UNIVERSITE LIBRE DE BRUXELLES

ULB FACULTÉ DES SCIENCES

FACULTY OF SCIENCES

INFO-H423 – DATA MINING

Cool train project from SNCB

Specialised Master in Big Data Sciences

Teacher :

Professor SAKR Mahmoud

Group Members :

ASSONFACK DEUMANI Patrice Miguelle 000528892

KASHAEVA Gulnur 000591511

LECLERCQ Florian 000446407

WORRALL Stephen 000591636

Academic year : 2023-2024

Table of Contents

1. Business reality

In the context of the lecture for the course “INFO-H423 - Data Mining”, the opportunity was given to work on real data problems.

The data are provided by the “The National Railway Company of Belgium” (SNCB) which is responsible for organizing and operating the rail service in Belgium. SNCB is a national company that works with many organizations and departments. For this study, we worked with the Technics Directorate and more precisely for the rolling stock team.

Figure 1, AR41 on Charleroi line.

1.1. Context

The context of the study is the problem of the cooling of diesel trains. The SNCB diesel trains are the AR41 which consists of two vehicles that are always coupled. The study aims to define if there are potentials problems or anomalies with the trains. For example, if one engine fails working this could be a strong indicator that there is an issue with the train.

The idea is to detect such an abnormal way of working by using the data provided by the SNCB and, in the end, be able to directly identify such problem. The final goal would be to speed up

the detection of potential issues to avoid long maintenance and trains issue that conduct to delay.

1.2. Data

From SNCB, we received the file "ar41_for_ulb.csv" which contains all the data we need to start our analysis.

This file consists of twenty-two columns, a mix of data types.

Description of the columns:

- Mapped_veh_id: This column is the mapped vehicle ID, which is a unique identifier for each vehicle in the dataset. This identifier is used to link data from different sensors on the same vehicle.
- lat : This column represents the latitude coordinates of the vehicle at the time of the measurement.
- lon: This column represents the longitude coordinates of the vehicle at the time of the measurement.
- RS_E_InAirTemp_PC1: This column contains the Engine Air Temperature (IAT) reading from the first pressure sensor (PC1). IAT measures the temperature of the air entering the engine.
- RS_E_InAirTemp_PC2: This column contains the IAT reading from the second pressure sensor (PC2).
- RS_E_OilPress_PC1: This column contains the engine oil pressure reading from the first pressure sensor (PC1). Oil pressure is important for lubricating and cooling the engine. A low oil pressure reading can indicate a problem with the oil.
- RS_E_OilPress_PC2: This column contains the oil pressure reading from the second pressure sensor (PC2).
- RS_E_RPM_PC1: This column contains the engine RPM (revolutions per minute) reading from the first pressure sensor (PC1). RPM is a measure of engine speed.
- RS_E_RPM_PC2: This column contains the RPM reading from the second pressure sensor (PC2).
- RS_E_WatTemp_PC1: This column contains the water temperature reading from the first pressure sensor (PC1). Water temperature is important for cooling the engine and preventing overheating.

- RS_E_WatTemp_PC2: This column contains the water temperature reading from the second pressure sensor (PC2).
- RS_T_OilTemp_PC1: This column contains the engine oil temperature reading from the first temperature sensor (T). Oil temperature is an important factor in determining engine performance and oil viscosity.
- RS_T_OilTemp_PC2: This column contains the oil temperature reading from the second temperature sensor (T).

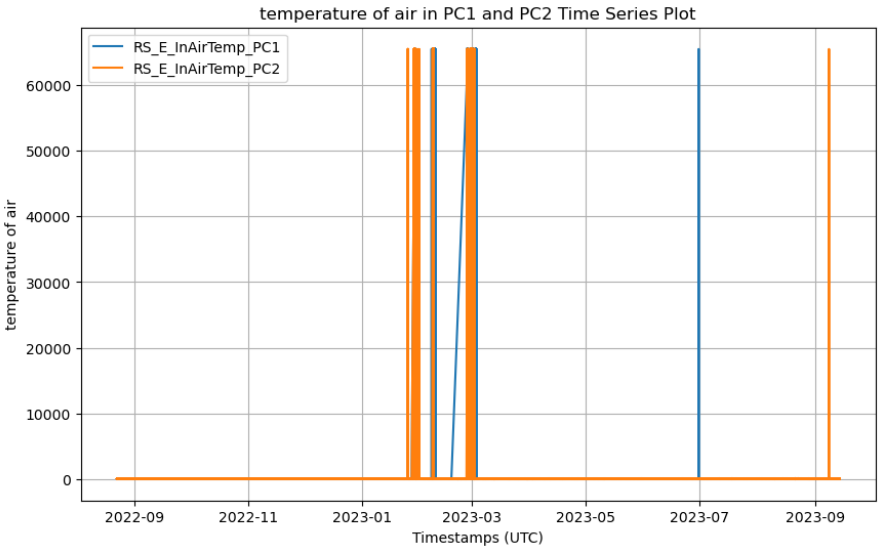
Statistical description:

The description of the dataset was made using python. All the variables were quantitative, hence we presented the mean and the standard deviation. The number of observations and missing values in each variable were also presented.

Variables	N	Mean (SD)	Missing Values
V1	17679273	8.84*10^6 (5.10*10^6)	0
Latitude	17679273	5.09*10^1(3.13*10^1)	0
Longitude	17679273	4.23*10^0 (5.98*10^1)	0
Air temperature in PC1	17679273	3.20*10^1 (3.28*10^2)	0
Air temperature in PC2	17666547	3.23*10^1 (3.48*10^2)	12726
Oil pressure in PC1	17679273	2.64*10^2 (1.15*10^2)	0
Oil pressure in PC2	17666547	2.71*10^2 (1.16*10^2)	12726
RPM PC1	17679273	9.12*10^2 (3.83*10^2)	0
RPM PC2	17666547	9.08*10^2 (3.88*10^2)	12726
Water temperature in PC1	17679273	7.69*10^1 (1.37*10^1)	0
Water temperature in PC2	17666547	7.61*10^1 (1.45*10^1)	12726
Oil temperature in PC1	17679273	7.65*10^1 (1.45*10^1)	0
Oil temperature in PC2	17666547	7.62*10^1 (1.54*10^1)	12726

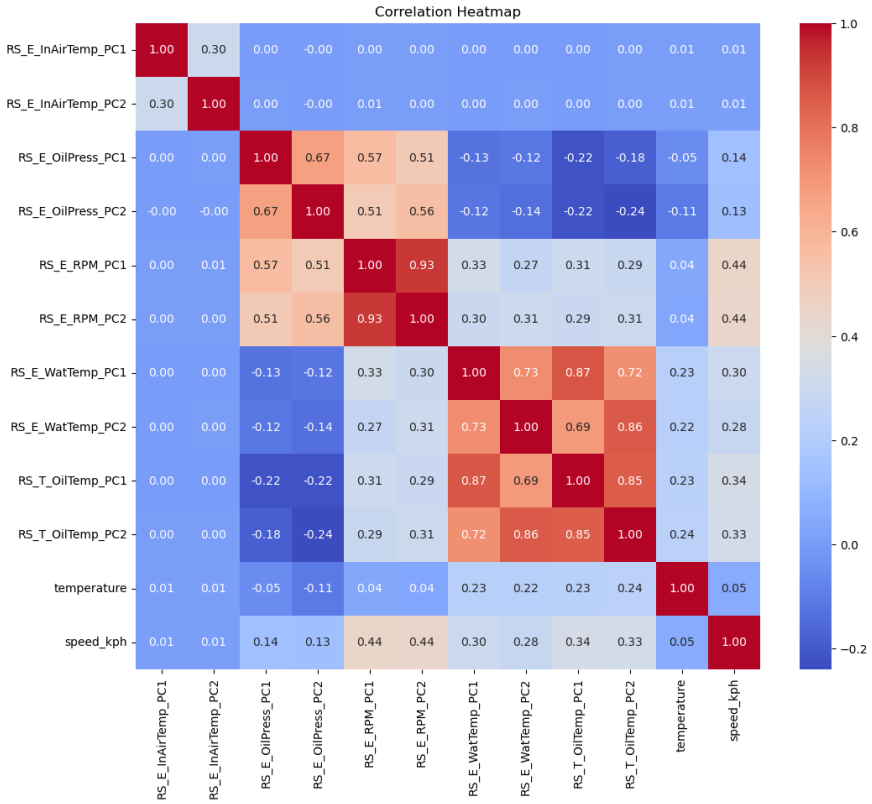
We can see that the missing values are more common for observations related to PC2. Compared to all the observations in the dataset, the proportion of missing values is not that much.

We made a time series graph to see how some of the variables behave with time. In this case we have the air temperatures in PC1 and PC2 as seen below.



We can see that the temperature of air remains constant at the beginning but in the first semester of 2023, we have big peaks (rise and fall) of temperatures of air in PC1 and PC2. Same in July and September 2023.

A heat man correlation matrix was also used to see how the variables are linked to one another.



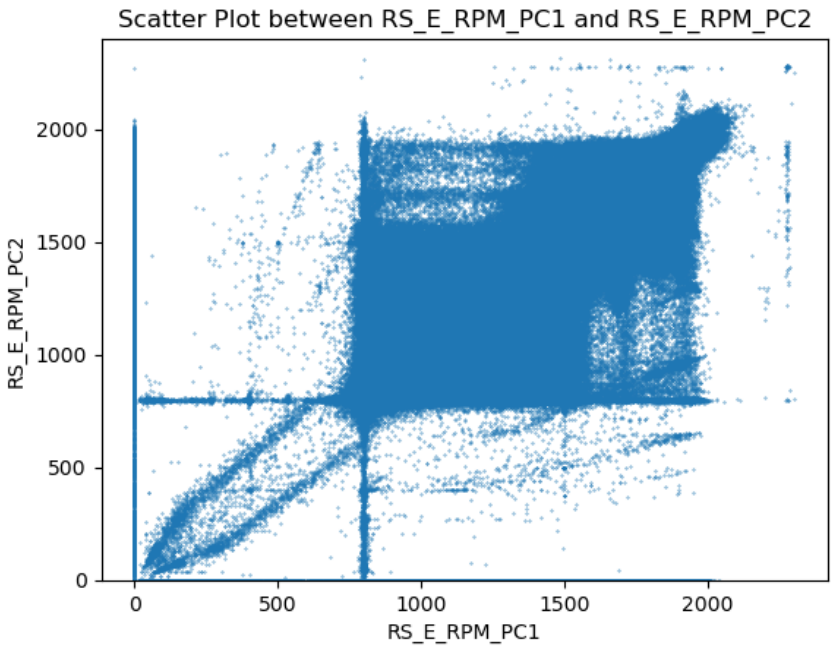
From the matrix we can see that:

#RS_E_OilPress_PC1 and #RS_E_OilPress_PC2 have an inverse correlation with RS_T_OilTemp_PC1 and RS_T_OilTemp_PC2, this is likely because the pressure of the oil is higher at cooler temps due to higher viscosity of the oil.

#RS_E_RPM_PC1 and RS_E_RPM_PC2 show a strong positive correlation, which might imply they are related or measure similar aspects, likely because the engines operate in tandem in normal operating conditions.

Oil and water Temperatures also show a strong positive correlation as they are thermally coupled physical systems.

We then made a scatter plot to visualize the correlation between RS_E_RPM_PC1 and RS_E_RPM_PC2.



We can have the confirmation of the strong positive correlation between the 2 variables with this image.

2. Preprocessing the data and unifying the dataset

Here, we will discuss how we decided to preprocess our data and which other sources we incorporate into it. To verify and test our hypothesis, we need to derive some new variables from our large dataset. This needs to be performed under a specifics logic we will explain in the subsection.

2.1. Working with noisy data

Verifying our assumptions involves being able to check the value of a sample train and confirm our derivative by looking at the value and the expected ones. To do this, our first actions was to sort the dataset by vehicle (mapped_veh_id and timestamps_UTC). This sort allows us to have a somewhat time-series representation of each train.

As we are working with time-series data, we need to be able to get the data from specifics columns the moment before. To achieve this, we used some lag functions[1]. The first-order lag function was perfectly suited for this task. By using them, we were able to calculate the time-elapsed between two measurements of the timestamps_UTC column. This allows us to derive the time-elapsed column.

- The first business rule

Our first "business" rule consists of dropping rows for which the number of 0 is more than 40% in a specific column. Our goal is to avoid extra computation that would create approximation in our model. We decided to implement this rule based on our assumption that a sensor which sends many 0 indicates that there is an issue with the sensor at that moment.

- The second business rule

Our second business rule consists of dropping row groups for which a column is taking more than 20% of NaN values. Our purpose is to decrease the need of computation. If a train has more than 20% of NaN value in a column on a specific date, trying to approximate the potential value will lead to misleading results and then, it can create false positive. We prefer to avoid this risk by dropping the row group from the dataset.

[1] A lag function is a statistic that calculates the value of a variable at a previous point in the data.

Implementing the first 2 rules in order resulted in a modest reduction, reducing the row count from 17.679.273 to 16.805.603.

- ❖ The third business rule

Finally, during our exploratory analysis having examined the time-elapsed between two rows, we determined that there are many instances of near duplicate rows of data occurring in a short time interval. We noticed the time-interval between two rows can vary between less than 20 seconds to more than 50 seconds. However, during the interview with SNCB made by our classmate, we noticed that the normal time-elapsed for the sensor is 30 seconds. This helps us to define our first "Business Rules" as to drop every row for which the time-elapsed is less than 20 seconds. The application of this 3rd and final rule helps us to decrease duplicate, redundant data, row count reduction from 16.805.603 to 10.369.748.

2.2. Enrichment data

To integrate weather data, we utilized the Meteostat Python library, which provides access to open weather and climate data through Pandas. Our objective was to find an effective method for accurately approximating weather data that is not computationally expensive at the same time.

For each geographical point, we identified the nearest weather station, so we approximated each of them with 27 weather stations located in Belgium, France, Netherlands and Germany. Additionally, we rounded the time to facilitate working with hourly data. Subsequently, we downloaded historical hourly weather data for all 27 stations to our computer, stored in .csv files covering the required time period. Finally, by binning the rows location to the nearest weather station, we approximated the weather for all points based on the hourly data obtained from their nearest weather stations.

Also, we created a separate dataset containing all the known SNCB workshop (12) with their specialty types, city of location and coordinates extracted from SNCB website.

	ID	Type	City	Place_Name	Company	Speciality	Coordinates	Latitude	Longitude
0	1	Traction	Charleroi	Atelier de traction de Charleroi	SNCB	locomotives & railcars	50.400, 4.458	50.400	4.458
1	2	Traction	Kinkempois	Atelier de traction de Kinkempois	SNCB	locomotives & railcars	50.609, 5.583	50.609	5.583
2	3	Traction	Hasselt	Atelier de traction d'Hasselt	SNCB	locomotives & railcars	50.937, 5.305	50.937	5.305
3	4	Traction	Arlon	Atelier de traction d'Arlon	SNCB	locomotives & railcars	49.678, 5.817	49.678	5.817
4	5	Traction	Anvers	Atelier de traction d'Anvers-Nord	SNCB	locomotives & railcars	51.294, 4.383	51.294	4.383
5	6	Traction	Schaerbeek	Atelier de Traction de Schaerbeek	SNCB	locomotives & railcars	50.878, 4.379	50.878	4.379
6	7	Traction	Ostende	Atelier de traction d'Ostende	SNCB	locomotives & railcars	51.216, 2.946	51.216	2.946
7	8	Central	Cuesmes	Atelier central de Cuesmes	SNCB	passenger equipment	50.446, 3.933	50.446	3.933
8	9	Central	Malines	Atelier central de Malines	SNCB	passenger equipment	51.057, 4.292	51.057	4.292
9	10	Central	Salzinnes	Atelier central de Salzinnes	SNCB	passenger equipment	50.468, 4.844	50.468	4.844
10	11	Polyvalent	Melle	Atelier polyvalent de Melle	SNCB	passenger equipment & wagons	51.013, 3.779	51.013	3.779
11	12	TGV	Forest	Atelier TGV de Forest	SNCB	TGV	50.822, 4.318	50.822	4.318

Figure 2, SNCB workshop list.

3. Refine the data and cluster the dataset

After adding the data weather in our dataset, we decided to derivate more columns:
We decided to use a combination of python script and RapidMiner to augment the dataset.

3.1. Deriving more columns

By taking the GPS longitude and latitude, we were able to generate elevation data using the API provided by opentopodata.org, however the public API is restricted to 1 request per second and was insufficient to process the 17+ million records in a timely manner. Fortunately, they also provide an open-source implementation of their server through GitHub, both in source code and in a docker container. Using an Ubuntu Linux VM we were able to deploy and configure a localized API server capable of processing approx. 1,200 records per second. Using the etopo1 provided by the US National Centers for Environmental Information. The data provides elevation data rounded to the nearest meter. A separate python program was written to import the original dataset, extract GPS data, submit over https to the local API server, extract the JSON output, parse the result, and to populate the dataset with new data.

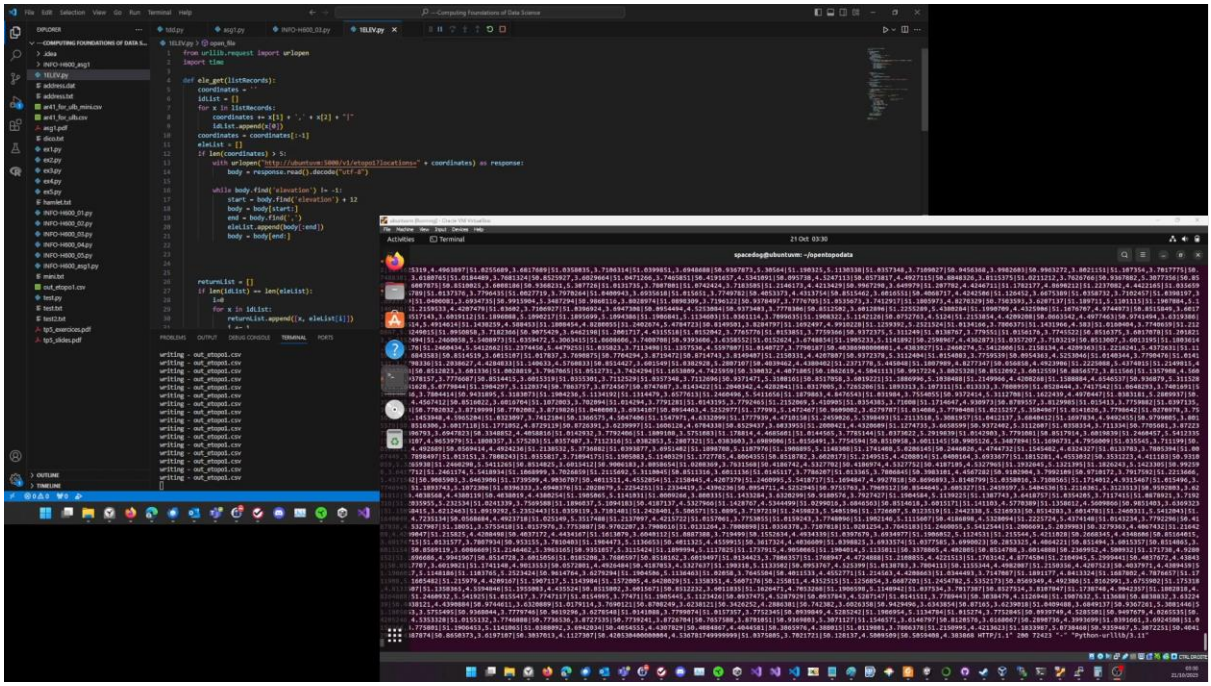


Figure 3, opentopodata.org server running on a local VM, being called by python to populate a pandas dataframe.

Next, we developed a translation from GPS longitude and latitude to Cartesian coordinates in meters, this was a more challenging problem than we thought as it's not a simple scale conversion, nor is it a problem solved by straight forward geometry. The problem is that the earth is not necessarily a sphere. A nearer approximation would be ellipsoid, yet even then the earth's surface is still more irregular than this shape. The problem is complicated further by various countries adopting various CRS (Coordinate Reference System) definitions/standards of what shape of ellipsoid best fits the earth in such a way that it yields the most accurate local results. E.g. The US selects a standard that suits their land mass, while that same standard when applied to the terrain of Australia returns poor accuracy.

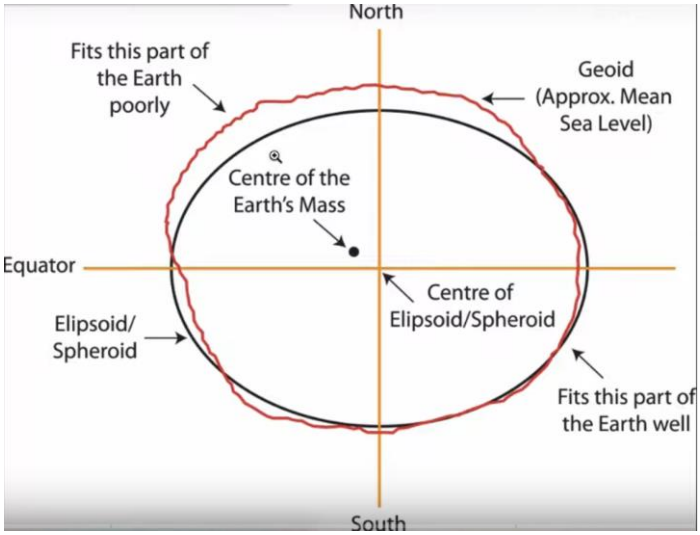


Figure 4, A Coordinate Reference Ellipsoid approximation of the earth does not suit all regions equally.

So, selecting a CRS is critical to retrieving the most accurate results. We opted for a 2 step process, first we use **EPSG:4326,** to translate GPS to a generalized Coordinate Reference, and then use **EPSG:31370** to refine this generalized reference to a localized on the Belgium. EPSG:31370 was selected by searching the epsg.io (https://epsg.io/31370), and verified by comparing to 4 other localized CRS, all of which returned distances withing a margin of .001% for all non zero values in the database.

accuracy observation between coordinate systems.ods - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

	distout_EPSG_31370	distout_EPSG_31300	distout_EPSG_3812	distout_EPSG_3447	max	min	difference	"0.001%"	is the difference smaller than .001%
2707066									
3686713	2.47149572237142	2.47149572218167	2.47147732745678	2.47147732815347	2.47149572237142	2.47147732745678	0.00002	0.00024715	TRUE
2715602	3.11687705786825	3.11687705737892	3.11686435262984	3.116864353909	3.11687705786825	3.116864353909	0.00001	0.00031169	TRUE
7063665	47.7656010440385	47.7656010498332	47.765242661471 4	47.765242661916 9	47.7656010498332	47.765242661471 4	0.00036	0.00477656	TRUE
12721095	48.9292092291095	48.9292092355515	48.9288408285172	48.9288408289777	48.9292092355515	48.9288408285172	0.00037	0.00489292	TRUE
12089058	46.5314579219414	46.5314579274694	46.5311073713182	46.5311073710922	46.5314579274694	46.5311073710922	0.00035	0.00465315	TRUE
706083	46.0853269126205	46.0853269182757	46.0849726477 13	46.0849726474494	46.0853269182757	46.0849726474494	0.00035	0.00460853	TRUE
14738580	46.6626769923181	46.6626769986632	46.6623237779148	46.6623237776847	46.6626769986632	46.6623237776847	0.00035	0.00466627	TRUE
12877015	46.9471222898243	46.9471222953083	46.9467788123604	46.9467788121516	46.9471222953083	46.9467788121516	0.00034	0.00469471	TRUE
7063664	46.3328552426352	46.3328552476977	46.332517781634	46.3325177818352	46.3328552476977	46.332517781634	0.00034	0.00463329	TRUE
7219929	45.1087210074408	45.108721012553	45.108383500352	45.108383500793	45.108721012553	45.108383500352	0.00034	0.00451087	TRUE
1245281	45.956364001033	45.9563640069672	45.9560129998967	45.9560130001435	45.9563640069672	45.9560129998967	0.00035	0.00459564	TRUE
2711315	0.833145450225768	0.833145450381127	0.833140491905912	0.833140491886913	0.833145450381127	0.833140491886913	0.00000	0.00008331	TRUE
15011575	0.662157209539194	0.662157210449549	0.662154736387798	0.662154738111749	0.662157210449549	0.662154736387798	0.00000	0.00006622	TRUE
10342793	0.448681621294783	0.448681621296939	0.448678891880238	0.448678891821251	0.448681621296939	0.448678891821251	0.00000	0.00004487	TRUE
1253760	1.23651391211055	1.23651391 14882	1.23650545 16379	1.23650545233687	1.23651391211055	1.23650545 16379	0.00001	0.00012365	TRUE
706081	0.659627626309781	0.659627625903944	0.659622088701358	0.659622089675644	0.659627626309781	0.659622088701358	0.00001	0.00006596	TRUE
8270722	1.45975011369523	1.45975011453195	1.45973672826468	1.45973672880701	1.45975011453195	1.45973672826468	0.00001	0.00014598	TRUE
9360563	0.748096255973543	0.748096257 0584	0.748090204539 84	0.748090203720225	0.748096257 0584	0.748090203720225	0.00001	0.00007481	TRUE
10342792	1.5986351064053	1.59863510438956	1.5986251864304	1.5986251892539	1.5986351064053	1.5986251864304	0.00001	0.00015986	TRUE
8981770	43.7309331190761	43.7309331248784	43.7306024908084	43.730602489763	43.7309331248784	43.730602489763	0.00033	0.00437309	TRUE
1262111	43.5534150521834	43.5534150576801	43.5530928076533	43.5530928074599	43.5534150576801	43.5530928074599	0.00032	0.00435534	TRUE
16106575	43.8863569123241	43.8863569174655	43.8860367707043	43.8860367708866	43.8863569174655	43.8860367707043	0.00032	0.00438864	TRUE
10178403	44.0437538797419	44.0437538851738	44.043423163744	44.043423163744	44.0437538851738	44.043423163744	0.00033	0.00440438	TRUE
14998834	46.6329425129987	46.6329425192511	46.632586413785	46.6325864137946	46.6329425192511	46.632586413785	0.00036	0.00466329	TRUE
9360566	50.9334794461653	50.9334794528441	50.9330965536783	50.9330965536872	50.9334794528441	50.9330965536783	0.00038	0.00509335	TRUE
1997081	43.9635943866116	43.9635943920916	43.9632640402191	43.9632640400126	43.9635943920916	43.9632640400126	0.00033	0.00439636	TRUE
12716735	59.9351024450454	59.9351024523307	59.9346541351688	59.9346541349927	59.9351024523307	59.9346541349927	0.00045	0.00599351	TRUE
8266541	44.7111309854729	44.711130991264	44.7107943693159	44.7107943695811	44.711130991264	44.7107943693159	0.00034	0.00447111	TRUE
701840	59.7190358721141	59.7190358792717	59.7185908677963	59.7185908677963	59.7190358792717	59.7185908677963	0.00045	0.00597190	TRUE
10174022	45.0876791277949	45.0876791329448	45.0873409232793	45.0873409239606	45.0876791329448	45.0873409232793	0.00034	0.00450877	TRUE
8060948	58.9043029699021	58.9043027055988	58.9038662038037	58.9038662048536	58.9043027055988	58.9038662038037	0.00044	0.00589043	TRUE
8270723	44.5210024929898	44.5210024978179	44.5206688760817	44.5206688765011	44.5210024978179	44.5206688760817	0.00033	0.00445210	TRUE
15973620	56.3283955274384	56.3283955336099	56.3279798992034	56.327979899212	56.3283955336099	56.3279798992034	0.00042	0.00563284	TRUE
3682407	46.1173428536298	46.1173428584979	46.1170018692845	46.1170018688666	46.1170018688666	46.1170018688666	0.00034	0.00461173	TRUE
7046381	69.7912653079614	69.7912653158934	69.7907601667772	69.7907601665602	69.7912653158934	69.7907601665602	0.00051	0.00697913	TRUE
11914329	42.1072431515915	42.1072431560733	42.1069389655365	42.1069389653189	42.1072431560733	42.1069389653189	0.00030	0.00421072	TRUE
3690945	107.140977245991	107.140977258033	107.140238357607	107.140238357427	107.140977258033	107.140238357427	0.00074	0.01071410	TRUE
7228366	40.7864686175104	40.7864686216018	40.7861902972615	40.7861902972486	40.7864686216018	40.7861902972486	0.00028	0.00407865	TRUE

Figure 5, Contrasting various Coordinate Reference Standards for accuracy (< 0.001%).

CRS implementation was done using the **geopandas** python extension collection, and returned Cartesian coordinates, used to accurately derive further information.

Distance: using **Pythagoras Theorm** $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ using previous (filtered) record for x1, y1 and current row for x2, y2, output in meters.

Expression

```
sqrt(pow(([x-1]-x),2)+pow(([y-1]-y),2))
```

Speed: using $speed = distance/time$ using difference between pervious row and current to calculate time interval. Output in KPH

Expression

```
((distance / date_diff([timestamps_UTC-1],timestamps_UTC,DATE_UNIT_SECOND,"Europe/Brussels"))/1000)*3600
```

Acceleration: using the definition of acceleration:

$a = (v_f - v_i) / \Delta t$;

- **a** – Acceleration;
- v_i and v_f are, respectively, the initial and final velocities;
- Δt – Acceleration time;

Using previous and current speed and time for Vi Vt and Delta-t

Slope: calculated as $m = \frac{y_2 - y_1}{x_2 - x_1}$ where x is time. Idea being to calculate the change of elevation over time. Elevation y is retrieved from 1 record before and 1 after the current row. Returns an angle in degrees, usually a small value as train tracks are generally built to a gradual incline.

Expression

```
([etopo1_elevation+1] - [etopo1_elevation-1]) /
date_diff([timestamps_UTC-1],[timestamps_UTC+1],DATE_UNIT_SECOND,"Europe/Brussels")
```

Direction Angle: Calculated using the **atan2** trigonometric function:

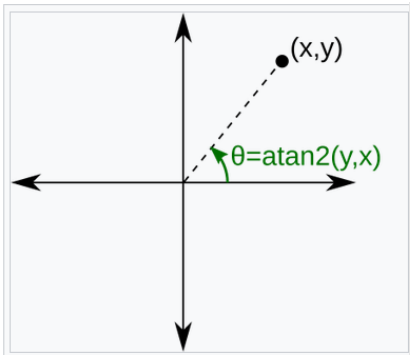


Figure 6, atan2 converts a point x, y to an angle Theta.

It returns and angle Theta moving counter clockwise from the positive x-axis, indicating the direction of movement of the train between the previous and next rows, after offset and translation: 0-360 degrees where 0:West, 90:South, 180:East, 270:North

Expression

```
1 (((atan2([x-1]-[x+1],[y-1]-[y+1]))+PI)/(2*PI)*360
```

We also generated smoothed columns of elevation, slope and acceleration using the **moving average** technique, sampling 2 rows before, and 2 after. We tried many parameters in this smoothing but found diminishing returns of detail when sampling with stringer settings than this due to the time spacing of the data, a train can easily stop, disembark passengers, load passengers and be on the move again in the 5 rows spanning the smoothed sampling. For this reason, the source columns are also retained for reference/comparison to the original hard data when running anomaly detection later.

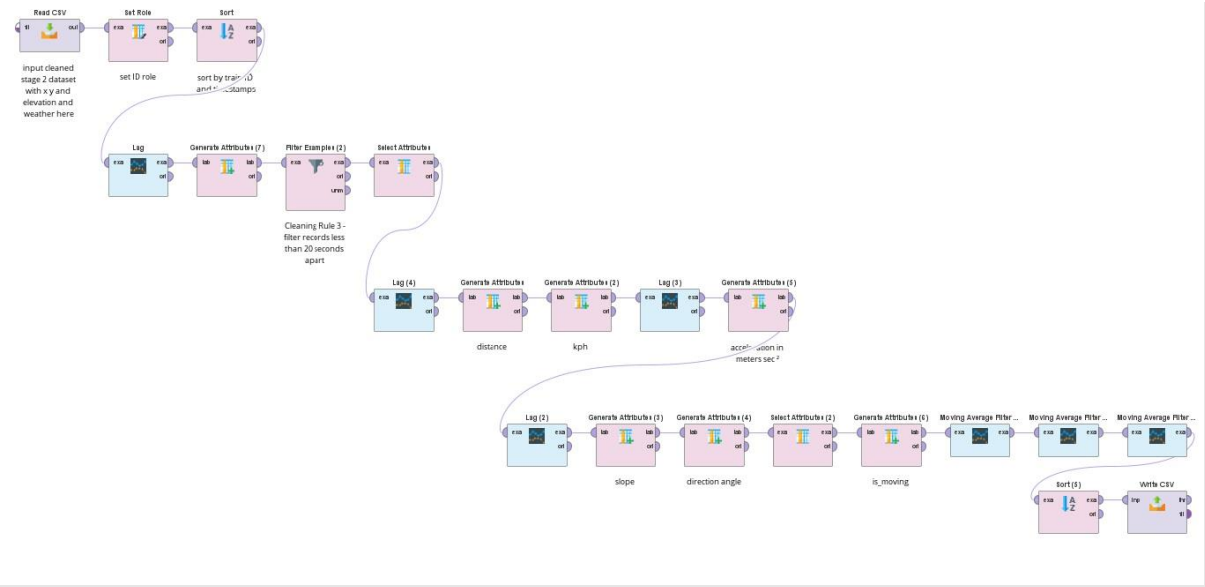


Figure 6, Rapidminer process for final cleaning and derived columns.

3.2. List of columns added to the dataset

etopo1_elevation	elevation from GPS rounded to nearest 1 meter.
x y	Cartesian coordinates calculated from GPS in meters.
station	nearest weather station.
round_time	time rounded to the hour.
temperature	External temperature.
rel_hum	Humidity.
precipitation	Precipitation.
pressure	Air pressure.
distance	Distance calculated from the previous record for this vehicle.
speed_kph	Speed.
acceleration	Acceleration.
slope	Estimated slope (typically a small number as train tracks are generally designed to be on level terrain).
angle	Direction the train is moving 0-360 degrees where 0:West, 90:South, 180:East, 270:North.
is_moving	It's a Boolean measure to know if the train moving. It's estimated if oil pressure of either engine exceeds 900.
etopo1_elevation_filtered_ma_2_2	Elevation smoothed using moving average 2 before, 2 after.
acceleration_filtered_ma_2_2	Acceleration smoothed using moving average 2 before, 2 after.
slope_filtered_ma_5_5	Slope smoothed using moving average 2 before, 2 after.

4. Anomalies reporting

To determine the anomalies present in each of the variables of the data set, two methods were used. The first method is a statistical model known as exponential smoothing and the second is a Machine learning model called One Class SVM.

4.1 Exponential Smoothing

The process begins by selecting a specific variable from the dataset, for instance, 'RS_E_InAirTemp_PC1', for anomaly detection. The goal is to identify irregularities or deviations from the expected behavior of this variable.

Variable Selection: Choose the variable of interest, such as temperature measurements ('RS_E_InAirTemp_PC1'), for anomaly detection.

Exponential Smoothing: Apply exponential smoothing to the selected variable. This involves assigning exponentially decreasing weights to past observations to create a smoothed version of the variable.

Residual Calculation: Calculate residuals by finding the difference between the original values of the variable and their corresponding smoothed values. Residuals indicate how much the actual data deviates from the smoothed trend.

Statistical Analysis: Compute the mean and standard deviation of the residuals. These statistical measures help establish a threshold for identifying anomalies.

Threshold Setting: Define a threshold for anomaly detection based on the standard deviations of the residuals. Typically, anomalies are detected when the absolute value of the residual exceeds a certain threshold (e.g., set as three times the standard deviation).

Anomaly Detection: Identify anomalies by comparing the absolute values of the residuals against the threshold. Instances where the residuals exceed this threshold are flagged as potential anomalies.

Visualization: Generate visualizations that illustrate the original variable, its smoothed version, and mark potential anomalies. This visual representation aids in observing deviations and irregularities in the dataset.

Anomaly Display: Display the detected anomalies, including their timestamps, original variable values, corresponding smoothed values, and the calculated residuals. This information helps in understanding the specific points in time where anomalies occur and their magnitude.

This process utilizes exponential smoothing to create a smoothed version of the variable and subsequently compares it with the original data to detect potential anomalies. The residuals, calculated as the differences between actual and smoothed values, serve as indicators of deviations from the expected trend, helping identify irregular patterns or outliers in the dataset.

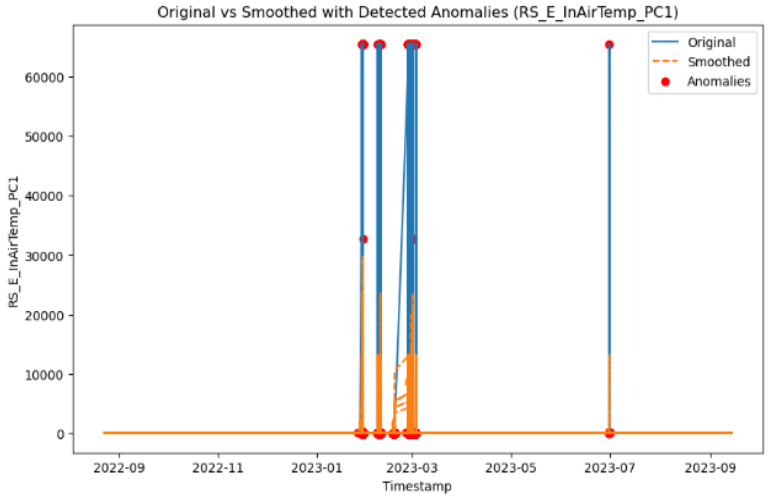
4.2 One class SVM

Column Selection: Choose relevant columns from the dataset that are essential for anomaly detection, such as temperature, pressure, humidity, or other pertinent variables.
Data Extraction: Extract the selected columns from the dataset to create a focused subset of data specifically for anomaly detection purposes.
Handling Missing Values: Address missing values within the selected columns. Common approaches include imputation, where missing values are replaced by a measure such as the mean or median of the column.
Standardization (Scaling): Standardize the dataset by scaling the values. This step ensures that all variables have a comparable scale, typically with a mean of 0 and a standard deviation of 1. Standardization facilitates better performance of the anomaly detection model.
One-Class SVM Model Creation: Create a One-Class SVM model. This specific type of SVM is trained solely on 'normal' data to learn its characteristics and boundary. It is designed to identify instances that deviate significantly from this learned 'normal' behavior.
Anomaly Prediction: Employ the trained One-Class SVM model to predict anomalies in the dataset. The model assigns labels to each data point: 1 for normal instances and -1 for potential anomalies. Instances deviating significantly from the learned 'normal' behavior are labeled as anomalies.
Anomaly Counting: Count the number of instances labeled as anomalies by the One-Class SVM model. This count represents the estimated number of abnormal occurrences or outliers within the dataset.
One-Class SVM, in this context, serves as a machine learning technique tailored for outlier detection in datasets where normal data vastly outweighs abnormal instances. By learning and identifying patterns of normal behavior, the model effectively pinpoints data points that significantly differ from this norm, flagging them as potential anomalies.

4.3 Results

We presented plots for the Exponential smoothing for each variable.

**Anomalies Detection model for air variable 1*

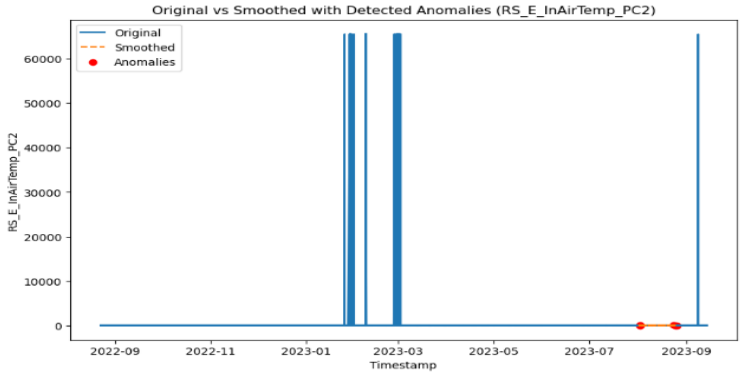


The plot illustrates the residuals over time for the variable RS_E_InAirTemp_PC1. On the x-axis, you'll find the timestamps indicating when the measurements were taken. The y-axis represents the residuals, showcasing the differences between the actual RS_E_InAirTemp_PC1 data and its smoothed values.

Positive residuals, seen above the zero line, signify moments where the actual values exceeded the expected smoothed values. Conversely, negative residuals, below the zero line, indicate instances where the actual values fell below the anticipated smoothed data.

Anomalies appear as significant spikes or drops in the plot. These deviations highlight instances where the observed RS_E_InAirTemp_PC1 values notably deviated from the anticipated smoothed values. These peaks or dips may signal irregularities or outliers in the temperature data, indicating potential anomalies.

**Anomalie Detection model variable 2*

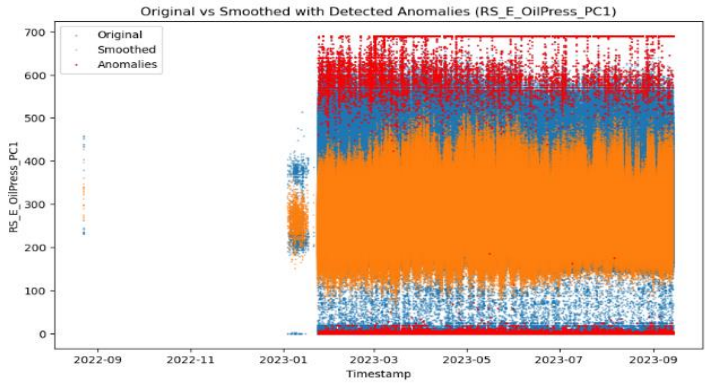


In the plotted graph, we see a time series representation, with timestamps on the x-axis and temperature residual values on the y-axis. The RS_E_InAirTemp_PC2 line would illustrate the

actual recorded temperatures. Simultaneously, the Smoothed_RS_E_InAirTemp_PC2 line would represent the expected or smoothed temperatures.

Anomalies would be visualized as notable spikes or dips in the Residuals line, showcasing the difference between actual and expected temperatures. These peaks or drops at specific timestamps indicate instances where recorded temperatures significantly differ from the anticipated values, signaling potential anomalies or irregularities in the RS_E_InAirTemp_PC2 data.

**Anomalie Detection model variable 3*



In this context, the plotted graph would likely represent a time series where the x-axis contains timestamps, and the y-axis illustrates the oil pressure values (RS_E_OilPress_PC1). The line representing Smoothed_RS_E_OilPress_PC1 would depict the expected or smoothed oil pressure readings.

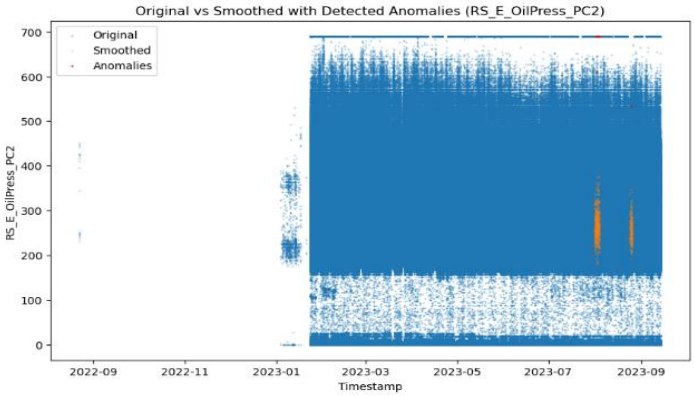
Magnitude of Anomalies: Anomalies are primarily observed with notably high or low values in RS_E_OilPress_PC1 compared to their corresponding smoothed values (Smoothed_RS_E_OilPress_PC1). This discrepancy results in considerable positive or negative residuals.

Variability over Time: The anomalies are distributed across various timestamps, suggesting that these deviations from the expected values occurred sporadically throughout the dataset's time range.

Outliers: There are several instances where the residuals are significantly high or low, suggesting abrupt changes or extreme fluctuations in the oil pressure, potentially indicative of equipment malfunctions, measurement errors, or irregular system behavior.

The recurring theme of large deviations from the smoothed values signifies potential critical variations in the oil pressure data, warranting further investigation into the underlying causes behind these irregularities.

**Anomalie Detection model variable 4*



Looking at the anomalies detected for RS_E_OilPress_PC2:

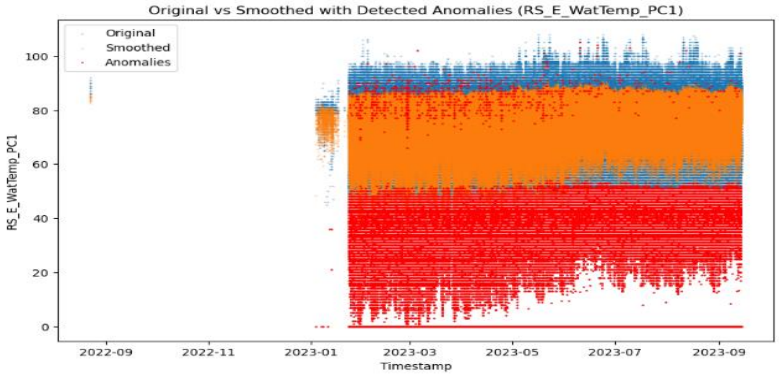
Low Values: There are multiple instances where the observed values are at the lower limit (3.0) compared to their smoothed counterparts. This consistent low reading denotes potential irregularities or sensor issues.

Magnitude of Residuals: The residuals show a consistent negative pattern, indicating a significant discrepancy between the actual values and the smoothed values. This discrepancy implies an underperformance or deviation in the pressure readings, suggesting possible anomalies or issues in the oil pressure data collection for these instances.

Intermittent High Residuals: There are also anomalies with considerably high positive residuals, suggesting sudden spikes or deviations in the oil pressure values compared to the smoothed data.

Overall, these anomalies suggest irregularities in the oil pressure sensor readings, either due to technical glitches, sensor malfunctions, or irregular fluctuations in the system, both in instances of extremely low values and sporadic high fluctuations.

**Anomalie Detection model variable 5*



For the RS_E_WatTemp_PC1 anomalies:

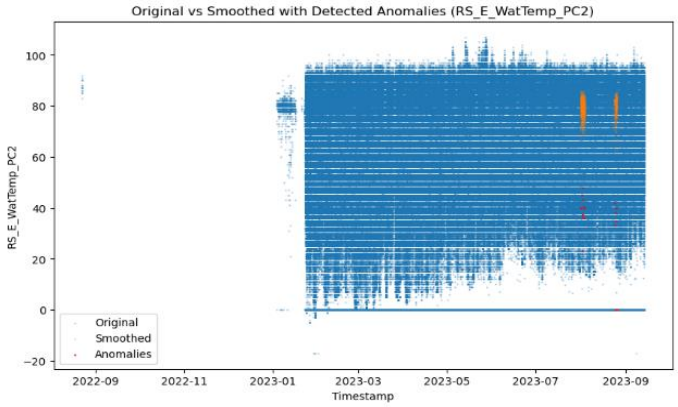
Consistent Low Values: Multiple instances exhibit markedly lower values compared to their smoothed counterparts. This consistent pattern of lower readings indicates potential irregularities or malfunctions in the water temperature sensor during these times.

Magnitude of Residuals: The residuals are consistently negative, indicating a significant discrepancy between the observed and smoothed values. This discrepancy reflects an underperformance or deviation in the water temperature readings, suggesting potential anomalies or issues in the sensor data collection.

Occasional Sudden Drops: Some instances show notably steep drops in water temperature readings, leading to significantly higher negative residuals. This suggests abrupt fluctuations or erratic behavior in the water temperature sensor.

Overall, these anomalies suggest potential malfunctions or irregularities in the water temperature sensor readings, with consistent underperformance and occasional sudden drops causing discrepancies from the expected smoothed values.

**Anomalie Detection model variable 6*



For the RS_E_WatTemp_PC2 anomalies:

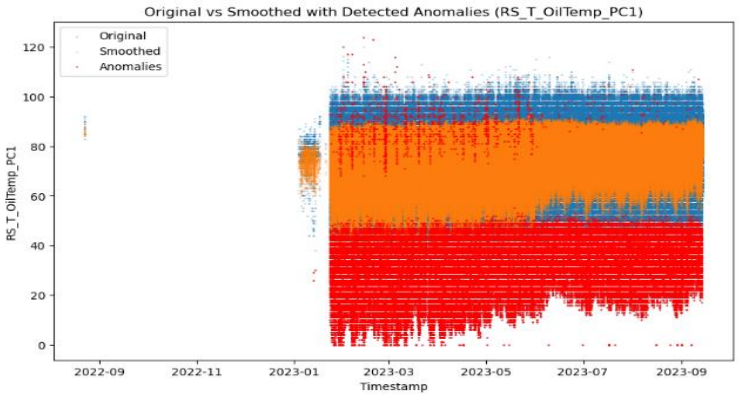
Consistently Low Readings: The observed values consistently fall significantly below the expected smoothed values, suggesting a persistent underperformance or issue with the water temperature sensor for these timestamps.

Magnitude of Residuals: All residuals are negative, indicating a consistent discrepancy between the observed and smoothed values. The negative residuals signify that the observed readings are consistently lower than the expected smoothed values by a significant margin.

Low or No Values: There are instances where the sensor records extremely low or zero values, indicating potential irregularities or sensor malfunctions during these specific timestamps.

Overall, these anomalies point toward an issue or malfunction in the water temperature sensor, consistently reporting lower values than anticipated across various timestamps, with occasional readings close to zero.

**Anomalie Detection model variable 7*



For the RS_T_OilTemp_PC1 anomalies:

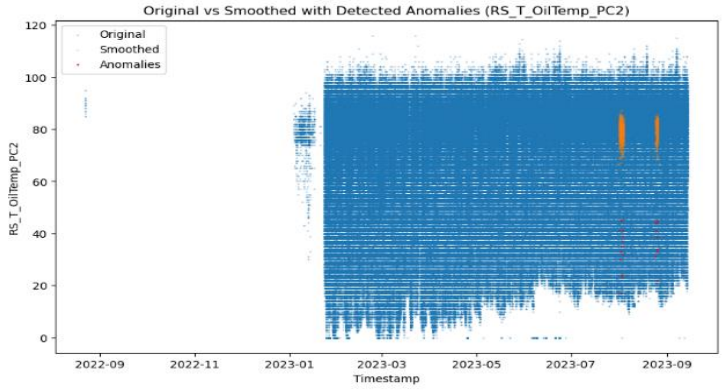
Consistent Temperature Deviations: The observed temperatures consistently fall significantly below the expected smoothed values, suggesting an ongoing issue or sensor malfunction across multiple timestamps.

Magnitude of Residuals: The residuals are consistently negative, indicating that the observed oil temperatures are consistently lower than the expected smoothed values by a significant margin.

Low Temperature Readings: Instances where the observed temperature is notably lower than the expected values might indicate potential issues with the oil temperature sensor, consistently reporting readings much lower than the anticipated values.

In summary, these anomalies suggest an ongoing problem with the oil temperature sensor, consistently reporting lower-than-expected values across various timestamps, with some instances showing significantly low readings compared to the anticipated values.

**Anomalie Detection model variable 8*



For the RS_T_OilTemp_PC2 anomalies:

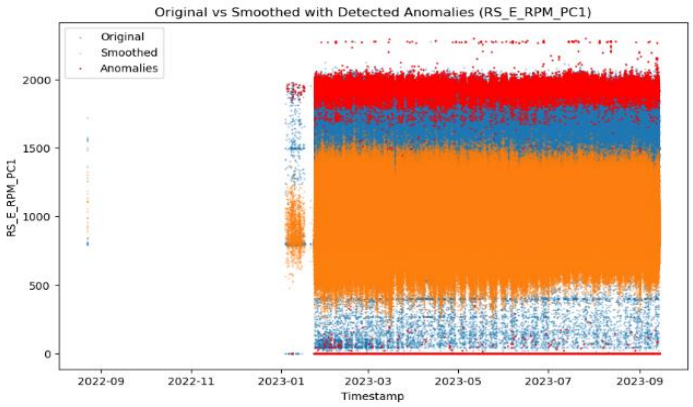
Consistently Lower Temperatures: Across multiple timestamps, the observed oil temperatures consistently fall significantly below the expected smoothed values. These deviations indicate a consistent issue or sensor malfunction affecting the reported values.

Magnitude of Residuals: All residuals are consistently negative, signifying that the observed oil temperatures are consistently lower than anticipated by a significant margin.

Consistent Deviations: The deviations between observed and expected temperatures are consistent across various timestamps, indicating a potential systematic issue with the sensor or anomalies impacting this particular parameter's measurements.

In summary, the anomalies suggest a consistent problem with the oil temperature sensor, consistently reporting significantly lower-than-expected values across multiple timestamps, potentially indicating a sensor malfunction or systemic issue.

**Anomalie Detection model variable 9*



For the RS_E_RPM_PC1 anomalies:

Extreme RPM Values: These anomalies display instances where the reported RPM values deviate significantly from the expected smoothed RPM values. In some cases, these deviations are exceptionally high, even reaching over 1000 RPM difference between observed and expected values.

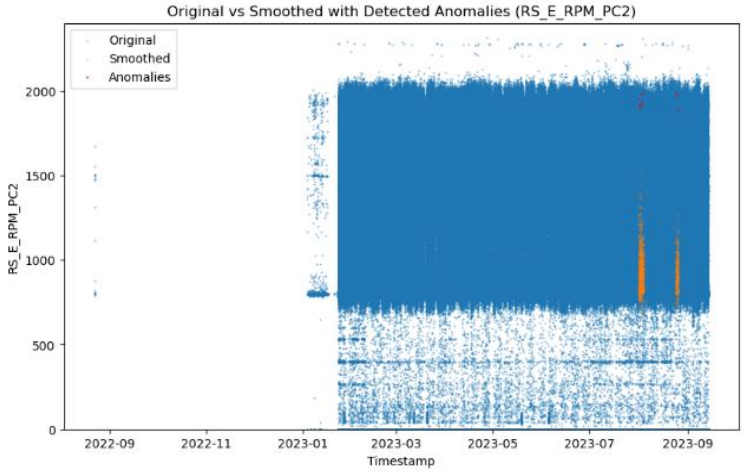
Large Negative Residuals: Most of the anomalies exhibit large negative residuals, indicating that the reported RPM values are substantially lower than the smoothed or expected values. The magnitude of these differences is considerable, indicating potential faults or sensor malfunctions causing RPM underreporting.

Intermittent Anomalies: These anomalies are not consistent across all timestamps, suggesting intermittent issues rather than a continuous problem. Some instances report zero RPM, indicating potential measurement errors or sensor failures during those periods.

In summary, the anomalies in RS_E_RPM_PC1 highlight sporadic but significant discrepancies between observed RPM values and the expected smoothed values. These differences suggest potential sensor faults or intermittent issues in reporting RPM data. The cases with zero RPM readings could signify sensor failures or data acquisition problems during those specific timestamps.

A notable particularity observed in the anomalies of RS_E_RPM_PC1 is the presence of a considerable number of zero RPM readings. Unlike typical anomalies that primarily involve extreme deviations from the expected values, here, a significant subset of anomalies records zero RPM. This specific occurrence of zero RPM suggests potential sensor failures or data acquisition issues, distinct from anomalies involving extreme but non-zero RPM values.

**Anomalie Detection model variable 10*



In the anomalies detected for RS_E_RPM_PC2, the residuals seem to indicate a significant deviation between the actual and smoothed RPM values, specifically with a positive residual, indicating an increase in RPM compared to the smoothed values. This deviation appears notably large in the particular timestamp mentioned, with an RPM value of 2272 compared to the smoothed RPM of 1210.84, resulting in a residual of 1061.16. Such deviations might signify sudden spikes or irregularities in RPM recordings at that timestamp.

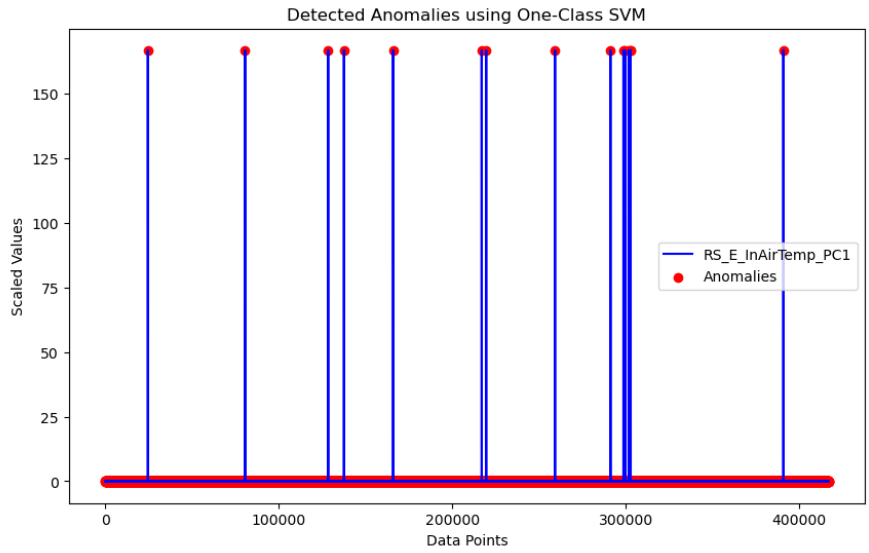
In the anomalies detected for RS_E_RPM_PC2, the residual value for the timestamp 2023-08-09 14:36:05 shows a notable deviation in the RPM values. The residual of 1061.16 suggests a considerable increase in RPM compared to the smoothed values. This substantial difference might indicate an abrupt or irregular spike in RPM at that specific timestamp, which is significantly higher than the expected or smoothed RPM value.

For the results of the One class SVM, due to time constraints and the fact that it takes a longer time to run the codes we just presented the graph for the first variable and a capture of the overall result as seen below.

**Over all anomalies*

Identifies 20,841 potential anomalies within the dataset.

**Anomalies detection for variable 1*



This graph is like the first graph on exponential smoothing with high peaks and drops.

4.4 Comparison of two methods

The results obtained from the provided methods showcase different approaches:

The Exponential Smoothing method applies smoothing techniques to the dataset, generating smoothed values for various parameters such as temperatures, pressures, and RPMs. However, it doesn't explicitly count or identify anomalies, instead offering smoothed data for analysis and trend visualization.

On the other hand, the One-Class SVM method stands out by identifying a significant count of potential anomalies within the dataset. Specifically, it flags 20,841 instances as potentially abnormal based on deviations from learned 'normal' patterns.

In essence, Exponential Smoothing focuses on transforming the data for trend analysis without pinpointing anomalies. It primarily offers smoothed data for visualization and trend analysis. Meanwhile, One-Class SVM explicitly highlights specific data points that significantly deviate from expected 'normal' behavior, providing a targeted identification of potential anomalies.

Ultimately, the choice between these methods depends on the analytical goals: Exponential Smoothing aids in trend understanding, while One-Class SVM excels at explicit anomaly detection based on substantial deviations. Further validation and investigation are essential to confirm the nature and impact of the anomalies highlighted by the One-Class SVM method.

5. Models building

We initiated our model preparation by eliminating explicit outliers, such as zeroes and physically implausible values, as they can significantly impact the final result. Subsequently, we conducted a basic analysis and observed a correlation between column values and temperature. Specifically, the correlation between outside temperature and the mean value of 'RS_E_InAirTemp_PC1,' calculated for each outside temperature value, demonstrated linearity with a slope of 19.6 and an intercept of 0.84

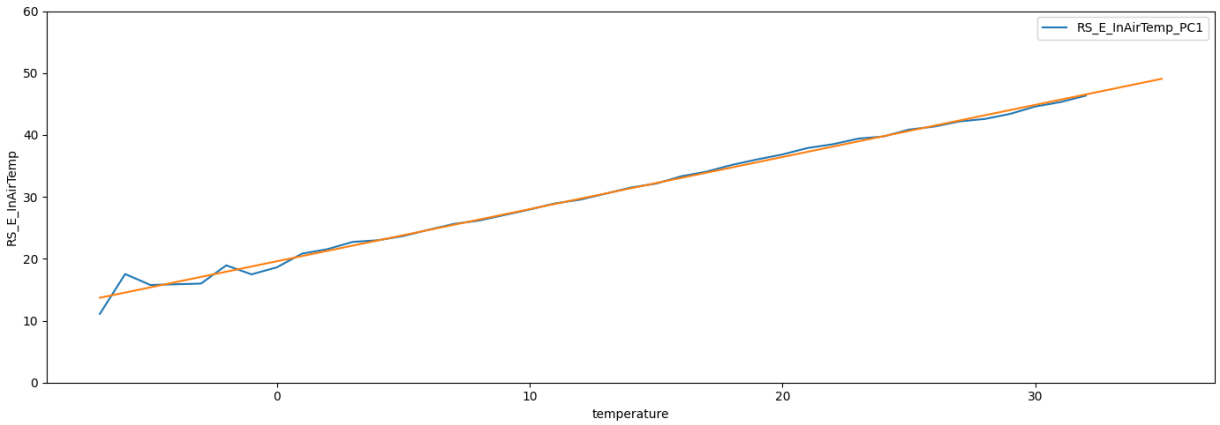


Figure 7, RS_E_InAirTemp graph analysis.

Additionally, we assume that the location of a train can influence variables, because it implicitly considers the proximity to cities, villages and train stations, and natural features such as slope.

Consequently, we removed all lines containing at least one zero or inappropriate value, except for two RPM values (since these can be zero).
Three new columns were added: latitude and longitude rounded to the first decimal place, and outside temperature rounded to the nearest integer. However, due to insufficient data for very low and high temperatures, we decided to categorize all temperatures below -2°C as -2°C and temperatures above 30°C as 30°C.

Subsequently, we grouped the data by rounded temperature, latitude, and longitude, resulting in a new data frame with mean values and standard deviations for various parameters within each cluster, including:

['RS_E_InAirTemp_PC1', 'RS_E_InAirTemp_PC2', 'RS_E_OilPress_PC1', 'RS_E_OilPress_PC2', 'RS_E_RPM_PC1', 'RS_E_RPM_PC2', 'RS_E_WatTemp_PC1', 'RS_E_WatTemp_PC2', 'RS_T_OilTemp_PC1', 'RS_T_OilTemp_PC2'].

Each line in the original dataset was assigned to a specific cluster.

Example, distribution of train parameters for one random cluster, vertical orange lines indicate values 2 standard deviations around the mean value:

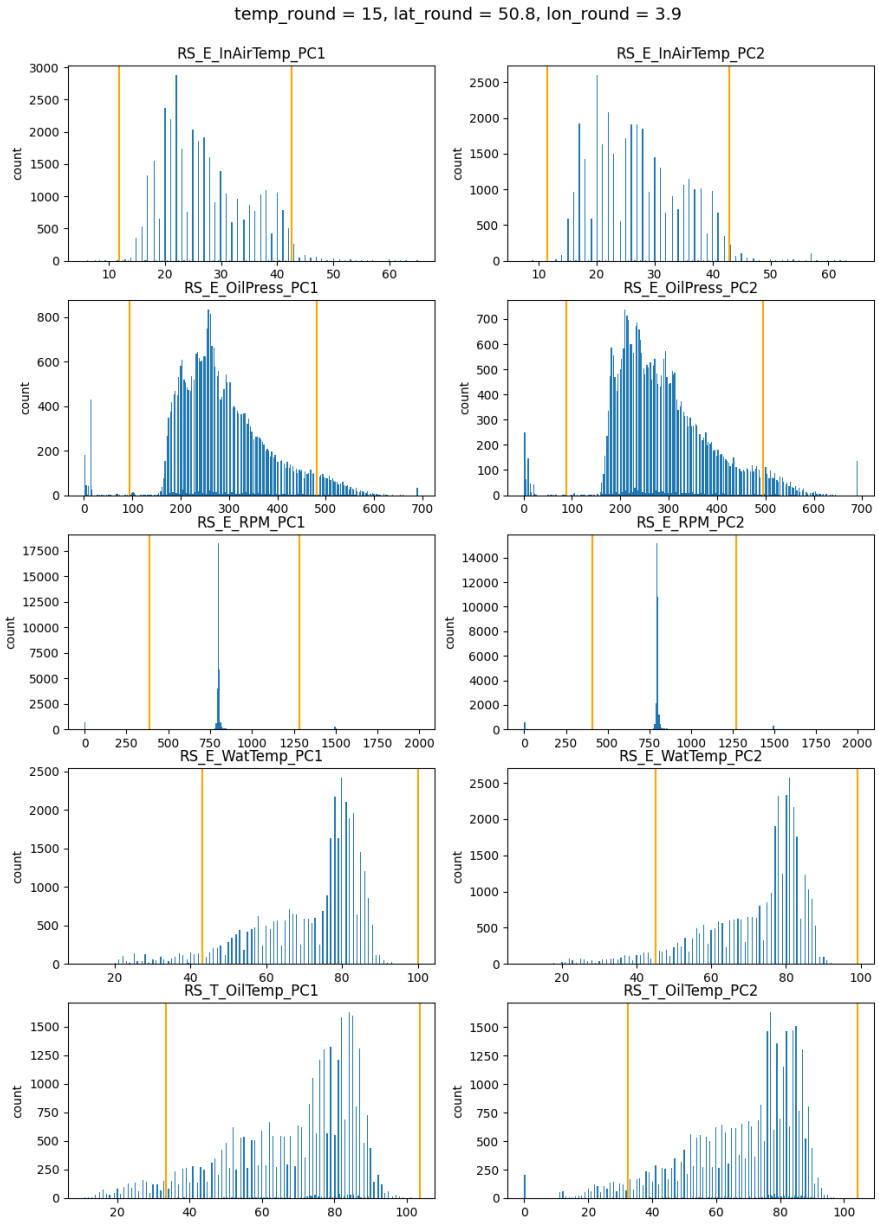


Figure 8, Distribution of train parameters on a random cluster.

Subsequently, we generated a code for each line in the original dataset, indicating potential outliers. If a parameter deviated by more than 2 times the standard deviation, it was marked as a potential anomaly.

The code could contain the following numbers:

-2: no potential outliers,

0: outlier is 'RS_E_InAirTemp_PC1'

1: outlier is 'RS_E_InAirTemp_PC2'

2: outlier is 'RS_E_OilPress_PC1'

3: outlier is 'RS_E_OilPress_PC2'

4: outlier is 'RS_E_RPM_PC1'

5: outlier is 'RS_E_RPM_PC2'

6: outlier is 'RS_E_WatTemp_PC1'

7: outlier is 'RS_E_WatTemp_PC2'

8: outlier is 'RS_T_OilTemp_PC1'

9: outlier is 'RS_T_OilTemp_PC2'

For example, a code of '-2' indicates that our algorithm did not identify any anomalies in that line. Alternatively, a code like '2468' signifies that the values of 'RS_E_OilPress_PC1', 'RS_E_RPM_PC1', 'RS_E_WatTemp_PC1', and 'RS_T_OilTemp_PC1' are considered anomalies.

Below, the pictures indicates 40 most popular codes, their count and the percentage of these codes in the dataframe:

	flag	count	percent		flag	count	percent		flag	count	percent		flag	count	percent
0	-2	12971369	73.370489	**10**	2468	125852	0.711862	**20**	23456789	88891	0.502798	**30**	26789	37684	0.213154
1	45	461795	2.612070	**11**	79	119939	0.678416	**21**	236789	81017	0.458260	**31**	145	37180	0.210303
2	1	285341	1.613986	**12**	5	114964	0.650276	**22**	135	70072	0.396351	**32**	89	28371	0.160476
3	0	265959	1.504355	**13**	9	104734	0.592411	**23**	24	58323	0.329895	**33**	689	23600	0.133490
4	2	263758	1.491905	**14**	3579	101950	0.576664	**24**	345	55517	0.314023	**34**	1345	22438	0.126917
5	3	224104	1.267609	**15**	8	97833	0.553377	**25**	6789	55292	0.312750	**35**	268	21912	0.123942
6	12345	169183	0.956957	**16**	4	97746	0.552885	**26**	1234567	42481	0.240287	**36**	789	19701	0.111436
7	7	153862	0.870296	**17**	23	96905	0.548128	**27**	246	40831	0.230954	**37**	234567	18649	0.105485
8	2345	152712	0.863791	**18**	245	96456	0.545588	**28**	1357	40698	0.230202	**38**	36789	17056	0.096475
9	68	127135	0.719119	**19**	6	90014	0.509150	**29**	13579	38368	0.217022	**39**	35	16065	0.090869

So, we have a new big data frame which is an original data frame with several new generated columns and a column with the code explained above. Then we grouped all rows by train_id, flag and date(we could choose any other time interval, but we think it is enough) and calculated

the count of each code and its percentage relative to the total number of rows. Then we state that a train has a real problem if the percentage is more than 40% and number of entries is more than 100(if we have a small number of rows we do not have enough statistics).

Applying these conditions we got a data frame with 1138 rows:

	mapped_veh_id	date	flag	count	percent
309242	196.0	2023-06-02	2	3148	76.705653
310461	196.0	2023-08-09	2	3079	86.854725
310656	196.0	2023-08-30	2	2996	84.632768
310597	196.0	2023-08-28	2	2919	82.155925
308501	196.0	2023-05-03	2	2842	81.292906
137450	148.0	2023-01-27	5	2767	88.799743
309260	196.0	2023-06-03	2	2762	99.495677
137550	148.0	2023-02-02	5	2726	90.384615
95359	136.0	2023-01-24	3	2662	93.142057
309172	196.0	2023-05-31	2	2655	74.411435

Figure 9, New dataframe after conditions application.

If the outliers flag contains only one number we believe that a train has a problem with one sensor which does not work correctly. If a train has a real problem which affects an engine for example, we believe that there will be a deviation in more than one sensor. This is because all five parameters for each engine are connected.

For example, oil pressure is higher when the engine is cold due to the increased viscosity of the oil. So, we expect that the deviation in the oil pressure should be accompanied by the deviation in oil temperature. Exploring the data, we can strictly confirm this physically motivated correlation.

Below is an illustration of a problem with one sensor:

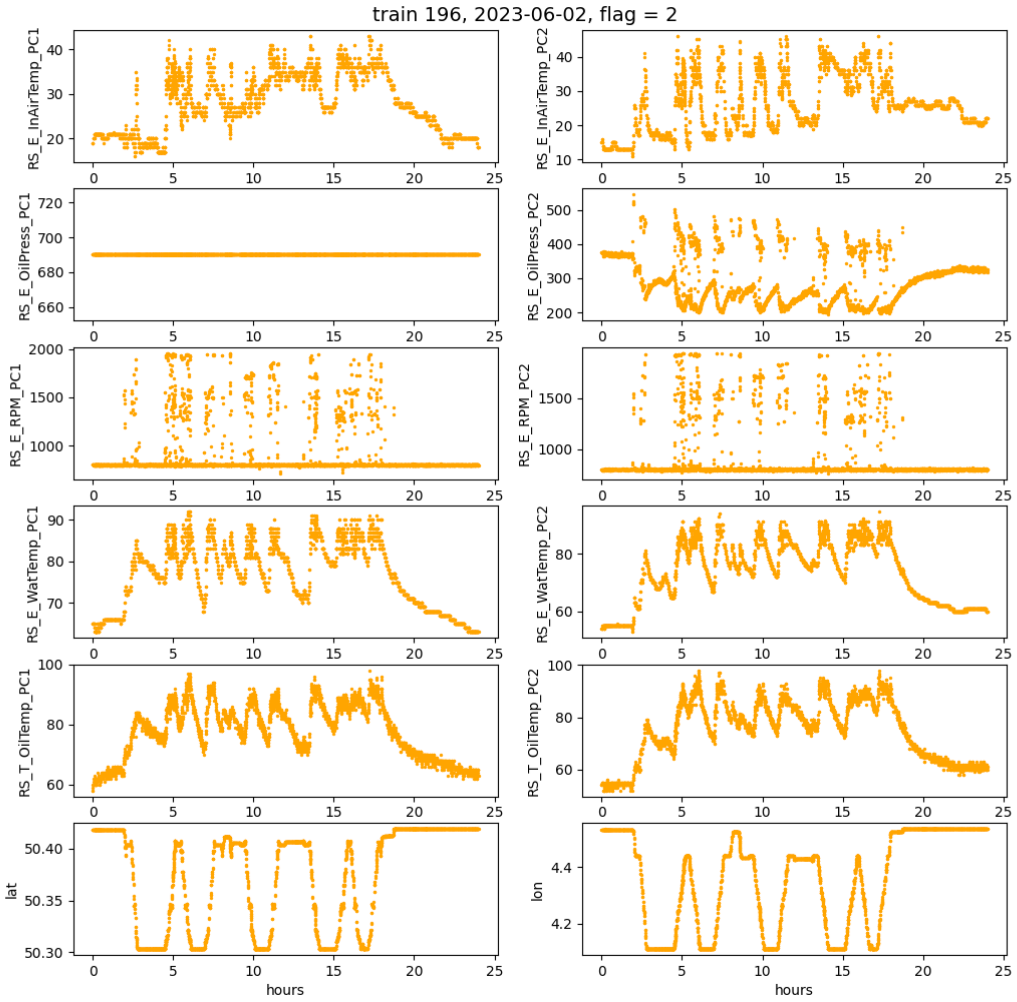


Figure 11, Representation of effective error on train 196.

Now, we consider more complicated and interesting problems.

An example below shows a case where 4 sensors of one engine deviate significantly from expected mean values. We argue that this indicates a breakdown in one of the train systems. As a confirmation of our hypothesis, we can follow the train behavior around this date. We can notice that the same problem appeared the day before. On the other hand, one can notice that the train spent the day after that in the workshop. The next day after the workshop, the train returned to operations and its indicators fluctuated around the expected mean. Clearly this

means that the train was fixed in the workshop which confirmed our initial guess. This demonstrates the capabilities of our model.

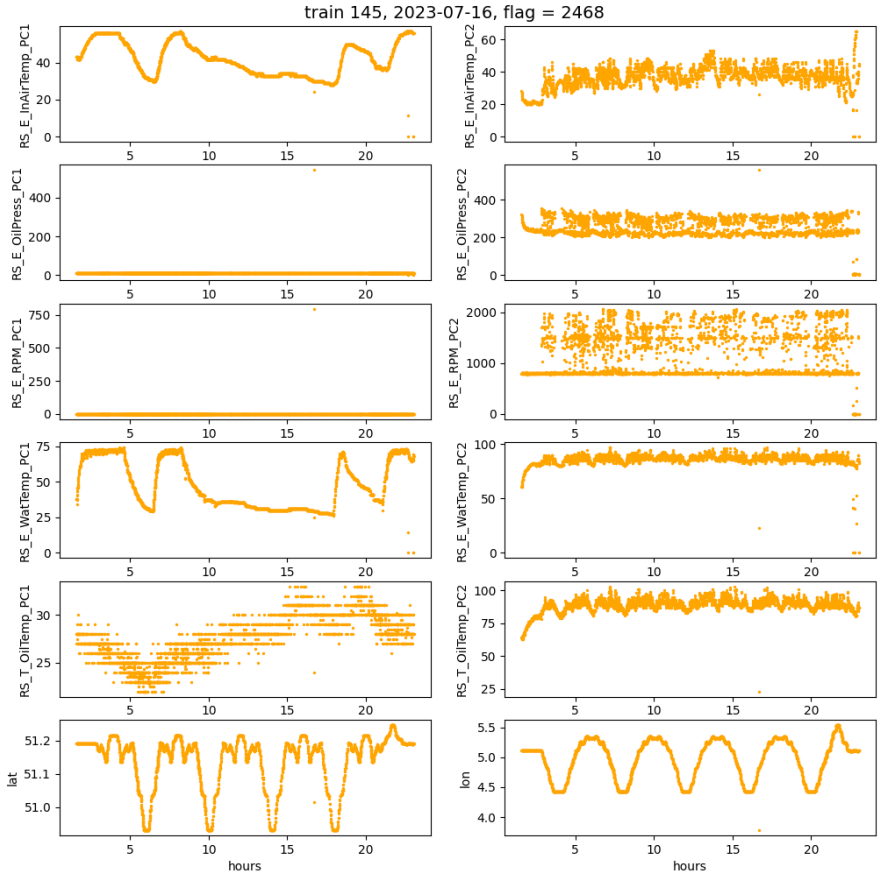


Figure 12, Train 145 features analysis on the 16.07.2023.

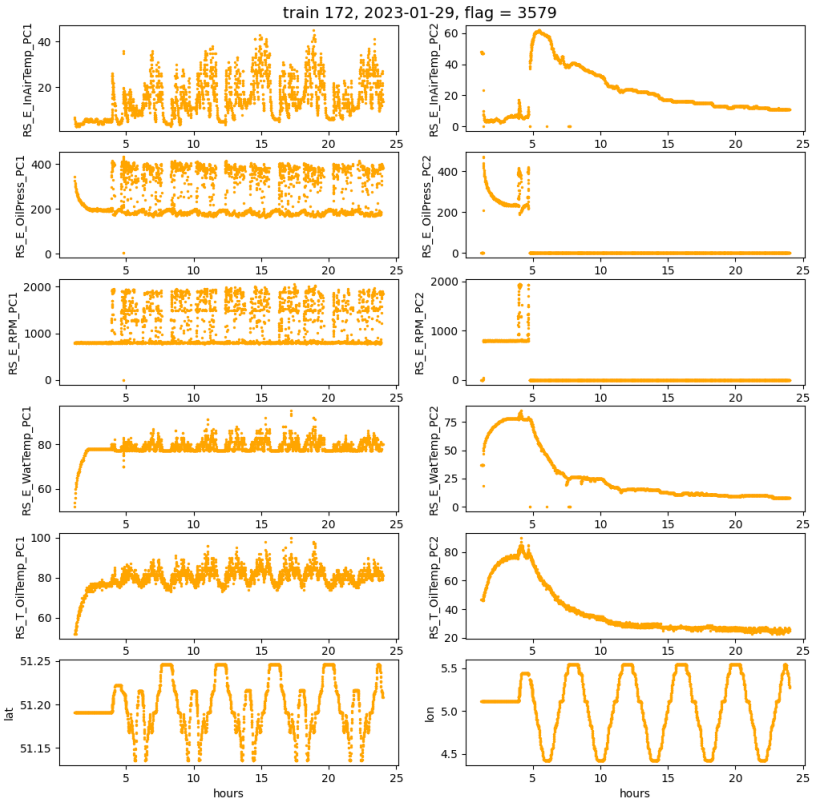


Figure 13, Train 172 features analysis on the 29.01.2023.

The last example demonstrates the train with 3 broken sensors (OilPressure1, RPM1 and WaterTemperatur1). However, the oil temperature of the first engine correlates strongly with the oil temperature of the second engine. Thus, we argue that in this situation both engines work well, but it is the sensors which broke down.

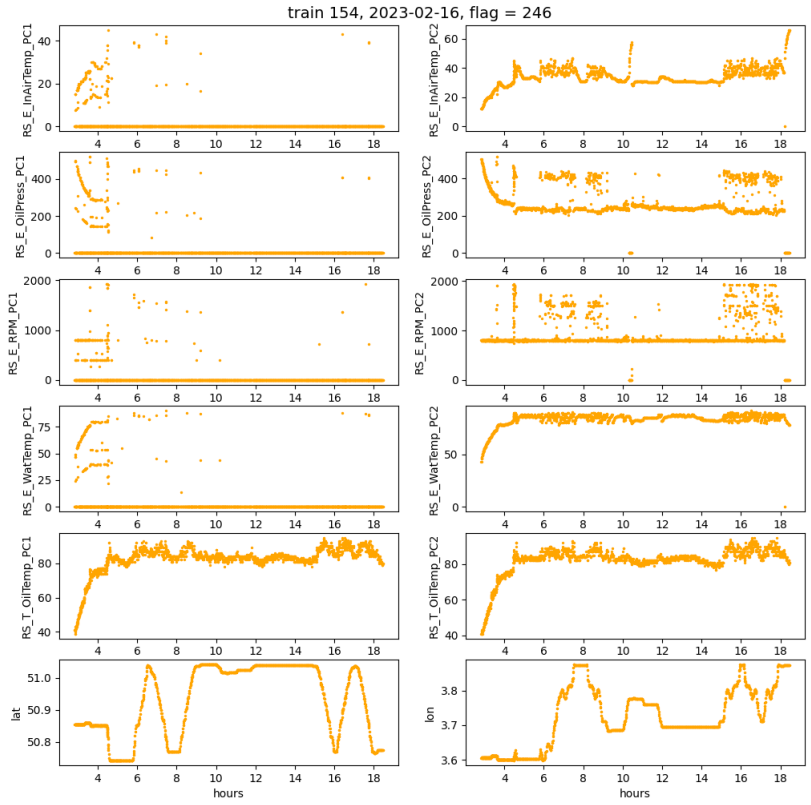


Figure 14, Train 154 features analysis on the 16.02.2023.

Clearly our model is not perfect and has its own advantages and disadvantages. As an advantage we can claim that our model is quite simple and easy to understand. However, it is flexible and has a number of parameters that can be tuned to increase performance. First group of parameters is a distance and temperature bin sizes. In our baseline model, we fix these values to roughly 10 km and 1 degree. Second parameter is a number of standard deviations around the mean that marks an acceptable region of values for a given sensor. In our analysis we fixed this threshold to 2*std. Increasing this number leads to model blindness to outliers while lowering this threshold decreases model sensitivity.

On the other hand, our model does not consider a special regime of work engine heating, acceleration etc. This can shift expected mean and standard deviation but we believe it won't change the main predictions of the model as these regimes occupy only a moderate fraction of time.

Our model can be directly used in streaming mode. Using the calculated lookup table our model can on fly determine if a given line of parameters is anomalous or not. If the anomaly persists, for example if 50 of 100 last parameter lines are detected as anomalous with the same flags, this indicates problems with sensors or breakdown of one of the systems of the train.

6. Dashboard creation

The dashboard has been created by using Dash library from Python.

Dash is a powerful tool for building interactive web applications, particularly in the data science domain. Its strengths lie in its ease of use, the python syntax, and its integration with Plotly. However, Dash is difficult to master, it takes time to develop exactly the dashboard you want.

The more graphs there are on the dashboard and the more interconnections there are between these graphs, the heavier and more difficult the integration work becomes.

By using this library, the goal was to quickly have a dashboard prototype with interactive graphics.

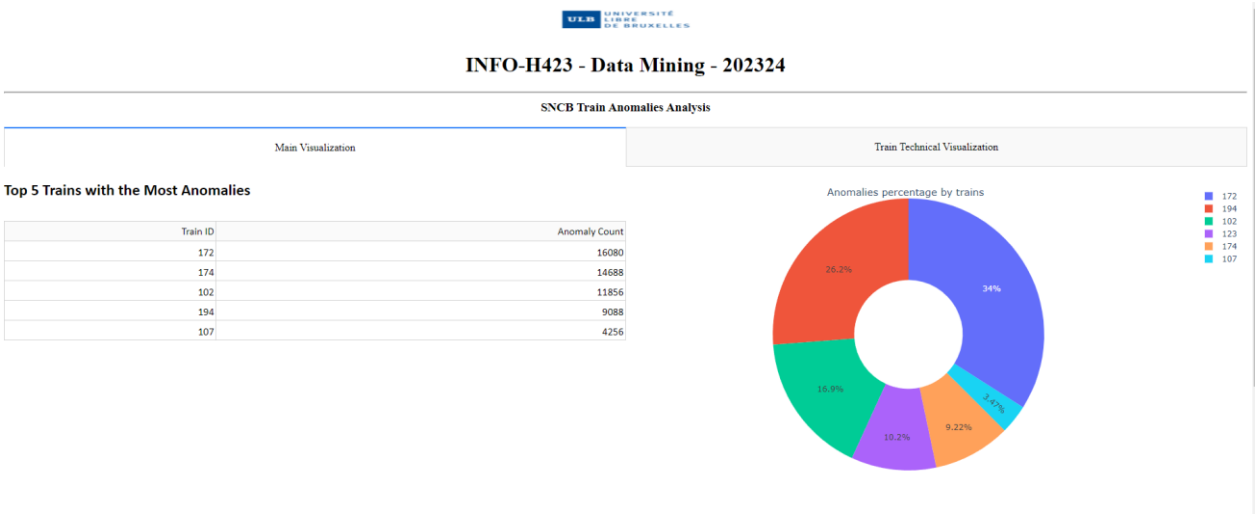


Figure 15, Main page dashboard.

The purpose of the dashboard is to help SNCB to identify and investigate the potential anomalies on specifics trains. In this context, we decided to have two tabs in the dashboard :

1. The Main Visualization tab.

2. The Train Technical Visualization tab.

6.1. The Main Visualization tab

The first tab, call “Main Visualization”, purpose is to give a high overview on the train identified by our model. The idea was to quickly being able to check which train have the most issues and what is the part of this train in the global percentage.

Here, with the sample data, we have a representation with two plots :

- The first one contains the Top 5 Train with the most Anomalies.
- The second one contains the repartition by percentage of anomalies by trains.

By looking at both plots, we see the train with the most anomalies seems to be the train 172 with 16.080 anomalies counted which represent 34% of the data. Pay attention to the fact that this representation is done on a subsample of the main data and the dashboard is a prototype.

As soon as we have noticed train 172, we would like to see how the information is distributed in the year for this train. There, the third and last plot of this tabs kick-in :

Figure 16, Number of entry for each trains by day.

In the filter list, we will select only the data for the train 172 :

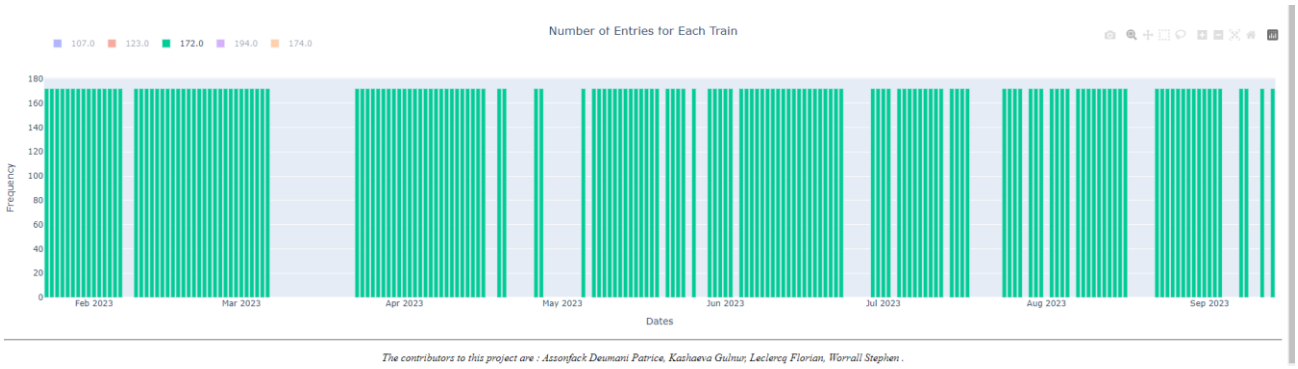


Figure 17, Data distribution for train 172.

So, if we sum up our logic, the first tab of the dashboard allows us to find the distribution of anomalies per train per day and identify a train we would like to investigate in the next tab.

6.2. The Train Technical Visualization tab

The second tab is dedicated to the technical analysis of the train on a day. It is composed by two plots :

- The first one is a plot containing an analysis of features information for a specific train. To select the train and the dates, you have two drop downs.
- The second one is a map where you can see the movement of the train on a map and the correlated external temperature.

We will continue if our example on train 172 and we will select a day (28.08.2023).

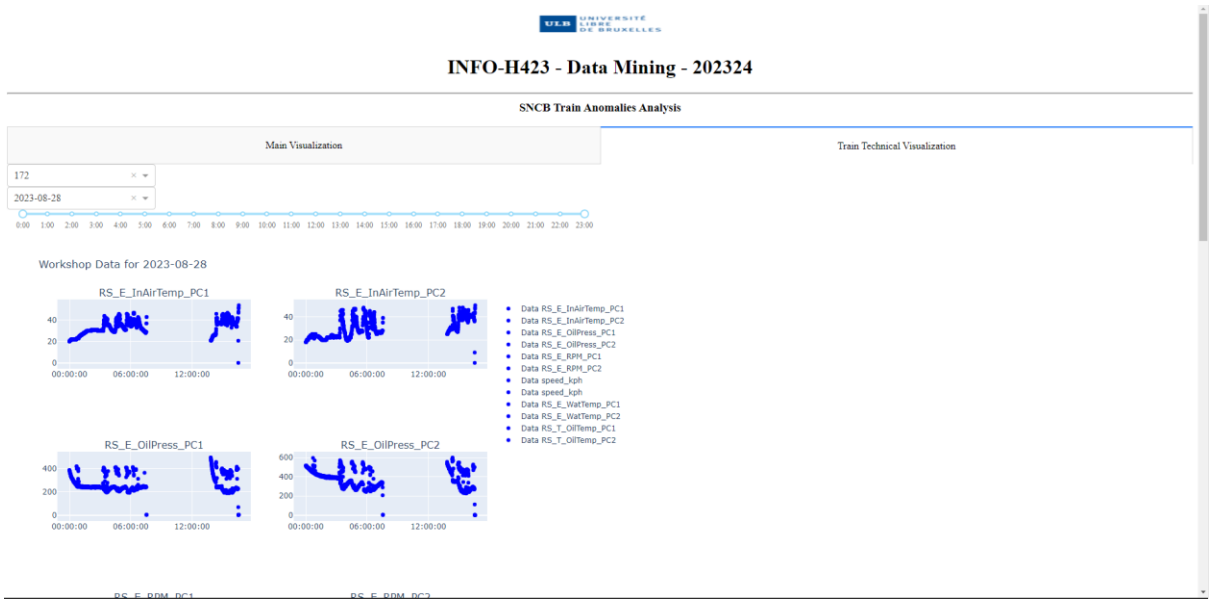


Figure 18, Feature analysis on train 172 the 28.08.2023.

As we can see on the first plot “RS_E_InAirTemp_PC1”, there seems to be an increase in the temperature at the end of the day. We can not well see it so we will zoom:

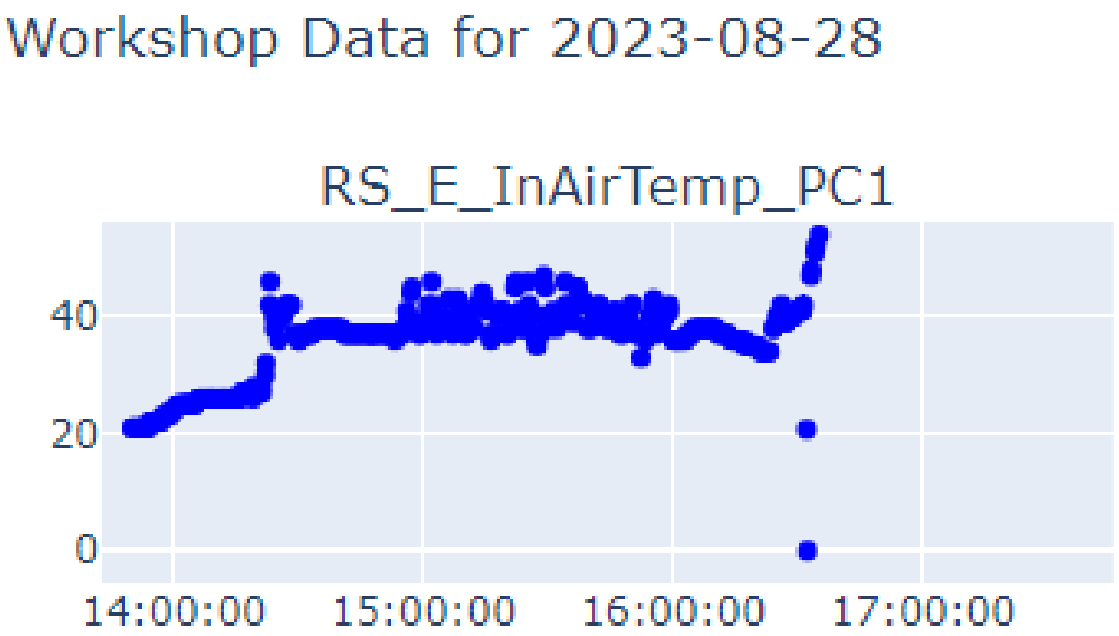


Figure 19, Zoom inside the plot.

For each train, we have 12 subplots showing these features :

Figure 20, Subplots in the technical analysis plots.

Finaly, we want to be able to see the information on a map and link it with the weather data. It's an interactive map where you can zoom in and out. The train color depends on weather data. Here, in our example for train 172 on the 28.08.2023, the external temperature is superior to 20 degrees.

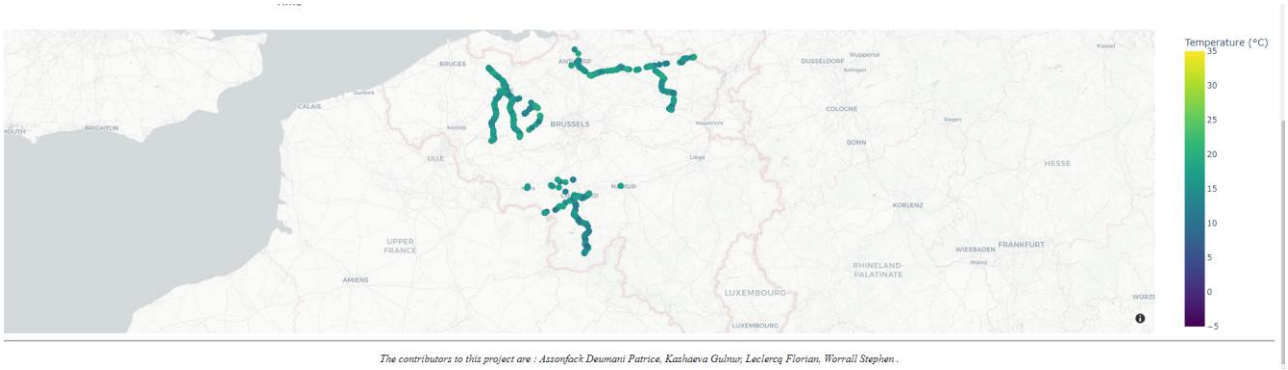


Figure 21, The interactive map.

We can zoom on Charleroi to have more details :

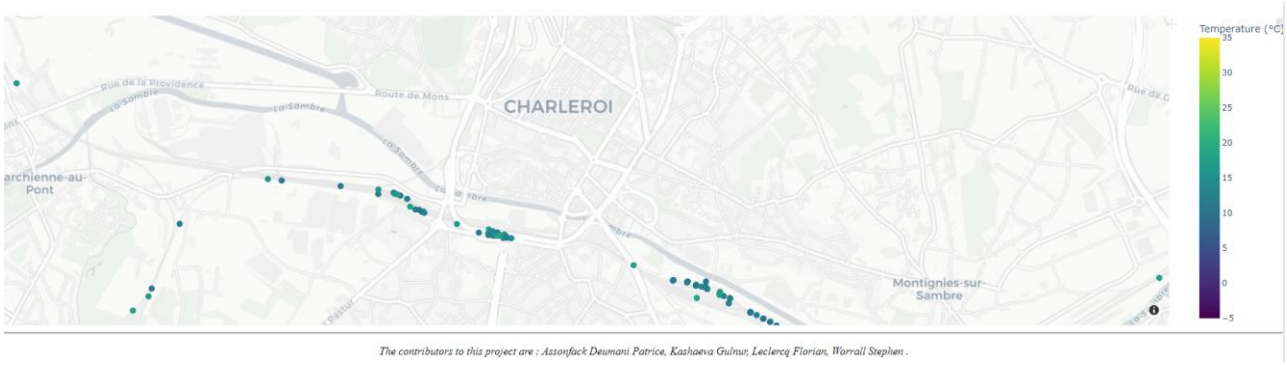


Figure 22, Charleroi zoom.

We can zoom until we see the building (here a workstation) :

Figure 23, Charleroi workstation.

The mapping is done with open street map.

7. Streaming package

Throughout the project we have developed the components of the streaming package, many of the rapidminer derivative functions have been replicated in python. Also research in jupyter notebooks has been reduced into concise python functions that can be found in the files section of Appendix 1. These scripts can be easily imported into a live streaming implementation of the work described in this paper.

Appendix 1: Files description:

hard columns:

hard columns are generated only from in-line single row data and is not dependent on sorting or surrounding rows that would need to be re-processed.

	weather files.
1ELEV.py	elevation generation script connects to local VM API (see opentopodata.tar.gz).
gpd_notebook.ipynb	long/lat to xy notebook used to test geopandas and to process the dataset.
lonlat_to_xy.py	long/lat to xy refined general function.

cleaning scripts and derived columns:

cleaning processes and soft column generation derived from several sorted/cleaned rows of data.

data_clean_stage1.py	cleaning step 1.
data_clean_stage1.py	cleaning step 2.
data_clean_stage3_combined_process.rmp	cleaning step 3 → generated soft derived columns → smoothing.

analysis:

miguelle data mining codes.ipynb	coded for data description and anomalies detection plots.

dashboard:

Dashboard_jupyter.zip	code for the dashboard and associated data samples.
Data_197.csv	csv file containing sample data for train 197.
Top5_Trains.csv	csv file containing sample data for the top 5 train dash table.
SNCB_ANOMALY_V3.csv	csv file containing sample data with the anomalies detected.
ULB_Logo.png	ULB log.
Dashboard_Video	A video use case of the dashboard in action.

large files and raw data files:

here is a link to large additional files hosted on google drive...
https://drive.google.com/drive/folders/1XPCdCy6faeMtUixY9P8J-B2gqUTTTfFj?usp=sharing

opentopodata.tar.gz	tarball of the complete, configured opentopodata docker implementation.
temp_weather.csv	weather and local temperature generated per record.
elevation_etopo1.csv	elevation data generated from GPS per record.
xy_out.csv	x, y Cartesian data generated from GPS per record.
cleaned_stage1.csv	result of rule 1 cleaning.
cleaned_stage2.csv	result of rule 2 cleaning.
full_expanded_dataset_cleaned_stage_3.csv	result of rule 3 cleaning + derived processes, smoothing.
full_expanded_dataset_cleaned_stage_3_mini.csv	reduced processed dataset (10 trains) for faster shape testing.
full_expanded_dataset_cleaned_stage_3_nano.csv	further reduced processed dataset (3 trains) for fastest shape testing.
data_cube.csv	all means and standard deviations mentioned in model creation section.
outliers.csv	results of outlier detection.
final_with_flag.zip	Final dataset with results of outliers flagged.