the stop codes of the stoppages.

2.1.4 Structure Schema

The following schema shows a graphical representation of the dataset structure provided by the company:

```
energy/
├── location_Tormatic-channel_XXX-register_Ea_Imp_******.csv
├── ⋮
energy_bak/
├── location_Tormatic-channel_XXX-register_Ea_Imp_******.csv
├── ⋮
├── Tormatic_****.csv
├── ⋮
fermi/
├── Fermate/
│   ├── Fermate YYMM.xls
│   ├── ⋮
│   ├── Le macchine sono tutte uguali.docx
│   └── Significati dei fermi.docx
```

2.2 Initial Dataset Structure

Here we are going to detailledly describe the structure of our dataset.

2.2.1 Energy Bak Dataframe

ID	TimeStamp	Ea_Imp (kWh)
0	2021-04-01T00:00:00Z	445.3
1	2021-04-01T00:15:00Z	437.9
2	2021-04-01T00:30:00Z	450.6
3	2021-04-01T00:45:00Z	457.6
4	2021-04-01T01:00:00Z	438.9
5	2021-04-01T01:15:00Z	442.6
6	2021-04-01T01:30:00Z	444.2

Table 2.1: Recorded Energy Consumption at 15-minute Intervals

Inside the energy_bak folder there are multiple csv files for each machine with the structure described at 2.1.

As we can see from the previously table structure there are three columns in our csv files. Columns are composed by the following elements:

1. **ID**: This is a unique identifier that identifies each unique energy consuption in a certain timestamp.
2. **TimeStamp**: This is the time at which the energy consuption is recorded, theres a 15 minutes interval between each timestamp, so the energy is observed every 15 mins.
3. **Ea_Imp**: This is the value of the energy consuption recorded.

2.2.2 Fermate Dataframe

These are the columns present in the fermate dataframe and their meanings:

1. **SHIFT_DATE**: Date of the shift, i.e., the date on which the stoppage occurred.
2. **SHIFT_CODE**: Identifier of the shift.
3. **SHIFT_START**: Start time of the shift.
4. **SHIFT_END**: End time of the shift.
5. **START_DATE**: Date and time the stoppage began.
6. **END_DATE**: Date and time the stoppage ended.
7. **RESOURCE**: Identifier of the machine involved in the stoppage.
8. **PRODUCTION_ORDER**: Number of the production order in progress at the time of the stoppage.
9. **STOP_CODE**: Code that identifies the reason for the stoppage.
10. **T_STOP**: Duration of the stoppage, expressed in units of time.
11. **QTY_GOOD**: Quantity of finished product considered good.
12. **QTY_SCRAP**: Quantity of waste produced during the shift.

We need to pay a particular attention to the **STOP CODE** column, since its the reason why the machine was stopped. We found out by analyzing the dataset that there are several different reasons. We are going to enumerate them:

1. **STOP_CODE: 0** Indeterminate
2. **STOP_CODE: 1** Startup
3. **STOP_CODE: 2** Setup
4. **STOP_CODE: 4** Routine Maintenance
5. **STOP_CODE: 6** Heating
6. **STOP_CODE: 7** Raw Material Absent
7. **STOP_CODE: 8** Tool Sharpening
8. **STOP_CODE: 9** Extraordinary Maintenance

9. **STOP_CODE: 10** Tool Replacement
10. **STOP_CODE: 11** Bar Loading
11. **STOP_CODE: 12** Correction of Out-of-Tolerance Dimensions
12. **STOP_CODE: 13** Management System Failure
13. **STOP_CODE: 14** Operator's Oversight
14. **STOP_CODE: 15** Ordinary (Chip/Bar Loading)

2.2.3 Tormatic Dataframe

The Tormatic dataframe contains data about the produced pieces by designated machines in a specified time frame. The following are the columns that are present inside each file:

1. **ID**: The numerical ID that is unique to each entry.
2. **COD_MACC**: Identifies the machine that produced the pieces.
3. **COD_ART**: The code of the designated item to be produced.
4. **TIMESTAMP_INIZIO**: Identifies the start of the production cycle.
5. **TIMESTAMP_FINE**: Identifies the end of the production cycle.
6. **NUMERO_PEZZI_PROD**: The number of produced pieces in the time frame.
7. **EXPSTATUS**: Couldn't find the meaning of the column, it contains the value 0 in all the entries.
8. **ODP**: Couldn't find the meaning of the column, it contains numerical codes with a predefined length of 10.

The *TIMESTAMP...* and the *NUMERO_PEZZI_PROD* columns are of upmost importance for the analysis of the dataset, because they will allow us to determine if there is some correlation with the produced energy and stoppages.

3. Explorative Data Analysis (EDA)

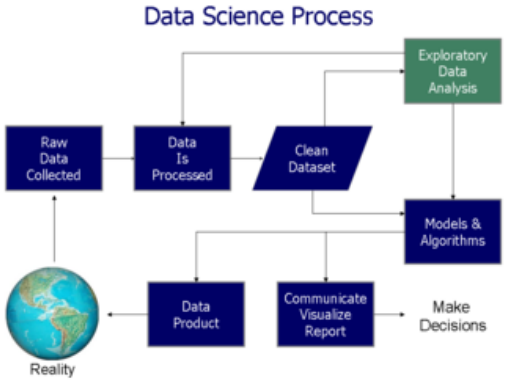

Figure 3.1: Data project life cycle

In the initial phase of our project, we embarked on an exploratory data analysis (EDA) to thoroughly understand the underlying patterns, detect anomalies, test hypotheses, and check assumptions with the help of statistical summaries and graphical representations. EDA is a critical step in the data science project lifecycle as it allows for a deeper understanding of the data's characteristics, which is essential for making informed decisions about suitable models and hypotheses for further analysis. By identifying trends, patterns, and relationships within the data, EDA helps in uncovering insights that guide the subsequent stages of modeling and analysis, ensuring a robust foundation for the project's analytics.

For the execution of our EDA, we selected Google Colab as our notebook environment. Google Colab provides a powerful, cloud-based platform that facilitates easy sharing, access to high-end computational resources, and seamless integration with Google Drive. It's an ideal choice for collaborative projects and exploratory analysis due to its user-friendly interface and the capability to run complex computations without requiring local hardware resources.

Pandas, a cornerstone Python library for data manipulation and analysis, was our primary tool for handling the data. Renowned for its ease of use, efficiency, and flexibility, Pandas significantly simplifies the data analysis process. It offers a wide array of functionalities for data cleaning, transformation, and aggregation, making it possible to prepare the dataset for analysis quickly. Its powerful data structures, such as DataFrames and Series, allow for handling and analyzing large datasets with ease, making Pandas an indispensable tool for data scientists.

In addition to Pandas, we leveraged Matplotlib, a comprehensive library for creating static, interactive, and animated visualizations in Python. Matplotlib enabled us to craft a variety of graphs and charts, which are vital for EDA, as they allow for the visual examination of data distributions, trends, and patterns. Through these visualizations, we were able to communicate complex insights in an intuitive and accessible manner, facilitating a better understanding of the data's characteristics and the relationships between variables.

Furthermore, our analysis incorporated other Python libraries to enhance our exploratory data analysis. Libraries such as NumPy for numerical computing, SciPy for advanced computations, and Seaborn for statistical data visualization complemented our use of Pandas and Matplotlib. These tools collectively provided a comprehensive ecosystem, enabling us to conduct a thorough and effective EDA. The integration of these libraries allowed us to leverage their unique strengths, from NumPy's efficient array operations to Seaborn's high-level interface for drawing attractive and informative statistical graphics.

In summary, the initial stage of our project underscored the importance of EDA in uncovering insights and guiding the analytical process. By utilizing Google Colab, Pandas, Matplotlib, and other Python libraries, we established a robust framework for analyzing and understanding our dataset. This approach not only facilitated a deeper insight into the data but also set the stage for subsequent modeling and analysis, ensuring that our project was grounded in a solid understanding of the underlying data characteristics.

3.1 Feature Engineering

Feature Engineering is a fundamental stage in the data pre-processing phase of any data science project. It was our first step into analyzing data in a detailled way to be able to elaborate a ML model. It consists of critical steps aimed at preparing the dataset for modeling in a way that maximizes the potential for accurate and meaningful predictions.

The main steps of feature engineering are the followings:

1. **Handling Null Values:** This process involves identifying missing values withing the dataset and determining the most appropriate method to deal with them. There are several different strategies that chan vary from imputing missing values based on statistical measures like the mean or using more sophisticated imputation techniques like K-Nearest neighbors or regression imputation, another solution is to eliminate rows or columns with a significant amount of missing data.

2. **Creating and Deleting columns:** Feature Engineering often requires the transformation or creation of new features that better capture the underlying patterns in the data or are more aligned with the predictive modeling goals. This could involve decomposing dates into more useful components, like day or weeks, creating interaction terms to capture the effect of combinations of features. Additionally it may also involve deleting columns that are redundant, have little to not variance or do not contribute to the predictive accuracy.

3. **Feature selection:** This step involves evaluating and selecting the most relevant features to include in the model. Feature selection helps in reducing dimension-

ality, improving the model performance and decreasing training time and model complexity by reducing the amount of features we choose to train the model itself.

4. **Normalization and scaling:** Normalizing or scaling features is essential. This step involves adjusting the scale of the features so they have a consistent range or distribution, which helps in speeding up convergence and improving the model's ability to learn.

FE on the Energy Dataframe

As the first step on our feature engineering phase on the energy dataframe we decided to merge all of the energy files that were in the energy bak folder.

These starting csv files had the following structure:

Table 3.1: Initial Energy dataframe Summary

#	Column	Non-Null Count	Dtype
0	id	567805 non-null	int64
1	TimeStamp	567805 non-null	datetime64[ns, UTC]
2	Ea_Imp	567805 non-null	float64

Since the code of the machine was present in the name of the files we decided that when we were going to merge them we add to add another column, this column named channel was the machine code, so we knew each row to which machine it corresponded. Our final dataframe structure was the following:

Table 3.2: Final energy dataframe Summary

#	Column	Non-Null Count	Dtype
0	id	567805 non-null	int64
1	TimeStamp	567805 non-null	datetime64[ns, UTC]
2	Ea_Imp	567805 non-null	float64
3	channel	567805 non-null	int64

We then checked for missing values or NaN values, removed them and then saved the new dataframe. We didn't have to create any additional column since it would not bring any improvement to our analysis or the future model we were gonna develop since we did not have any type of information that could be created by transforming these three columns.

FE on the Stoppages Dataframe

First of all I combined all the fermate csv files that were present in the fermate directory. After this we saw the main characteristics of the dataset.

We identified the problems in the displayed dataframe and we applied a Feature Engineering phase that consisted on the following steps:

1. **Removed redundancies:** we noticed that the columns SHIFT START,START DATA. SHIFT END AND END DATE were redundant and contained the same informations, we provided to delete the duplicates.

Table 3.3: DataFrame Summary

#	Column	Non-Null Count	Dtype
0	SHIFT_DATE	364231 non-null	datetime64[ns]
1	SHIFT_CODE	364231 non-null	int64
2	SHIFT_START	364231 non-null	datetime64[ns]
3	SHIFT_END	364231 non-null	datetime64[ns]
4	START_DATE	364231 non-null	datetime64[ns]
5	END_DATE	364231 non-null	datetime64[ns]
6	RESOURCE	364231 non-null	int64
7	PRODUCTION_ORDER	364231 non-null	int64
8	STAGE	364231 non-null	int64
9	STOP_CODE	364231 non-null	int64
10	T_STOP	364231 non-null	int64
11	QTY_GOOD	364231 non-null	int64
12	QTY_SCRAP	364231 non-null	int64
13	DESFERM	364231 non-null	object

2. **Standardization:** we standardized the RESOURCE columns that identifies the machine code, since in this dataframe the machine code started with a zero, while on the energy dataframe they started with the first real code number. We made this to standardize the machinery codes.

3. **Removed machines:** we removed machines that were present in this dataset but not in the energy dataset since they were not relevant for our analysis and objectives.

4. **Converted time:** We converted the time columns in the same format as the energy dataframe.

5. **Saved dataframe:** We then assured that the dataframe did not have any missing value or possible issue, then we saved a copy of the new dataframe after all these steps.

[1]

FE on the Tormatic Dataframe

The *Tormatic* dataframe, which contains information about the producecd pieces, proved to be quite challenging.

As soon as we tried to do some basic analysis on the files of the dataset with pandas, we got errors upon errors. The files has shifted columns, meaning that at certain points inside them, some values (namely the numbers 5, 9 and 51/2") would appear inside the third or fourth column, shifting the rest of the values to the column on the right and creating and additional *phantom column*.

After deleting such values, we identified and removed some files that had only 4 columns and had no real use for our analysis.

[1]The notebook used for this feature engineering process can be found here: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/EDA.ipynb

The third step involved removing rows that had 0 produced pieces, since we are interested in the energy levels of machinery while items are being produced, and subsequently we deleted the *EXP_STATUS* and *ODP* columns, since they had no real use.

The fourth step was all about merging together the files into a single one, making the subsequent analysis easier.

In the fifth and final step, deleted entries from the *ID* and *NUMERO_PEZZI_PROD* columns, which in some cases had strings and dates instead of the numerical values.

Creation of Merged Datasets

In order to gain deeper insights from our various datasets, we opted to merge some of them. Initially, we combined the *Energy* and *Stoppages* datasets, resulting in the creation of the file **energy_fermate_merged.csv**[2]. We utilized both *modified_energy_data.csv* and *modified_fermate_data.csv*, taking into account their respective timestamps. Knowing that energy readings are recorded every 15 minutes, we selected the last recorded energy entry within 15 minutes from the commencement of a stoppage. This approach provided an indication of machinery consumption at the time of stoppage occurrence. Although we experimented with different timeframes such as 30 and 60 minutes, the incremental additions to the dataset were minimal, typically comprising only a couple of hundred entries. Consequently, we adhered to our original strategy. The merged dataset's columns encompassed all those present in the stoppages dataset, supplemented by two new columns: **EnergyRetrieval_ID** and **Energy_Value**. The former denotes the ID of the energy value in the Energy dataset, while the latter represents the corresponding energy value retrieved.

Subsequently, we merged the *Energy* and *Produced Pieces* datasets, utilizing *modified_energy_data.csv* and *produced_pieces_dataset.csv*, respectively. This integration yielded the dataset named *pieces_energy_merged.csv*[3]. The merging process mirrored the aforementioned procedure. For each entry in the produced pieces dataset, the start and end timeframes were considered alongside the corresponding energy values within that timeframe, linked to the respective machine. In cases where multiple energy values were identified, the mean value was computed and assigned. If no energy value was found within the specified timeframe, the system searched for an energy value within a 15-minute range from the start timestamp of the produced pieces dataset. If located, it was assigned to the *energy_value* column. In instances where no value was found, the entry for that particular row remained empty. The merged dataset retained all columns from the Produced Pieces dataset, with the addition of a new column: **energy_value**, representing the calculated energy value, whether a mean or a single value.

3.2 Data Analysis

In the following section we are going to showcase our analysis by describing each relevant notebook inside the *Analysis_and_Exploration* folder of our project.

[2]The notebook used for merging energy and stoppages can be found here: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/data_merger.ipynb

[3]The notebook used for merging energy and produced pieces can be accessed here: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/merged_pieces_and_energy_creation_dataset.ipynb

3.2.1 Energy Analysis

In the analysis of energy and stoppages data [4], we performed a comparative study between two distinct scenarios. The first scenario considers the total energy consumption of the machines, while the second scenario accounts for the energy consumption occurring 15 minutes before the cessation of machine operation.

A critical aspect of this study was the comparison of average energy consumption under the two scenarios, illustrated in the Figure 3.2:

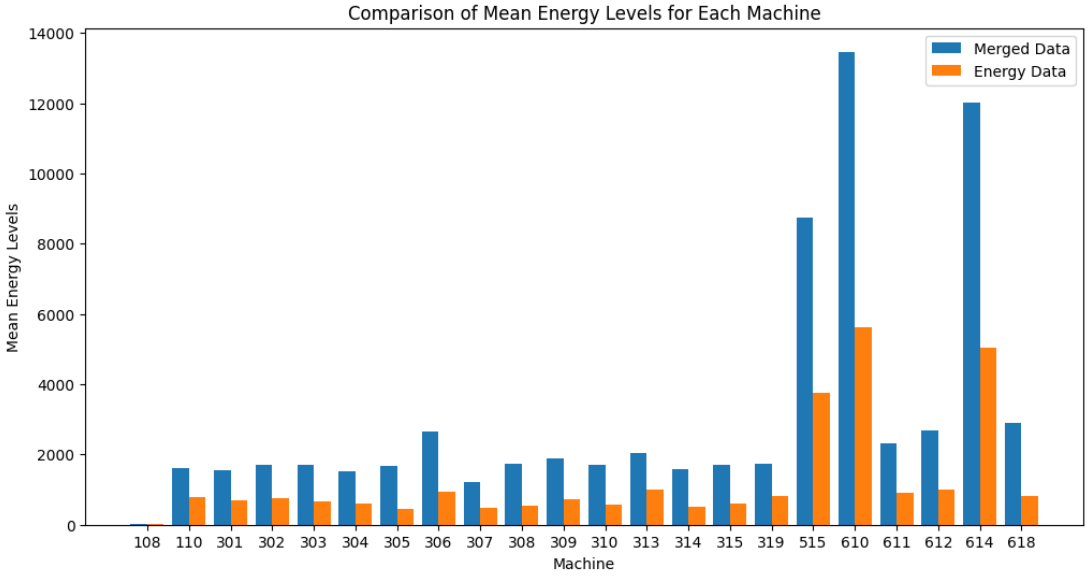

Figure 3.2: Comparison of mean energy consumption levels for each machine.

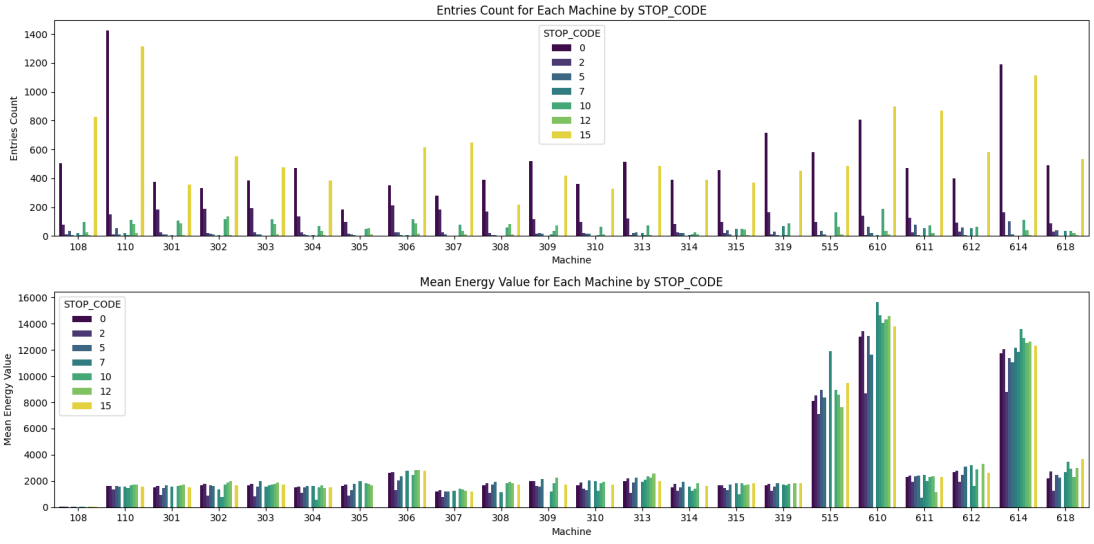

Figure 3.3: Frequency of entries for each stop code.

Subsequent to this comparison, we conducted an analysis focusing on the operational

[4]The notebook user for energy and stoppages analysis: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/collected_data.ipynb

stop codes of the machines. Given the variety of STOP CODES, it was pertinent to examine their frequency across different machines (Figure 3.3) and to analyze the average energy consumption associated with each stop code.

This analysis revealed that STOP CODES **0** and **15** are predominantly more common, with some machines showing a more uniform distribution of entries across all stop codes.

Furthermore, the investigation into energy consumption patterns identified three machines, specifically labeled as **515, 610, and 614**, which exhibited significantly higher energy consumption levels—peaking at 16,000 units—compared to other machines, which only reached up to 2,000 units.

We then addressed the most critical STOP CODES, listed down below, which communicate that there are problems present with the machinery:

- **4**: manutenzione ordinaria
- **8**: affilatura utensile
- **9**: manutenzione straordinaria
- **10**: sostituzione utensile

Machine	Count				Mean Energy			
	4	8	9	10	4	8	9	10
108	37	20	6	95	9.45	7.91	27.27	13.07
110	55	22	8	112	1637.59	1575.75	1459.29	1654.56
301	12	5	-	108	1444.08	1552.46	-	1626.99
302	17	4	2	117	1650.8	1366.75	768.35	1742.82
303	10	3	1	117	1559.48	1561	1690.6	1735.55
304	10	4	2	67	1538.78	1640.65	576.6	1531.99
305	9	2	-	51	1312.32	1985.8	-	1832.06
306	25	4	-	116	2032.98	2778.25	-	2483.51
307	13	2	-	76	1190.85	1228.8	-	1388.74
308	7	3	-	57	1743.76	1163.83	-	1825.62
309	20	-	10	33	1561.22	-	1189.88	1851.71
310	16	3	4	62	1316.27	1982.1	1271.05	1809.91
313	21	19	4	73	1882.36	1947.85	2071.6	2374.79
314	22	7	9	25	1519.57	1549.71	1257.92	1434.46
315	41	48	3	48	1300.26	1830.23	968.53	1850.02
319	30	67	4	87	1584.7	1707.49	1676.15	1786.7
515	36	2	-	162	8956.58	11898.85	-	8962.56
610	63	5	2	190	13070.53	15667.64	14669.6	14077.22
611	78	52	3	74	2349.84	2448.86	2012.87	2326.1
612	57	52	2	63	2457.31	3196.43	1596.65	2871.69
614	104	2	2	109	11392.84	11855.95	13611.8	12910.18
618	40	33	2	37	2443.35	2663.89	3484.75	2916.31

Table 3.4: Critical Stop Codes with Mean Energy and Entries Count for each Machine

The issue with these STOP CODES arises from the insufficient data available, making it difficult to draw any meaningful conclusions. The energy values associated with stop

codes amount to **31691**. However, when we distribute these entries across each machine and STOP CODE, the data becomes significantly limited, as demonstrated in Table 3.4. Furthermore, it's notable that there is no discernible trend indicating that one STOP CODE consistently exhibits a significantly higher mean energy value compared to others.

3.2.2 T-Test Analysis

The T-Test is a statistical method used to determine if there is a significant difference between the means of two groups, which may be related in certain features or completely independent.

It is widely used in hypothesis testing to ascertain whether to accept or reject the null hypothesis. In the context of our study, the T-Test was applied to assess the variance in energy consumption between two scenarios: one considering all energy consumption by the machines and the other focusing on energy consumption 15 minutes before the machines stopped.

Given the observation that the energy consumption associated with machine stoppages was approximately half of the overall energy consumption, we employed both independent and related T-Tests to analyze the data.

Independent T-Test:

The independent T-Test compares the means of two independent groups in order to determine whether there is statistical evidence that the associated population means are significantly different. The test is applicable when the means are collected from two distinct groups.

```
from scipy import stats

# Independent t-test
t_stat, p_value = stats.ttest_ind(combined_averages['Old_Average'],
                                  combined_averages['New_Average'],
                                  nan_policy='omit')

print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

Results: T-statistic: -2.281232016409947, P-value: 0.027670004000734216. Since the p-value is less than 0.05, we conclude that there is a significant difference between the two sets of averages.

Related T-Test:

The related T-Test, or paired T-Test, compares the means from the same group at different times, or under different conditions. It is used when the samples are not independent.

```
# Related t-test
t_stat, p_value = stats.ttest_rel(combined_averages['Old_Average'],
                                  combined_averages['New_Average'],
                                  nan_policy='omit')

print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

Results: T-statistic: -4.303034163635538, P-value: 0.00031480066717622476. With the p-value significantly less than 0.05, it indicates a significant difference between the

two sets of averages, reaffirming the findings from the independent T-Test.

The results from both tests indicate a **significant difference** in energy consumption between scenarios, providing statistical evidence to support the hypothesis that the operational conditions (including stoppages) have a substantial impact on energy consumption patterns. This analysis underscores the importance of considering specific operational intervals when assessing energy efficiency in industrial settings.

3.2.3 Production Quality Analysis

The following step in our Explorative Data Analysis journey was to analyze the production quality of the pieces [5] . To do this we took into account two columns in our dataset.

1. **AVG Good Quality**: This column measures the average quality of pieces that are considered as meeting or exceeding the production standards. The values are presented as a numerical score, where a positive value indicates a higher degree of precision or quality that exceeds the standard requirements, while a negative value suggests a deviation below the standard quality level. Essentially, this metric evaluates the excellence of the product being manufactured.

2. **AVG Scrap Quality**: Contrarily, this column quantifies the average quality of pieces that are deemed unsuitable for use and are discarded as scrap. These values could reflect the degree of defects or the extent to which scrapped pieces deviate from the required standards. A higher positive value indicates more significant deviations or a higher degree of defects in the scrapped pieces, suggesting poorer overall production quality or issues in the manufacturing process.

We decided to associate the quality of the production with each stop code, to understand how these two elements were correlated between each other, while also calculating the average energy consumption and seeing how would this factor in. We obtained the following results:

[5]The notebook used production quality analysis: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/quality_production_analysis.ipynb

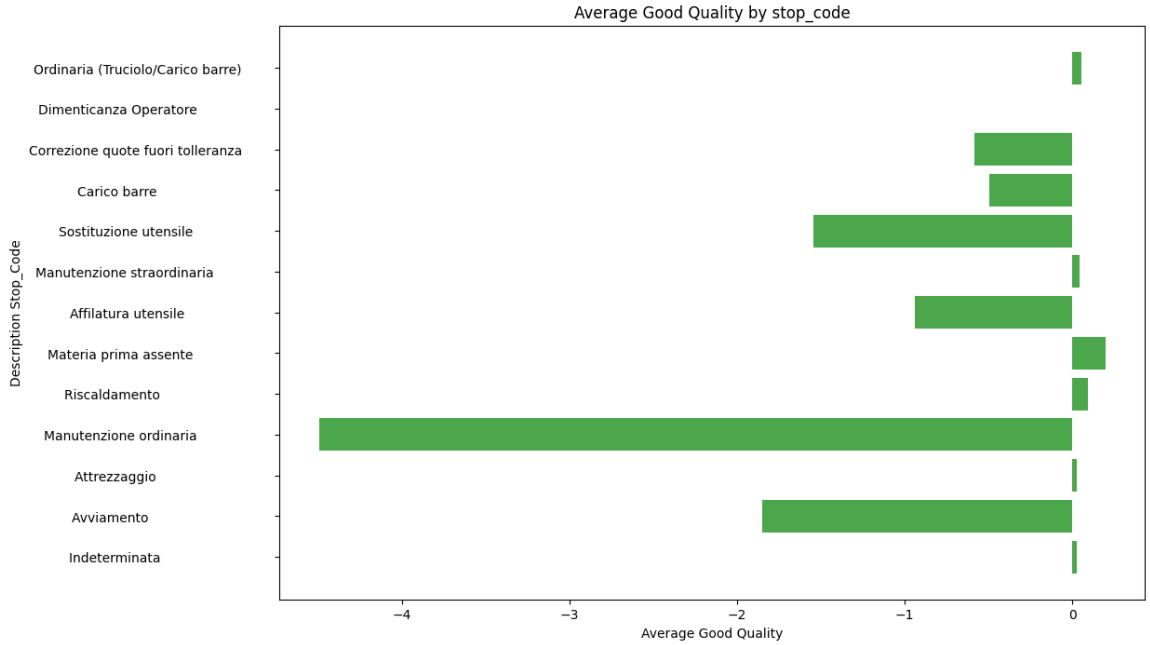

Figure 3.4: AVG Good quality associated with stop codes

E. Count	Stop Code	Description	A. Good Q	A. Scrap Q
30756	0	Indeterminata	-0.028125	0.227175
7787	1	Avviamento	-0.740593	0.966226
1392	2	Attrezzaggio	0.010776	0.124282
2316	4	Manutenzione ordinaria	-1.924870	2.072539
690	6	Riscaldamento	0.042029	0.039130
15	7	Materia prima assente	0.066667	0.000000
859	8	Affilatura utensile	-0.607683	0.623981
202	9	Manutenzione straordinaria	-3.787129	3.811881
5708	10	Sostituzione utensile	-2.720918	2.747547
2572	11	Carico barre	-0.332037	0.398911
362	12	Correzione quote fuori tolleranza	-0.251381	0.870166
21	13	Guasto a Gestionale	0.000000	0.000000
2	14	Dimenticanza Operatore	0.000000	0.000000
50145	15	Ordinaria (Truciolo/Carico barre)	0.067325	0.129405

Table 3.5: Summary of Entries Count, Description, Stop Code, Average Good Quality, and Average Scrap Quality of the stoppages dataset

Entries Count	Stop Code	Avg Good Quality	Avg Scrap Quality	Avg Energy
11590	0	0.025626	0.209146	3845.794254
2968	1	-1.847709	1.936995	3194.764858
405	2	0.029630	0.049383	1577.617531
723	4	-4.496542	4.615491	4474.461272
235	6	0.093617	0.000000	3425.416596
5	7	0.200000	0.000000	2871.340000
359	8	-0.938719	0.947075	2350.142618
64	9	0.046875	0.000000	2112.242188
1879	10	-1.542842	1.562001	4302.586535
1019	11	-0.494603	0.536801	3089.616487
147	12	-0.585034	2.040816	3651.067347
1	13	0.000000	0.000000	37.800000
12296	15	0.054652	0.126871	3937.463744

Table 3.6: Summary of Entries Count, Description, Stop Code, Average Good Quality, Average Scrap Quality and Average Energy for the merged dataset of energy and stoppages

The graph in Figure 3.4 illustrates the average good quality categorized by stop code, where the *y-axis* represents various stop codes or conditions, and the *x-axis* displays the values of average good quality. Negative values on this scale suggest that values closer to zero indicate better quality. This graph is complemented by Table 3.5, providing detailed information. The most significant decline in quality is observed in conditions corresponding to **Manutenzione Straordinaria** (extraordinary maintenance) and **Sostituzione utensile** (tool replacement), suggesting that these actions are associated with a decrease in the good quality of pieces.

Stop codes with minor or no impact on average good quality, such as *Indeterminata* (indeterminate), *Avviamento* (startup), and *Attrezzaggio* (setup), also exhibit low levels of scrap quality, indicating a stable production process during these conditions.

In Table 3.6, we present a summary of data considering only the stoppages that have an assigned energy value. Here, we observe significant changes in values. The most predominant stop code in terms of average scrap quality is **Manutenzione Ordinaria** (ordinary maintenance), followed by **Correzione quite fuori tolleranza** and **Avviamento**. *Manutenzione Ordinaria* also exhibits the highest average energy value at **4474**, followed by *Sostituzione utensile* with **4302** and *Ordinaria* with **3937**.

When examining the values of the most critical stop codes (Section 3.2.1), it is notable that stop codes **4** and **10** demonstrate noteworthy values in terms of scrap quality and average energy levels, suggesting potential anomalies worthy of further investigation.

3.2.4 Machinery Maintenance Period Analysis

In this notebook[6] we performed an investigation on the machinery maintenance period.

In this investigation, we delved into the patterns of machinery maintenance, specifically focusing on occurrences categorized under stop code 4 (Figure 3.5 and Figure

[6]The notebook used for ordinary maintenance analysis: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/machinery_maintenance_periods_analysis.ipynb

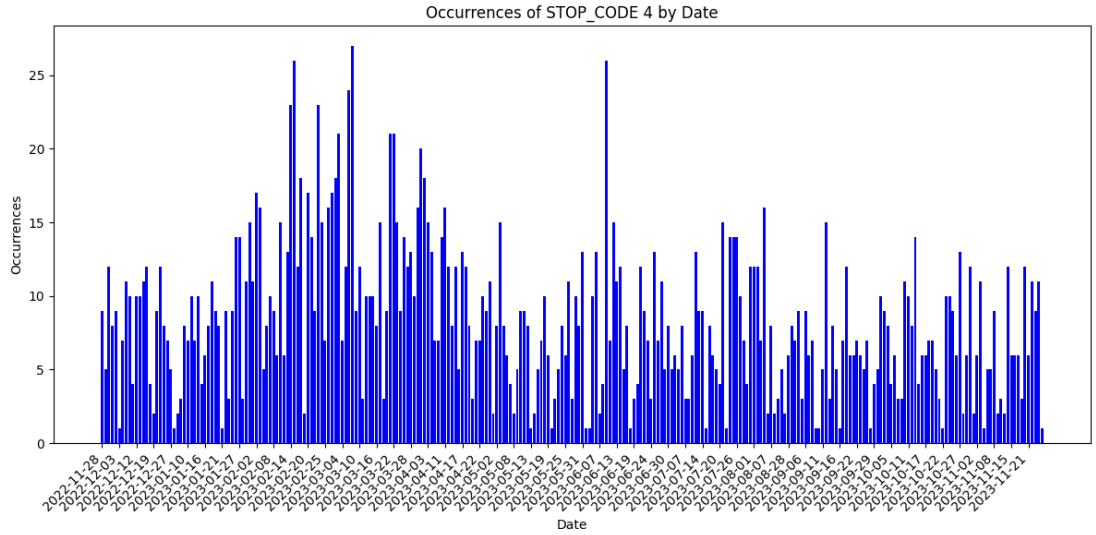

Figure 3.5: Occurrences of stop code 4 across all machines

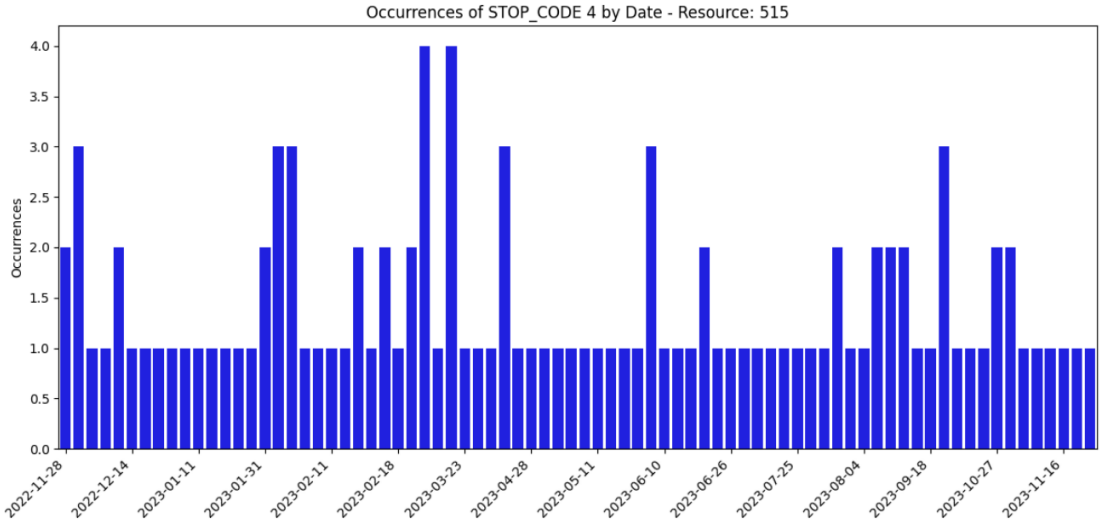

Figure 3.6: Occurrences of stop code 4 in machine 515

3.6), designated as ordinary maintenance. Our observations revealed that the frequency and timing of these maintenance stops varied significantly from day to day, suggesting potential discrepancies in the classification of maintenance activities or an inherent inconsistency in the scheduling of these maintenance events. Such variability raises questions about the use of the term "ordinary maintenance," which traditionally implies a routine or scheduled nature of maintenance activities.

The investigation further entailed a comprehensive analysis of each machine, involving the creation of daily graphs to track the evolution of ordinary maintenance stops over time. This approach allowed for a detailed examination of maintenance patterns, facilitating the identification of any irregularities or trends across different machines.

The observed inconsistencies in the occurrence of ordinary maintenance stops could have several implications. Firstly, it may indicate a need for revising the criteria or processes used to classify maintenance activities, ensuring that the labeling accurately reflects the nature and scheduling of the maintenance work. Secondly, the variability in maintenance scheduling might suggest a lack of standardized procedures or inefficiencies in maintenance planning, which could lead to increased downtime, reduced machinery lifespan, and potential impacts on production schedules.

3.2.5 Produced Pieces Dataset Analysis

As our last analysis, took into consideration the dataset with the number of produced pieces [7] . At first, we took into account the energy levels of the machine and confronted them with our previous findings in order to see if there would be some meaningfull differences. In Figure 3.7 we can see the energy level measurements of the different datasets, and we can see that the differenced between the energy levels associated to stoppages and the ones associated with produced pieces are almost the same. This trend is constant with all the machines present in the datasets.

We then conducted a further investigation on the correlation between the number of producted pieces and the energy values, to see if an higher number of producted pieces leads to an increase on energy consuption. To do so we created dataframe that had the following structure:

Table 3.7: Description of the merged_pieces DataFrame

Column Index	Column Name	Non-Null Count	Data Type
0	ID	715058	int64
1	COD_MACC	715058	float64
2	COD_ART	715058	object
3	TIMESTAMP_INIZIO	715058	object
4	TIMESTAMP_FINE	715058	object
5	NUMERO_PEZZI_PROD	715058	float64
6	energy_value	670645	object

We then decided to do a general scatter plot graph (Figure 3.8) to see the distribution of this correlation and we got the following result.

[7]The notebook used the analysis of produced pieces and energy: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Analysis_and_Exploration/energy_overall_analysis.ipynb

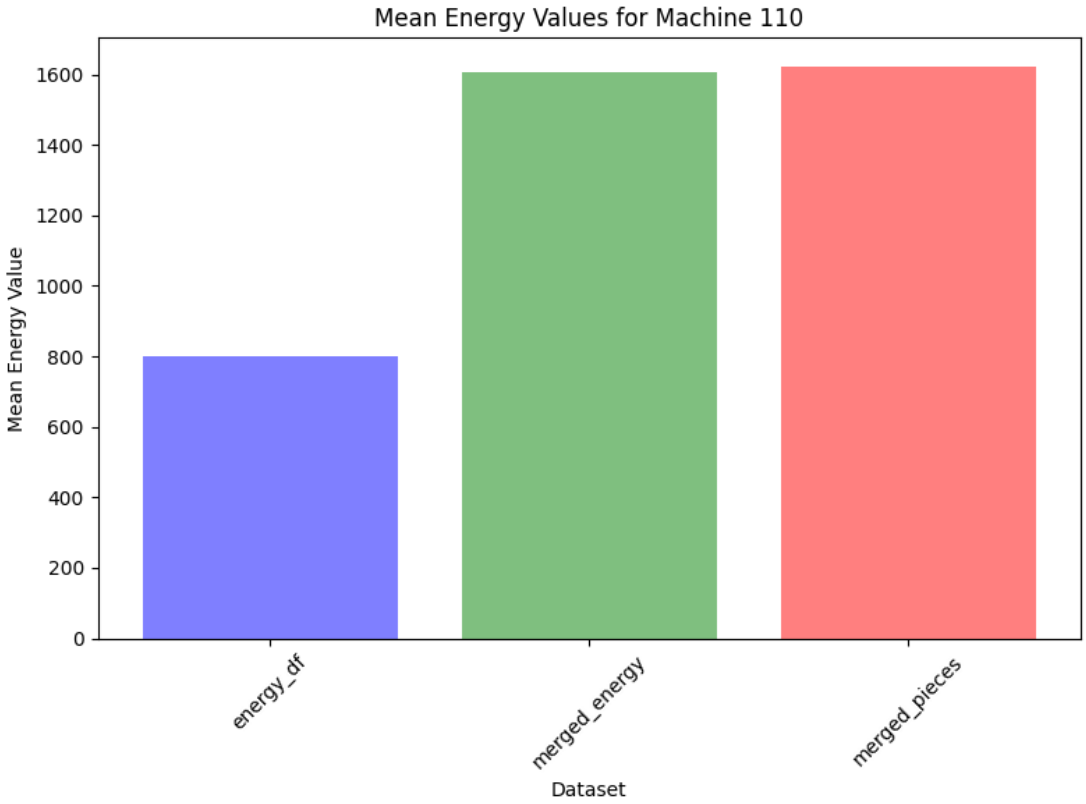

Figure 3.7: Energy level measurements with the different datasets in machine 110

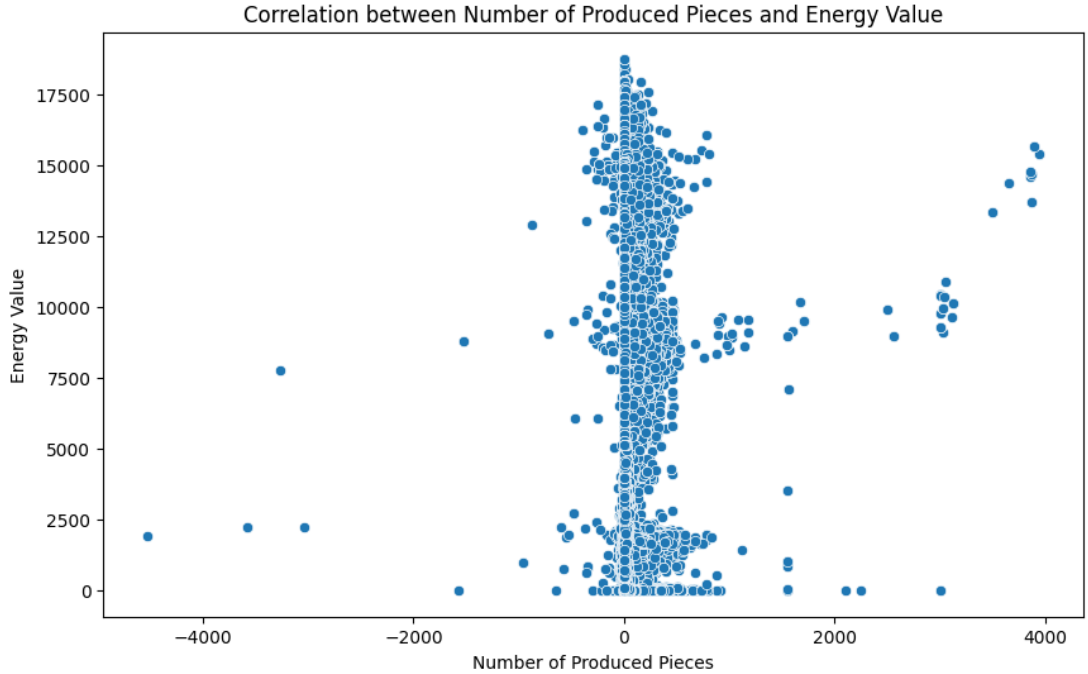

Figure 3.8: Correlation between energy and producted pieces for all machines.

The scatter plot shows a wide dispersion of points along the y-axis (Energy Value), while many data points are concentrated around the zero mark on the x-axis (Number of Produced Pieces). This indicates a lot of variation in energy values for a similar number of produced pieces. There are also some points with negative values for the number of produced pieces, which could indicate either data entry errors or a specific encoding in your dataset that would need to be clarified.

We then decided to do a correlation heatmap (Figure 3.9) between these two columns of our dataframe, which shows a correlation coefficient of 0.15 between the NUMERO_PEZZI_PROD and energy_value. This value indicates a very weak positive correlation between the two variables. In a correlation coefficient, a value close to 1 or -1 indicates a strong correlation, while a value close to 0 indicates no linear correlation. Hence, based on the coefficient, we cannot say that there's a strong linear relationship between the number of pieces produced and the energy value.

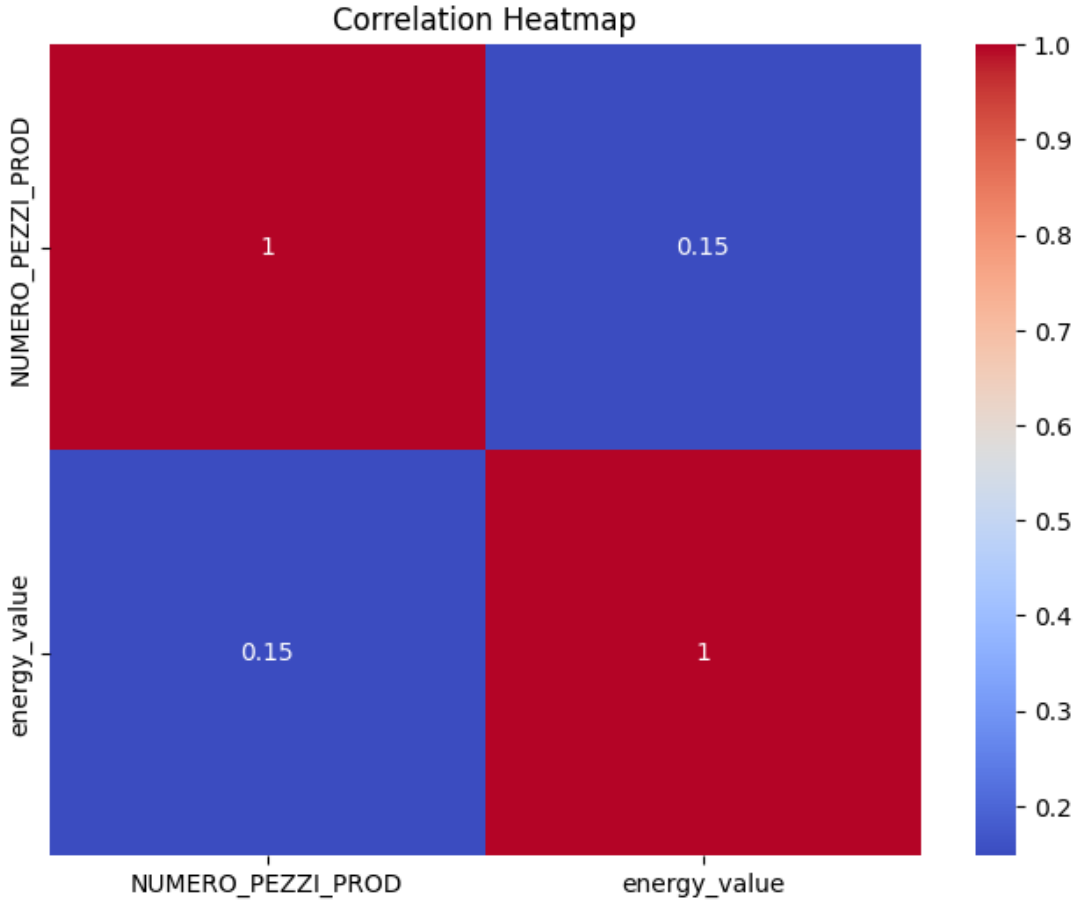

Figure 3.9: Heatmap between energy and producted pieces for all machines.

The observations suggest that the number of produced pieces may not be a strong predictor of the energy value or vice versa. This could be due to a variety of factors, such as:

The nature of the data and the processes involved might not have a linear relationship. There could be other variables that are affecting the energy value, which are not considered in this simple two-variable analysis. There may be outliers or errors in the data affecting the correlation.

We then plotted scatter plots for each machine, to see if this error was caused by some of them that could have registered some anomal events. This unfortunately was not the case; in Figure 3.10, which represent the data for the machine 610, we can see how, for the same number of pieces produces, the energy level increases, and this trend is present in all the machines.

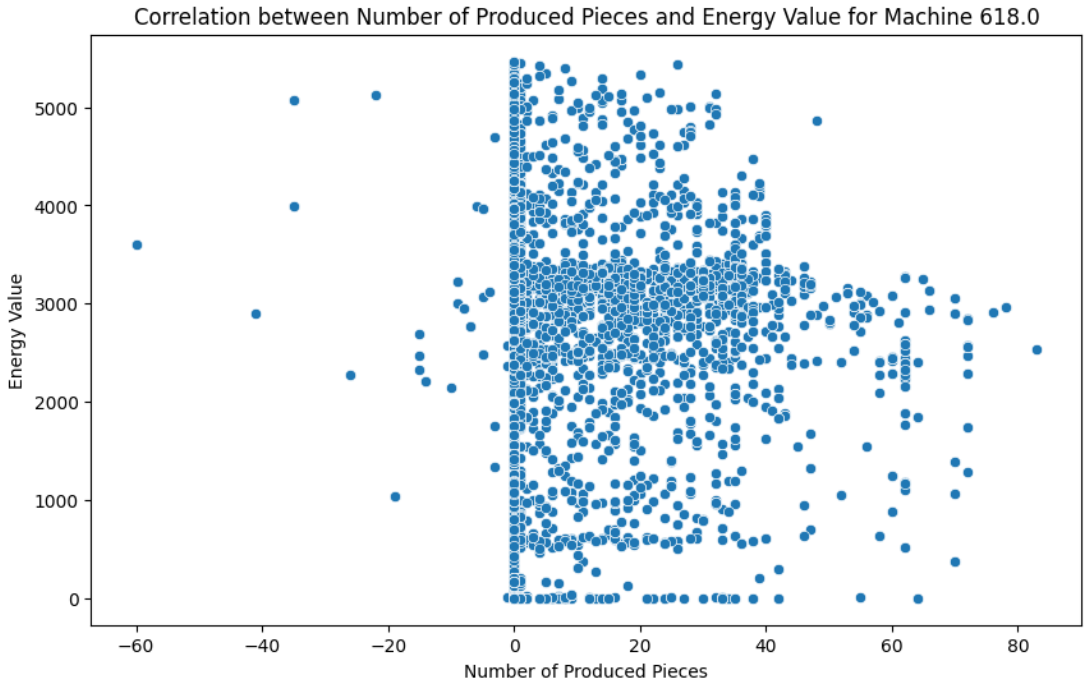

Figure 3.10: Scatter plot for machine 610

In fact this scatter plot suggests that there is no clear correlation between the energy value and the number of pieces producted.

3.3 Conclusions

From our analysis, we weren't able to find some significant correlation between stop codes, consumed energy and produced pieces. Our finding highlighted that:

- The provided datasets are insufficient. The number of stoppages of the machines and respective energy values are **31691**, too few when considering that they have to be divided across multiple machines and stop codes;
- The difference between the means of overall energy values across machines and ones associated with stop codes is **significant**;
- The description of **stop code 4** is questionable after verifying how many times it occurs in each machine;
- The energy values of the merged stoppages dataset are almost the same as the ones from the merged produced pieces dataset
- The same number of produced pieces corresponds to different levels of consumed energy across all machines;

The only possible correlation was found in Section , where the average quality of production was being compared to stop codes and energy levels. There, *stop code 4* had both the **highest average** of scrap quality and **highest levels** of average energy. The problem is that we have only **723** entries of stop code 4 across all machines, which makes the creation of a ML model quite challenging.

4. Kestra Orchestration

In this chapter we will analyze the pipeline that we created for the analysis of the data using Kestra [1] , an **orchestration tool** capable of building workflows that include a wide range of operations, from execution of scripts, to terminal commands.

4.1 Objectives

4.2 Data Pipeline

In this section the pipeline and its structure will be analyzed, alongside the used code for it.

4.2.1 Pipeline structure

The proposed structure for the pipeline is comprised of the following tasks:

- **Dataset download**: the raw datasets, which are present on an **AWS S3 bucket**, are going to be downloaded and made available for other tasks inside Kestra to use them
- **Dataset cleaning and analysis**: in this task **PySpark scripts** will process the downloaded datasets and delete non necessary columns and rows, in order to keep only the essential data for the analysis, after which it will perform analysis operations on the data to give to the user some insights
- **Dataset upload on database**: in this step the cleaned datasets are uploaded on a database, which could be **MongoDB** or **PostgreSQL**, in order to be made available for other operations
- **Visual analysis representation**: in this last task, **Power BI** is used to fetch the datasets from the database and visual representations are created in order to observe the behavior of the machines

A visual representation can be observed in Figure 4.1.

4.2.2 Pipeline Code

The written code is divided in two parts, the *PySpark* code needed to run the jobs for processing and cleaning and the *Pipeline* code needed to orchestrate the various tasks successfully.

[1] Tool website: https://kestra.io/

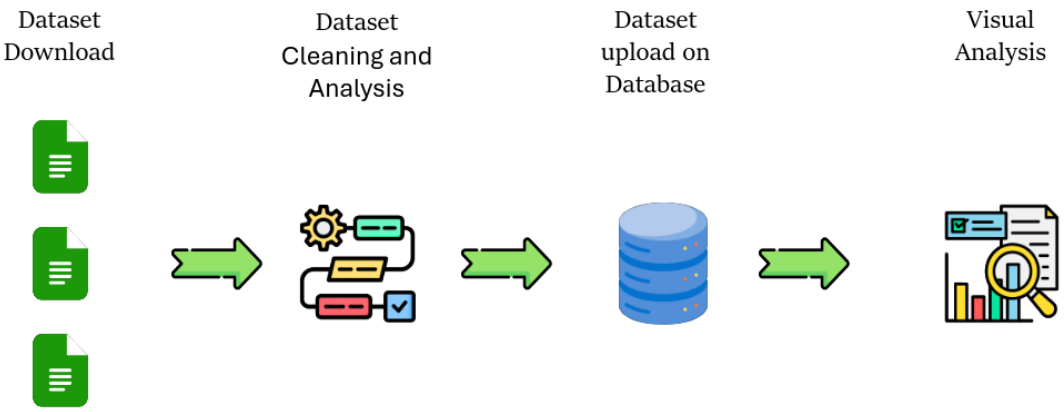

Figure 4.1: Visual representation of the Pipeline Structure

PySpark

We leveraged the power of PySpark [2] to clean our dataset and perform analysis like we did with Python on Google Colab. Apache Spark is a much more scalable solution and since we are in the domain of Big Data this is an essential point, through all of this our objective is to automize the dataset cleaning phase and orchestrate it in a way that every time we receive new data from the machines we are able to perform a basic analysis and clean the dataset in a way that we can work on it with a Machine Learning model.

Energy Data Cleaning

```
1 class EnergyDataCleaning:
2     def __init__(self, spark, input_file_path, output_file_path):
3         self.spark = spark
4         self.input_file_path = input_file_path
5         self.output_file_path = output_file_path
```

This part of code defines a class called Energy Data Cleaning, which is the class where we implemented the logic to clean the data related to the initial energy bak folder. The constructor of the class has three main elements:

1. **'self.spark = spark'**: this stores the Spark session passed to the constructor in the instance variable *spark*. This session will be later on used to read and process the data.

2. **'self.input_file_path** = input_file_path': this takes the file path of the raw data that needs cleaning and stores it in an instance variable.

3. **'self.output_file_path = output_file_path'**: here we store the file path where the cleaned data should be saved after all the cleaning operations are completed.

After this we followed the steps that we already executed on Google Colab to clean all the energy dataframe. As we can see from the following part of code:

[2]PySpark Project with Source code: https://github.com/Staffilon/KestraDataOrchestrator/tree/main/Analytics-Framework/Spark-Cleaning-Analysis

```
1      def clean_data(self):
2          df = self.spark.read.csv(self.input_file_path, header=True, ...
               inferSchema=True)
3
4          initial_count = df.count()
5          print(f"Initial data count: {initial_count}")
6
7          df = df.withColumn("TimeStamp", to_timestamp("TimeStamp", ...
               "yyyy-MM-dd HH:mm:ss+00:00"))
8
9          df = df.orderBy("channel", "TimeStamp")
10
11         df_clean = df.filter(df["Ea_Imp"].isNotNull())
12
13         cleaned_count = df_clean.count()
14         print(f"Cleaned data count: {cleaned_count}")
15         print(f"Number of removed rows due to null values: ...
               {initial_count - cleaned_count}")
16
17         df_clean.write.csv(self.output_file_path, header=True, ...
               mode='overwrite')
18
19         print(f"Sorted and cleaned CSV saved to {self.output_file_path}")
20
21         return df_clean
```

First of all, in line **7**, we perform a conversion of the **'TimeStamp'** column into a proper datetime object. This is crucial to ensure consistency and further time based analysis.

After this small step, we sort the Dataframe by the **'channel'** column, which represents a machinery, and the **'TimeStamp'** column. This ensures we can perform data analysis tecniques that rely on the order of the data, such as time series analysis.

Another major factor we want to ensure is that we dont need any null energy value, since we know our starting dataframe can have null values, and this could happen in the future aswell, we decided to filter all the null values of **'Ea_Imp'**, which represents the energy consuption of each machine. Later on we display also the amount of logs we deleted through this operation, as visible from the **lines 11-15**.

After the necessary cleaning operations are performed we finally save the dataframe in the desire output, this way we can work on a much cleaner dataset and perform data analysis or machine learning models on top of it.

Fermate Data Cleaning

In the **fermate_data_cleaning.py** file we can find the data cleaning regarding the fermate dataframe, this dataframe contains the stopping related to each machine.

From the following code we can distinct several steps that took part into this:

```
1        def clean_data(self):
2          df = self.spark.read.csv(self.input_file_path, header=True, ...
               inferSchema=True)
3
4          initial_count = df.count()
5          print(f"Initial data count: {initial_count}")
6
```

```
7           df = df.drop('START_DATE', 'END_DATE')
8
9           df = df.withColumn('RESOURCE', expr("int(ltrim(RESOURCE, '0'))"))
10
11          df = df.withColumn('SHIFT_DATE', to_timestamp('SHIFT_DATE', ...
                'yyyy-MM-dd HH:mm:ss').cast('timestamp'))
12          df = df.withColumn('SHIFT_START', to_timestamp('SHIFT_START', ...
                'yyyy-MM-dd HH:mm:ss').cast('timestamp'))
13          df = df.withColumn('SHIFT_END', to_timestamp('SHIFT_END', ...
                'yyyy-MM-dd HH:mm:ss').cast('timestamp'))
14
15          all_data_sorted = ...
                self.spark.read.csv("path_to_sorted_cleaned_energy_data.csv", ...
                header=True, inferSchema=True)
16          all_data_sorted = all_data_sorted.withColumn('channel', ...
                col('channel').cast('string'))
17
18          df = df.join(all_data_sorted, df.RESOURCE == ...
                all_data_sorted.channel, 'inner')
19
20          df.write.csv(self.output_file_path, header=True, ...
                mode='overwrite')
21
22          final_count = df.count()
23          print(f"Final data count: {final_count}")
24          print(f"Number of rows after cleaning: {final_count - ...
                initial_count}")
25
26          print(f"Cleaned CSV saved to {self.output_file_path}")
27
28          return df
```

This cleaning process consists of the following steps:

1. **Reading the Data:**We begin by loading the data using Spark's read.csv method. This allows us to read the CSV file located at self.input_file_path while inferring the schema automatically and considering the first row as headers. This step is crucial for establishing a baseline dataset from which to work.

2. **Dropping Irrelevant Columns:**We decided to remove the columns START_DATE and END_DATE using df.drop. This step is likely based on our understanding that these columns are not relevant for our analysis or contain redundant information.

3. **Standardizing the 'RESOURCE' Field:**The RESOURCE column is treated by removing leading zeros and converting it to an integer. This standardization makes the data more consistent and easier to work with, especially for numerical analyses or when joining with other datasets.

4. **Timestamp Conversion:**The SHIFT_DATE, SHIFT_START, and SHIFT_END columns are converted into timestamp format. This conversion is essential for any time series analysis or operations that require date and time calculations. It ensures that the date and time data are in a uniform and usable format.

5. **Joining with Another Dataset:**We read another dataset, all_data_sorted, from a specified path. After casting the channel column to a string, we perform an inner join with our primary dataframe df on the RESOURCE and channel columns.

6. **Writing the Cleaned Data:**After cleaning and transforming the data, we write the resulting dataframe to self.output_file_path as a CSV file, ensuring to include headers and overwrite any existing file. This step finalizes our cleaning process, making the cleaned data readily available for further analysis or reporting.

4.2.3 Analysis and Main

In our project, we have developed a comprehensive analysis framework focusing on two key areas: Energy consumption and machine stoppage ('Fermate') data. This framework is designed to provide insights into energy usage patterns and stoppage frequencies, durations, and reasons within an industrial setting.

Energy Analysis

The `EnergyAnalysis` class is central to our energy consumption analysis. It performs several crucial tasks:

- **Data Preprocessing:** Converts the 'TimeStamp' column to a standardized format, ensuring consistency in time-related data.
- **Total Consumption:** Calculates the sum of energy consumed ('Ea_Imp') grouped by each channel.
- **Average Consumption:** Computes the average energy consumed per channel.
- **Maximum Consumption:** Identifies peak energy usage for each channel.
- **Running the Analysis:** Executes the above methods and displays the results, offering a comprehensive view of energy usage.

Fermate Analysis

Parallelly, the `FermateAnalysis` class focuses on analyzing machine stoppage data, providing insights into operational efficiency and maintenance needs. Key functionalities include:

- **Data Preprocessing:** Standardizes the 'TimeStamp' format based on the 'SHIFT_DATE' column.
- **Stop Frequency:** Calculates the frequency of stops on a daily basis.
- **Stop Duration:** Computes the average duration of each stop, categorized by resources.
- **Stop Reason Count:** Analyzes the count of different stop reasons.
- **Time Series Analysis:** Offers a monthly view of stoppage patterns.
- **Stop Code Distribution:** Identifies the distribution and frequency of various stop codes.
- **Running the Analysis:** Conducts the above analyses and displays the findings.

Main Workflow

Our project is configured to work seamlessly with Kestra, an orchestration and scheduling platform. The main workflow consists of the following steps:

1. **Initializing Spark Session:** Sets up a Spark session which is the backbone for our data processing tasks.
2. **Data Cleaning:**
   - Cleans energy data using specified file paths in Kestra.
   - Similarly, cleans fermate data, preparing both datasets for analysis.
3. **Performing Analyses:**
   - Executes the `EnergyAnalysis` and `FermateAnalysis` on the cleaned datasets.
   - Displays the results, providing actionable insights.
4. **Error Handling:** Catches and prints any exceptions that occur during the process.
5. **Terminating Spark Session:** Ensures a clean and efficient closure of the Spark session upon completion or in case of errors.

This structured approach, combined with the integration with Kestra, allows for a scalable and efficient analysis pipeline, capable of handling large datasets and complex computations typical in industrial environments.

Orchestration Code

The *Orchestration code* [3] is a *yaml* file that contains the tasks which have to be done in order to execute the flow described at Section 4.2.1.

First we have the flow **id and namespace**, necessary on Kestra to respectively identify the task and the position where it is executed.

```
1 id: pipeline
2 namespace: tormatic
```

The next step involves starting a **parallel task**, which allows us to download simultaneously the datasets needed for analysis.

```
1 - id: parallelDownload
2     type: "io.kestra.core.tasks.flows.Parallel"
```

Inside the parallel task, we have two sub tasks, designed to download respectively the raw *fermate* nad *energy* datasets.

[3] File containing the orchestration code: https://github.com/Staffilon/KestraDataOrchestrator/blob/main/Data-Orchestration/pipeline.yml

```
- id: downloadFermateDataset
        type: io.kestra.plugin.aws.s3.Download
        accessKeyId:  "{{ secret('AWS_ACCESS_KEY_ID') }}"
        secretKeyId:  "{{ secret('AWS_SECRET_ACCESS_KEY') }}"
        region: "eu-west-2"
        bucket: "tormatic"
        key: "all_uncleaned_fermate.csv"
```

After obtaining the datasets, the next piece of work should have been their processing and analysis, but due to problems explained in Section 4.3, we have been unable make the code work.

As the final step, we have the parallel task which involves the upload of the processed datasets on the database.

```
  - id: parallelUpload
    type: "io.kestra.core.tasks.flows.Parallel"
    tasks:
```

Inside the parallel task, two sub tasks are used in order to upload the datasets on *PostgreSQL*, using the **CopyIn** type of action.

```
      - id: databaseUploadFermate
        type: "io.kestra.plugin.jdbc.postgresql.CopyIn"
        url: "{{ secret('PSQL_URL') }}"
        username: tormatic_owner
        password: "{{ secret('PSQL_PASSWORD') }}"
        format: CSV
        from: '{{ all_uncleaned_fermate.csv }}'
        table: datasets
        header: true
```

4.3 Encountered Problems

During the writing of the pipeline we encountered different problems in regards to configuration, which we will talk below:

- **PySpark code:** we could not run *PySpark* code inside Kestra. We tried both the *Spark CLI* and running it inside *Python* scripts, but either we were getting the error *JAVA_HOME is not set*, or some configuration problem regarding the lifecycle of a bus. We tried setting up the JAVA_HOME both with Docker and with the yaml file, but no solution was valid. As a side note, not even the script provided by Kestra would run successfully.

- **Python code:** after not being able to run the PySpark code, we decided to convert it into *Python code*, but in the same way we were getting errors upon errors. Among them, one was that the *pandas* library was not being recognized, even though we were installing it the proper way and, in other tested scripts, it was working correctly.

- **Database:** the last problem regards the database connection to which we upload the cleaned and processed datasets. We tried with both *MongoDB* and

PostgreSQL, but with neither we were able to succeed. MongoDB was returning errors regarding the passed files, even though we were following the guidelines of the documentation, while with PostgreSQL we could not establish a connection, even after trying two different providers. One thing to note is that the documentation provides only examples with local connections. As a way to still be able to do the *Analysis Visualization* step, we decided to manually upload the processed datasets on MongoDB.

We tried to reach out to the community on *Slack* in order to find some solutions, but we received no answer.

4.4 Analysis Visualization

In this section we are going to talk about the **Power BI** application [4] and how we used it to visually represent the analysis on the cleaned and processed datasets.

4.4.1 Objectives

The objectives in using Power BI include creating interactive dashboards and detailed visualizations to better understand trends related to consumption, downtime, and the quality of parts produced by industrial machines. By leveraging Power BI's advanced analytics capabilities, we aim to identify significant correlations within the data and gain insights to optimize operational efficiency, enhance preventive maintenance, and streamline production processes.

4.4.2 Power BI integration with MongoDb

For managing the data related to consumption, downtime, and quality of parts across different machines, we utilized **MongoDB** as the storage system. MongoDB was chosen for its flexibility and scalability in handling large volumes of structured and unstructured data from diverse sources. Subsequently, we linked the MongoDB database to Power BI to conduct in-depth analysis on the data. This linkage facilitated easy extraction of data from the database, enabling us to create interactive visualizations and informative dashboards within Power BI. Integrating MongoDB with Power BI allowed us to leverage the platform's advanced analytics capabilities, enabling us to identify trends, correlations, and valuable insights to optimize operations and enhance the performance of industrial machines. The first step for perform analysis is **connect Power BI to MongoDB**. Initially, within the Power BI interface, you select the option to import data from a new source. Then you choose MongoDB as the type of data source and provide the access credentials to the MongoDB database, including the server name, database name, and authentication credentials if necessary. A visual representation can be observed in Figure 4.2. After establishing the connection, Power BI displays a preview of the data available in the MongoDB database. At this point, you can select the tables or data collections to import into Power BI for analysis. Once the data is imported, you can begin creating visualizations and dashboards using Power BI tools.

[4] Tool website: https://powerbi.microsoft.com/

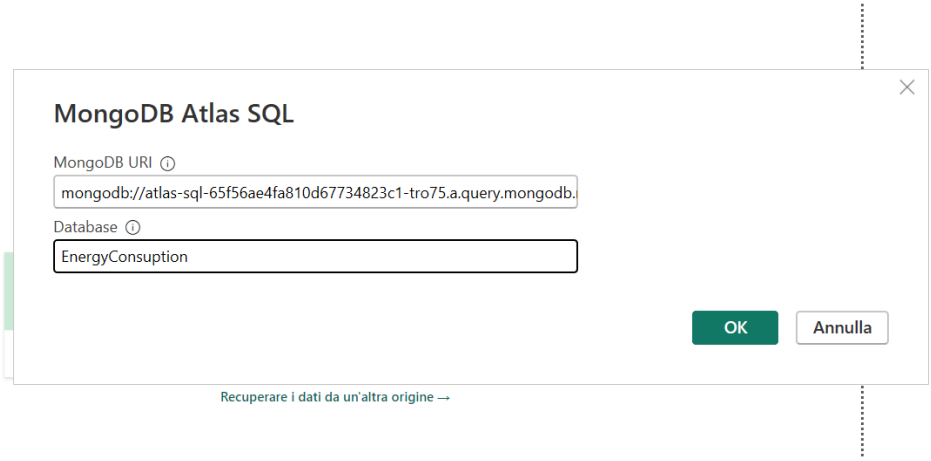

Figure 4.2: Connection MongoDb

4.4.3 Analysis

- **Energy consumption**: The initial analysis focused on energy consumption. Within this section, we employed two types of graphs: a column histogram and a pie chart. Both graphs depict the average energy consumption per month, with the pie chart additionally displaying the corresponding percentage breakdown. Furthermore, we included a table containing machine IDs, facilitating monitoring of energy consumption at both global and individual machine levels. A visual representation can be observed in Figure 4.3.

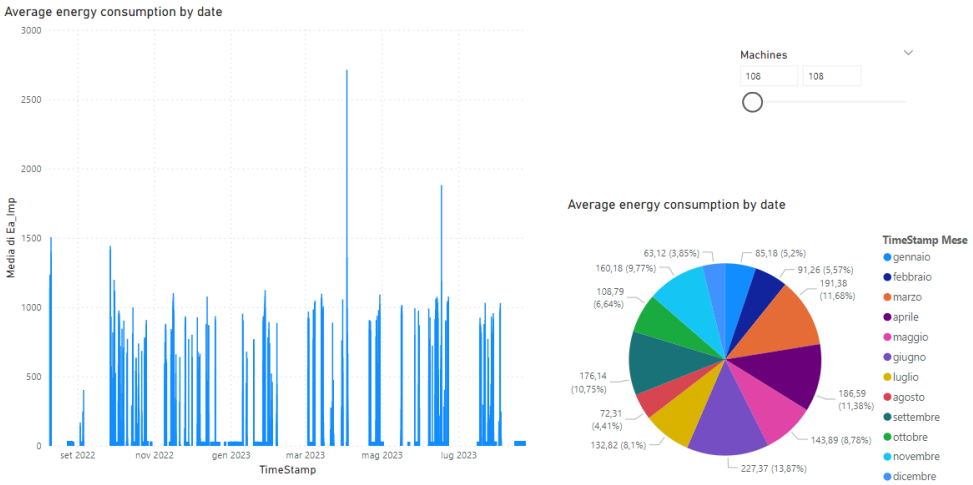

Figure 4.3: Average energy consumption during time

In addition to the average consumption, we graphically represented the maximum energy consumption value per month using a column histogram and visualized the maximum value with a gauge chart. This additional representation allows understanding not only the average consumption trend but also the maximum peaks. We also maintained the table containing the machine IDs, enabling monitoring both globally and for each individual machine. Thanks to this analysis, it is possible to identify if a machine has high consumption and try to understand the

reasons behind it, aiding in finding appropriate solutions. A visual representation can be observed in Figure 4.4.

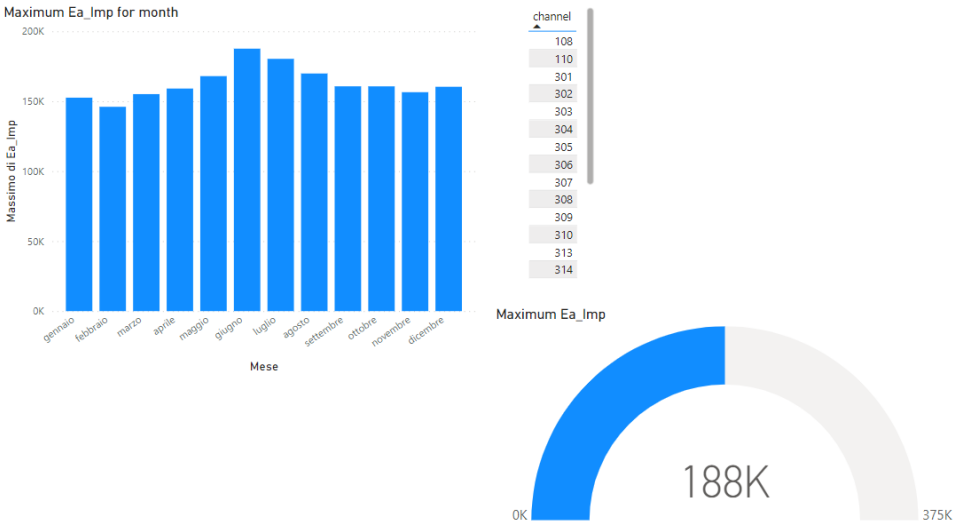

Figure 4.4: Maximum energy consumption during time

- **Machine stops**: The second analysis focused on the number of machine stops for each stop code. In this section, we used two types of graphs: a column histogram and a pie chart. Both graphs display the count of stops for each stop code, with the pie chart also showing the corresponding percentage. We utilized a table containing the machine IDs, along with the year and month, enabling monitoring both globally and for each individual machine. A visual representation can be observed in Figure 4.5.

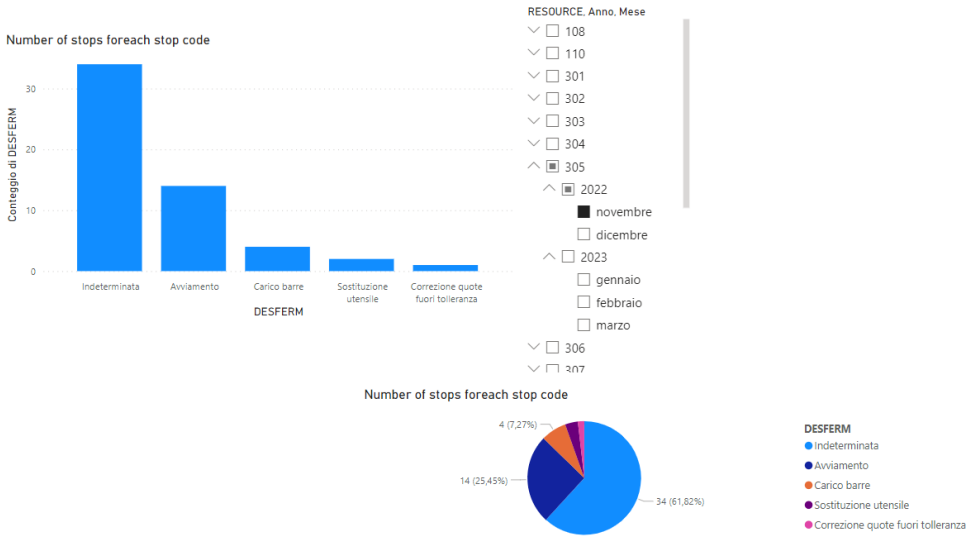

Figure 4.5: Number of stops foreach stop code

In addition to the number of stops for each stop code, we graphically represented the monthly average of *good* and *scrap* quality using a line chart. Using two pie charts, we showed the percentage of good quality for each month in the first

chart, while in the second one, we displayed the percentage of poor quality. This additional representation allows understanding not only the average trend of stops but also variations in the product quality level over time. We also maintained the table containing only the machine IDs, enabling monitoring both globally and for each individual machine. A visual representation can be observed in Figure 4.6.

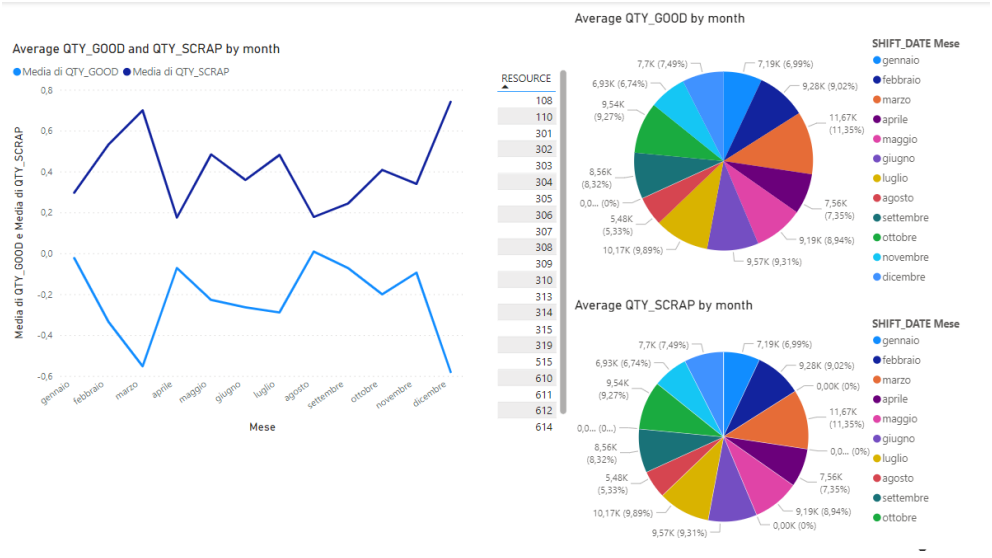

Figure 4.6: Average QTY GOOD and QTY SCRAP

- **Average energy consumption foreach stop code**:

  The third analysis focuses on the average consumption for each stop code. In this section, we used two types of graphs: a column histogram and a gauge chart. Both graphs show the average consumption for each type of stop code. We included a table containing the machine IDs, along with the year and month, allowing monitoring at both a global and individual machine level. A visual representation can be observed in Figure 4.7.

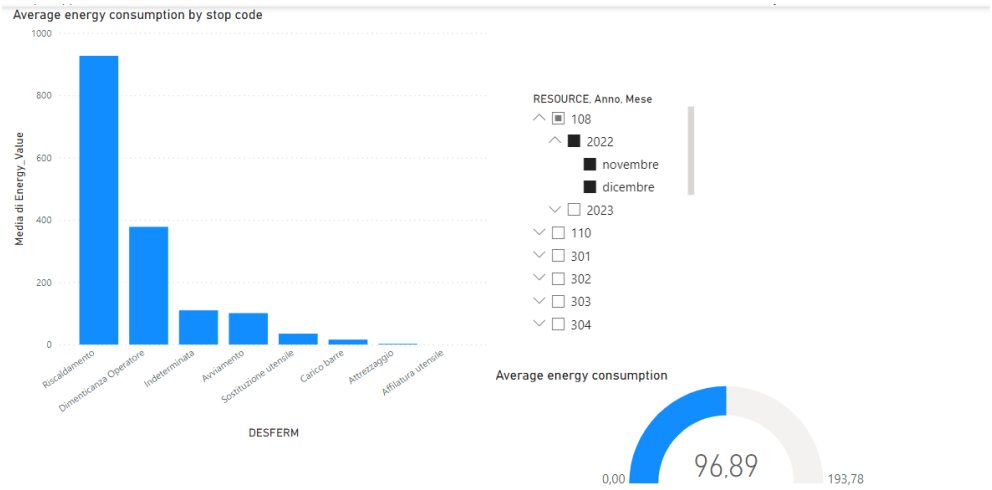

Figure 4.7: Average energy consumption by stop code

4.4.4 Conclusions

In conclusion, the visualizations provided a comprehensive analysis of data concerning energy consumption, machine stops, and product quality. Through column, pie, line, and gauge charts, we were able to examine various aspects of each variable, enabling us to identify trends, peaks, and variations over time. The inclusion of a table containing machine IDs allowed for data monitoring at both global and individual machine levels, facilitating a detailed performance evaluation. These visualizations offer a clear and informative overview of the data, which can be used to make decisions and identify areas for improvement in operational efficiency and product quality.

5. Conclusions and Future Developments

During this project, we learned more about time series, namely how to handle, process and analyze them. The *analysis part* was beneficial in regards to enhancing our skills as data scientists because, given the short comings of the provided datasets, trying to find some correlations made us look at data from *different angles*.

We also became more **proficient** with Python and learned useful constructs and and patterns that made us better programmers and Data Scientists.

Working with *Power BI* made us comprehend how visual analysis can be useful in real life scenarios, making us want to explore these technologies more.

Ultimately there were a couple of shortcomings in our project:

- **Dataset problems:** the dataset was quite challenging to work with, because it didn't provide enough data to create a model that would be able to detect anomalies or defects. Furthermore the data itself was messy, with new data randomly appearing and some labels having a questionable meaning
- **Pipeline creation:** during the final stage of the project, we were not able to fully develop the pipeline; the given documentation on the Kestra website addressed mainly basic use cases, or showed examples that were running on the local machine, leaving to the developer the task of trying to understand how would the online connection work. Furthermore, some examples that it provides **did not work**. There is no denying though that we are still novices with the software in question, so a senior developer could have come up with solutions to our problems, but this would point out to the Orchestrator not being accessible to inexperienced users.

5.1 Future Developments

The project can be further developed in regards to the following points:

- **Additional data:** if more data is provided, there can be enough *fuel* to be able to find patterns and to develop a ML model that would be able to predict anomalies or malfunctioning of machines
- **Pipeline work:** with more study and guidance on the *Kestra orchestrator*, the pipeline can be finished and the analysis work can be automated