

De-Suphurization Model

(Detailed description)

Tools Used: Python, Microsoft SQL

Contents

| | | |
|-------|--|----|
| 1 | Objective..... | 3 |
| 2 | Business Case | 3 |
| 3 | Salient features of DeS Model..... | 3 |
| 4 | Working Principle | 3 |
| 4.1 | Inputs for the model:..... | 4 |
| 5 | Model Training and Final Model Selection | 4 |
| 5.1 | Model 1 (CG/CR Carbon Model): | 6 |
| 5.2 | Model 2 (CG/CR Oxygen Model) : | 6 |
| 5.3 | Model 3 (VAG Model): | 6 |
| 6 | Model Implementation/Integration on Server:..... | 7 |
| 6.1 | Input Data..... | 7 |
| 6.2 | Output Tables | 8 |
| 6.3 | Code Walkthrough | 9 |
| 6.3.1 | Scheduler.py..... | 9 |
| 6.3.2 | AI_prediction.py | 9 |
| 6.3.3 | Node_rules.py | 11 |
| 6.3.4 | Application Batch File (app.bat)..... | 11 |
| 6.4 | Task Scheduler | 11 |
| 7 | Architecture used for the Backend | 14 |

1 Objective

Determining the optimal addition of Aluminium material required for de-sulphurisation of liquid metal to reduce process time.

2 Business Case

At the Ladle Furnace, during the De-Sulphurisation process, Aluminium is added based on samples taken from the LF. This step of taking samples increases the overall process time. So there was a need for a model which could recommend optimal value of Aluminium to be added at the beginning stages so that de-sulphurisation occurs to a satisfactory degree within the first two samples.

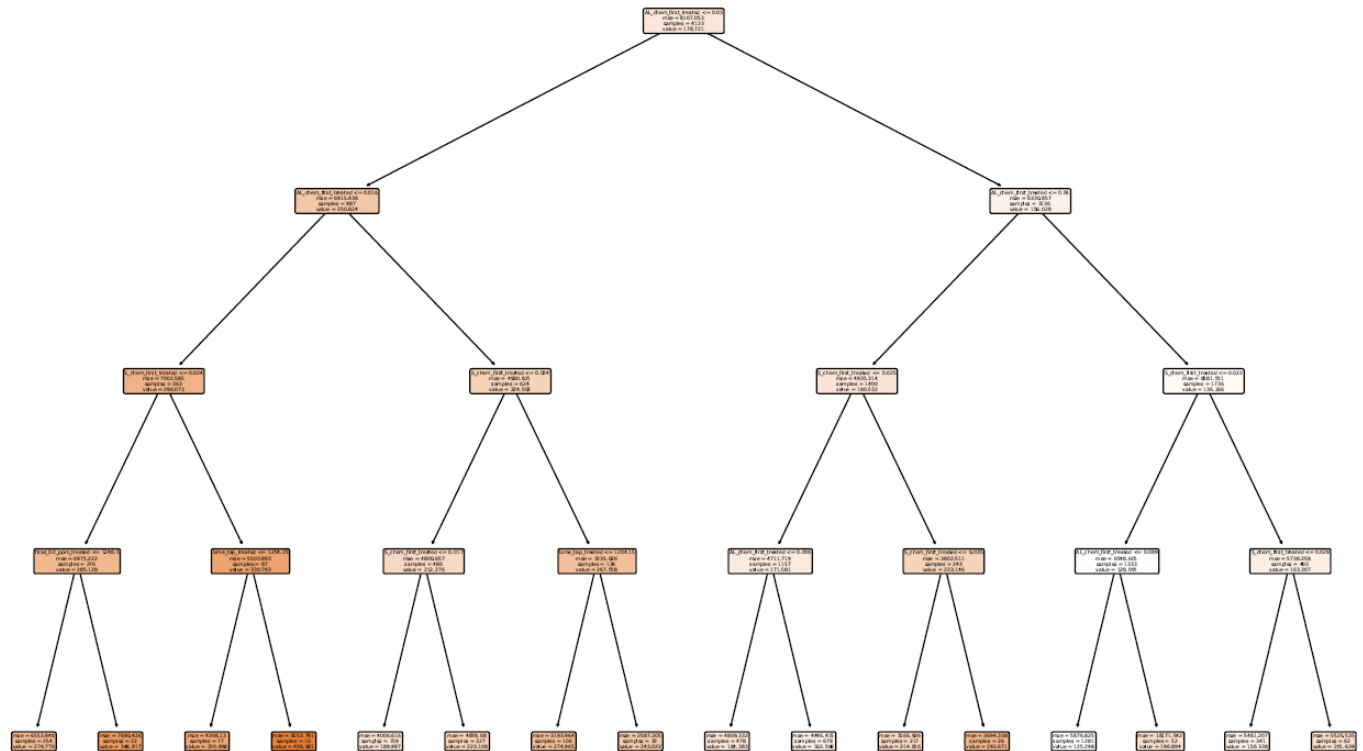
3 Salient features of DeS Model

The DeS model is developed to minimize heuristic decision making around amount of Aluminium to be added for optimal desulphurisation of steel. This model will enable operators to save on process time, aluminium material costs while at the same time ensure enough aluminium will be added so as to maintain optimal sulphur levels in the liquid metal.

- Model takes live input from L1/L2 server to account for all the parameters affecting the decision making and predicts the optimum aluminium material for each heat.
- Model is automatically run whenever a heat's sample chemistry report is recorded.
- Majority of the inputs are fetched directly from the existing systems.
- Prediction made by the model will be displayed in the L2 system for operators to use.

4 Working Principle

The model works on two prediction algorithms, namely Decision Trees and Multiple Regression. The model creates a Decision Tree from the data (example shown in image below). Then, using each of the leaves a multiple regression model is built. So the final model is an Ensemble of Decision Trees and Regression.

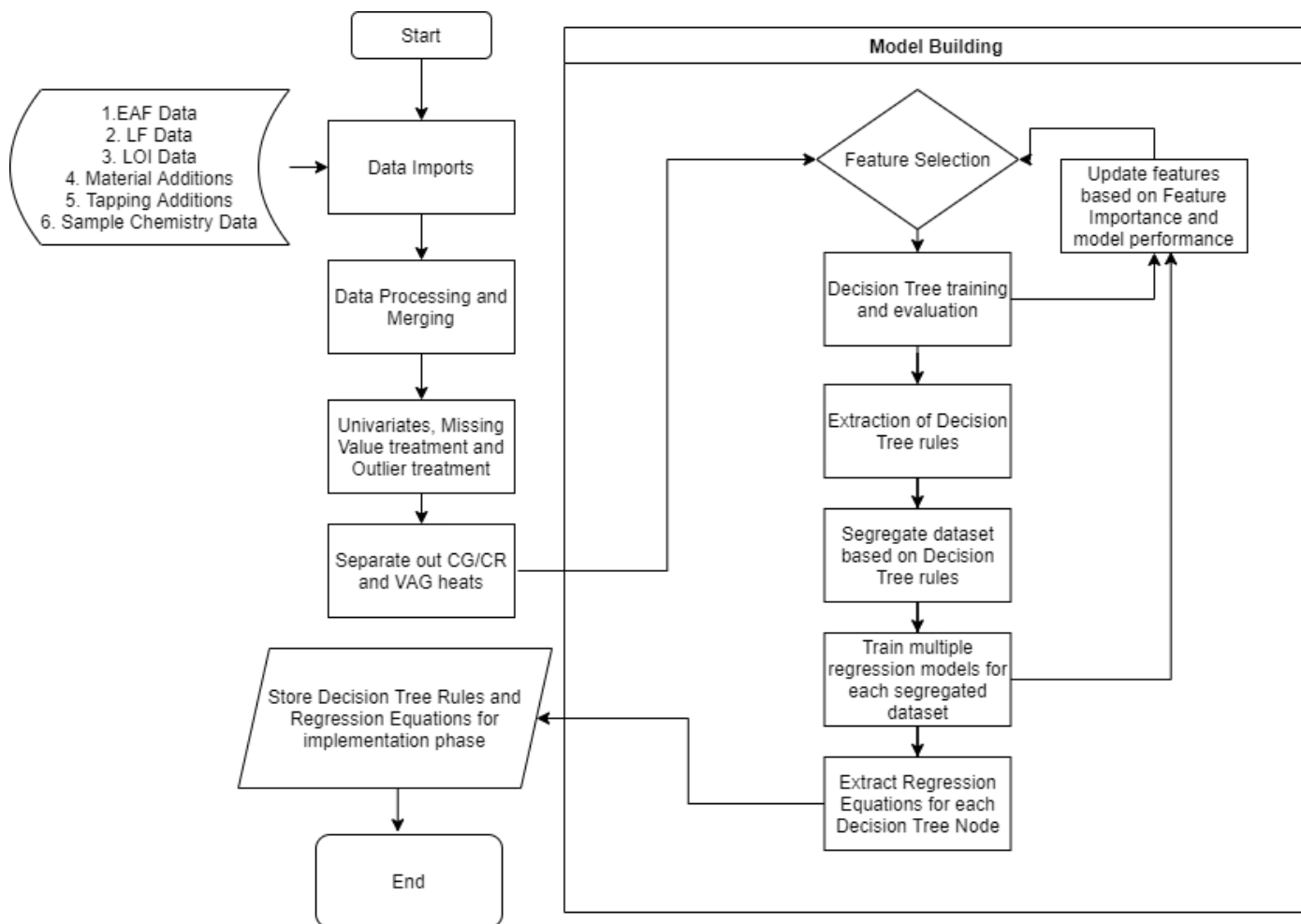


4.1 Inputs for the model:

- Liquid Metal Starting Weight
- First measured temperature of LM
- Aluminium Bar added
- Lime tapping addition
- SiMn tapping addition
- Opening Aluminium from first sample
- Opening Sulphur from first sample
- Opening Carbon from first sample
- Opening Silicon from first sample
- LOI (Loss on Ignition) Percentage
- LTA (Ladle Turnaround Time)
- Final Oxygen (ppm) in LM

5 Model Training and Final Model Selection

The model was built using data from Oct-2020 to Jul-2021. The following flowchart details the process of model training and selection.



Following this process, a total of three models were finalized. Here it is important to note that the final output of the model training exercise is a set of 16 conditions and regression equations that will help us predict aluminium material. Below, a hypothetical pseudocode is provided to help understand this.

This pseudocode illustrates the outputs of the model. The output of the decision trees i.e. the node conditions are highlighted in yellow, and the output of the regression model, i.e. the regression equations highlighted in blue. Together, these form the “node rules” based on which Aluminium is predicted. Here, “A”, “B” etc. are actually the independent variables like “Opening Aluminium”, “Opening Carbon” etc.

Function Aluminium Prediction(

```

If (A > 0.03) and (B <= 0.021) and (C < 0.04) and (D <= 0.1) then:
    Aluminium_material = 0.02 * A + 0.74 * B + 0.04 * C + (-2) * D + (-0.027) * E + (-0.1) * F
Elif (A < 0.03) and (B > 0.05) and (E > 0.06) and (F < 0.1) then:
    Aluminium_material = 0.07 * A + 0.35 * B + 0.1 * C + (-1) * D + (-0.5) * E + (-0.7) * F
Elif
.
.
.
.
.
.
Return Aluminium_material
  
```

)

Following this approach, a total of three models were trained. They are:

5.1 Model 1 (CG/CR Carbon Model):

This model was trained on only CG/CR heats. The features used in this model are:

- Liquid Metal Starting Weight
- First measured temperature of LM
- Aluminium Bar added
- Lime tapping addition
- SiMn tapping addition
- Opening Aluminium from first sample
- Opening Sulphur from first sample
- Opening Carbon from first sample
- Opening Silicon from first sample
- LOI (Loss on Ignition) Percentage
- LTA (Ladle Turnaround Time)

5.2 Model 2 (CG/CR Oxygen Model) :

This model was also trained only on CG/CR heats. The features used in the model are:

- Liquid Metal Starting Weight
- First measured temperature of LM
- Aluminium Bar added
- Lime tapping addition
- Opening Aluminium from first sample
- Opening Sulphur from first sample
- Final O2 in PPM

5.3 Model 3 (VAG Model):

This model was trained on only VAG heats. The features used in the model are:

- Liquid Metal Starting Weight
- Final O2 in PPM
- First measured temperature of LM
- Aluminium Bar added
- Lime tapping addition
- SiMn tapping addition
- Opening Aluminium from first sample
- Opening Sulphur from first sample
- LOI (Loss on Ignition) Percentage
- LTA (Ladle Turnaround Time)

The detailed equations for all the three models will be available in “node_rules.py” file embedded at the end of this document. The “final model” is an ensemble of the three models mentioned above. This is what has been implemented on the server as well, and will be elaborated on in the next section.

6 Model Implementation/Integration on Server:

The final model has been implemented on the STEELDNA server. The details of the implementation are recorded below.

6.1 Input Data

The model receives all required inputs from three SQL tables present within the **STEELDNA** SQL Server in the **DeS** database. The three tables used are:

1. **Dbo.heat_analysis:** This table contains first sample chemistry data. As soon as we receive the first sample chemistry for a heat, the model is automatically run (more on this in the code walkthrough section). The schema for this table is shown below, and only the relevant columns to the model are kept. The remaining columns can be ignored.

| Column Name | Data Type | Comments |
|----------------|---------------|--------------------------------|
| SEQ_VALUE | int | Index |
| MSG_TIME_STAMP | datetime | System datetime |
| MSG_FLAG | nvarchar(50) | Default column, not important |
| AGGREGATE | nvarchar(100) | Ladle Furnace number |
| SAMPLE_TIME | datetime | Time at which sample was taken |
| SAMPLE_CODE | nvarchar(50) | Can be either 25101 or 25201 |
| HEAT_NUMBER | nvarchar(50) | heat number |
| C | numeric(5, 3) | Opening Carbon |
| SI | numeric(5, 3) | Opening Silicon |
| S | numeric(5, 3) | Opening Sulphur |
| AL_TOTAL | numeric(6, 3) | Opening Aluminium |

2. **Dbo.lf_heat_data:** This table contains signals from the Ladle Furnace. So once we receive the first sample chemistry for the heat, we look at the lf_heat_data table to pick up data like Liquid Metal Starting Weight, LTA, First Measured Temperature etc. The table schema for the same is shown below:

| Column Name | Data Type | Comments |
|----------------|--------------|--|
| SEQ_VALUE | int | Index |
| MSG_TIME_STAMP | datetime | System datetime |
| MSG_FLAG | nvarchar(50) | default column, not important |
| HEAT_NUMBER | nvarchar(50) | heat number |
| GRADE_TYPE | nvarchar(50) | grade |
| LM_START_WT | float | liquid metal start weight |
| TAP_O2 | float | Oxygen content measured during tapping |
| O2AFTERCELOX | float | Oxygen added after EAF process |
| LTA | float | Ladle turnaround time |
| FIRSTMEASTEMP | float | First measured temperature |
| STATION | float | LF number |
| LIME | float | Lime tapping addition |
| SIMN | float | SiMn tapping addition |

| | | |
|-------|-------|------------------------|
| ALBAR | float | Aluminium bar addition |
|-------|-------|------------------------|

3. **Dbo.grade_mapping:** This is a fact table, i.e it is fixed . It's a mapping table which contains an exhaustive list of grades for which we will be predicting aluminium addition, as well as the grade type for the same. The grades can be either CG/CR grades or VAG grades. The model predicts for these two types separately, and we need to know what type of grade each heat is, hence the need for this mapping table. The table schema is as shown below:

| Column Name | Data Type | Comments |
|-------------|--------------|-----------------------|
| GRADE | nvarchar(50) | Grade |
| GRADE_TYPE | nvarchar(50) | Grade Type (CGCR/VAG) |

6.2 Output Tables

Once the model predicts the optimal value of Aluminium material, these predictions should be stored in the SQL database as well. This is done in two different tables.

1. **Dbo.output_table:** This is the main output table. Here we only save the required information like heat number, LF number, Aluminium prediction in Kgs and Aluminium prediction in metres. This table is what is pulled into the L2 screen and displayed to the operators. The schema for the same as shown below:

| Column Name | Data Type | Comments |
|----------------|--------------|--------------------------------|
| SEQ_VALUE | int | Index |
| MSG_TIME_STAMP | datetime | System date |
| HEAT_NUMBER | nvarchar(50) | heat number |
| AGGREGATE | nvarchar(50) | LF number |
| AL_KGS | float | Aluminium prediction in Kgs |
| AL_MTS | float | Aluminium prediction in Metres |
| MSG_FLAG | nvarchar(50) | Default |
| UPDATE_DATE | datetime | Time at which L2 is updated |

2. **Dbo.interim_outputs:** This table contains all the information regarding a heat, including all the input values used and what the final predictions were. This is used as a validation table to verify the efficacy or debug the model if necessary. The schema for the same is shown below:

| Column Name | Data Type | Comments |
|----------------|--------------|-----------------------|
| SEQ_VALUE | int | Index |
| MSG_TIME_STAMP | datetime | System date |
| MSG_FLAG | nvarchar(1) | default |
| HEAT_NUMBER | nvarchar(50) | heat number |
| GRADE | nvarchar(50) | grade |
| GRADE_TYPE | nvarchar(50) | grade type (CgCr/VAG) |
| AGGREGATE | nvarchar(50) | LF number |
| STATION | int | LF number |
| AL_chem_first | float | Opening aluminium |
| S_chem_first | float | Opening Sulphur |

| | | |
|---------------------|-------|---|
| C_chem_first | float | Opening Carbon |
| SI_chem_first | float | Opening Silicon |
| First_Probe_temp_If | float | First measured temperature |
| Al_Bar_If | float | Aluminium bar added |
| LM_Start_Wt_If | float | LM starting weight |
| LTA_If | float | Ladle turnaround time |
| Lime_tap | float | Tapping lime addition |
| Simn_Tap | float | Tapping SiMn addition |
| final_O2_ppm | float | final oxygen content in ppm |
| LOI_perc | float | Loss on Ignition |
| AL_KGS | float | Predicted Aluminium Kgs |
| AL_MTS | float | Predicted Aluminium metres |
| Carbon_Node | int | Which Carbon Node the heat fell under (for debugging) |
| Oxygen_Node | int | Which Oxygen Node the heat fell under (for debugging) |

6.3 Code Walkthrough

There are three python scripts that are used in the implementation of the model, and here we will give an overview of them in the order in which they are triggered. These codes can be found in the server at the following location: “C:\DeS\Codes”. The summary below is intended to provide a logical overview of the codes, and it is recommended that you run the code simultaneously to get a better understanding of the processes.

6.3.1 Scheduler.py

The scheduler script continuously runs every 5 seconds in a minute. The purpose of this code is simply to call the “main” function from the al_prediction.py file. The scheduler stops when the current time has a “seconds” value anywhere between 55 to 60. This is because we are also setting up a windows task using the Task Scheduler, which runs the scheduler every minute. The minimum duration at which a Task Scheduler can be scheduled is 1 minute, which is why we use a python scheduler to run the model every 5 seconds within that 1 minute. We will discuss again in the section dedicated to the Task Scheduler, but the rationale behind the same can be easily grasped using an example.

Assume that the Task Scheduler runs this python script at 12:00:00. The python scheduler will run the **al_prediction** code every 5 seconds until 12:00:55:10 (taking a hypothetical drift of 10 milliseconds), at which point it terminates. Then the Task Scheduler will run the python scheduler again at 12:01:00. This way, we ensure that even if the python code is terminated due to some system issue at any point, it will restart again at the next minute, and will continue running the model every 5 seconds.

6.3.2 Al_prediction.py

The al_prediction.py is the main part of the application. This code contains two logical sections:

1. **Check if a prediction is to be made:** Whenever the al_prediction.py file is called, it first checks if a new heat’s sample chemistry has been updated in the dbo.heat_analysis table. It identifies a “new” heat in the following way:
 - a. It pulls the latest 100 heats from the dbo.heat_analysis table.
 - b. It then filters out this list by removing:

- i. Any heat which occurred more than 3 hours ago
 - ii. Any heat from Ladle Furnace 5 (we are not predicting for these)
 - iii. Any heat where the latest sample code is not 25101 or 25201. 25101 is the sample code for First Pass First Sample, and 25201 is the sample code for Second Pass First Sample. We will only predict for Second Pass First Sample if we have not already made a prediction during First Pass. This is taken care of in the next two steps.
 - c. It then pulls the latest 100 heats from the `dbo.output_table` as well. This is the table where all predictions are stored and is detailed in the **Output Tables** section of this document. It drops all the heats for which we have already made a prediction.
 - d. This leaves us with a list of heats which are newly started, and we have yet to make a prediction for them. This list of heats is then passed to the next section of the code, which is the prediction section.
2. **Aluminium Prediction:** The second chunk of this code deals with the actual aluminium prediction. This is taken care of by the main function of the `al_prediction.py` script. The series of steps followed is as listed below:
- a. If the list of new heats to make predictions for is not empty, we proceed with the prediction code.
 - b. The first step is to import the latest data from the SQL database. So we import the latest 1000 rows from `dbo.heat_analysis` and `dbo.lf_heat_data`. Then we only keep those heats which we've identified as the new heats for which we are to make predictions.
 - c. We also import the `dbo.grade_mapping` table in entirety (it's a fixed size table, which doesn't change).
 - d. Then we move on to the pre-processing stage. Here we do the basic pre-processing steps listed below:
 - i. Creating "final_O2_ppm". This is a sum of "Tap_O2" and "O2AfterCelox"
 - ii. Converting Al_Bar to tons.
 - iii. Renaming columns
 - iv. Keeping only the latest signal from LF
 - v. Keeping the latest chemistry data
 - vi. Merging the LF data (`lf_heat_data`) with chemistry data (`heat_analysis`).
 - vii. Clipping values. This is done to take care of any incorrect data or missing data issues.
 - viii. Creating "LOI Perc". This is a constant value.
 - ix. Merging with the grade mapping table. In this step, we will drop any heats which are of grades not present in the grade mapping table. So for example, if the heat is a billet grade, then it will be dropped as we are not predicting for those grades.
 - e. After all these processing steps, we check if we still have any heats for which we are to make predictions, if yes, we move on to the next stage.
 - f. If the heat is of type CG/CR, then we call the function for CG/CR prediction. If the heat is of type VAG, then we call the function for VAG prediction. The actual prediction is done using node rules and regression equations taken from the `node_rules.py` file, which will be discussed right after this.
 - g. Once the aluminium prediction is done, it is converted to meters by multiplying by 3. 10Kgs of aluminium is usually achieved by adding 30m of aluminium wire, hence the conversion factor.
 - h. The predictions as well as all the input data is then inserted into the `dbo.interim_outputs` table. This table is used to cross-check/validate the predictions.
 - i. The final predictions itself, along with information like heat number, ladle furnace number etc is stored in the `dbo.output_table`. This table is directly picked up to display the output values in the L2 screen.

6.3.3 Node_rules.py

The node_rules.py script contains the various regression equations as well as the decision tree-based conditions on when to use these equations. Each of these functions is of the form shown in the below pseudocode:

```
If first condition is satisfied:  
    Use the first regression equation to predict aluminium material  
Else if second condition is satisfied:  
    Use the second regression equation to predict aluminium material  
Else:  
    Use default equation
```

This is main “predictive” engine of the code. There are three functions which enable this. They are:

1. **Oxygen_tree:** This function contains all the rules and equations obtained from the “oxygen model” that was trained (refer Model Training section).
2. **Cgcr_aluminium_prediction:** This function is a combination of both the oxygen and carbon models. These rules were arrived upon after extensive analysis of the carbon and oxygen model predictions.
3. **Vag_aluminium_prediction:** This function is the set of rules and equations obtained from the model that was trained on purely VAG data.

These functions are called from the al_prediction.py file based on the grade type of a given heat.

6.3.4 Application Batch File (app.bat)

This is a batch file with only one command: python “C:\DeS\Codes\scheduler.py”. This batch file is what triggers all the scripts to run. The batch file is run by the Task Scheduler at every minute.

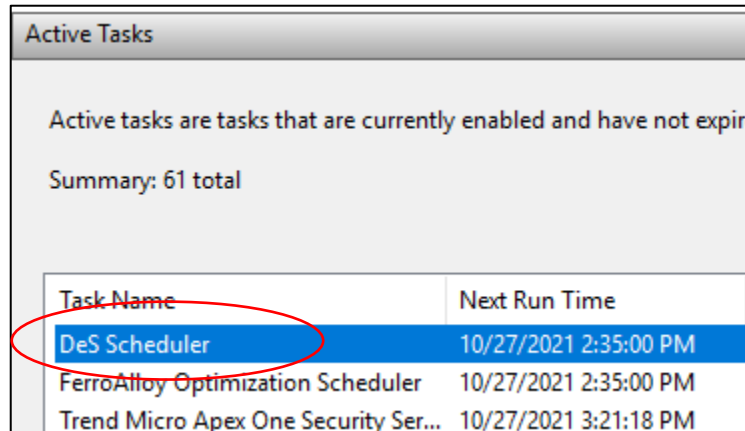
6.4 Task Scheduler

As mentioned in the previous section, the entire codebase is run every 5 seconds to check if there is a new heat, and if yes, a prediction is made for that heat. The Windows Task Scheduler is used to start this codebase every minute, and the python code runs every 5 seconds in that minute. As mentioned above, this is done to ensure that even if the python script is terminated due to some system issue, it gets restarted again in the subsequent minute. Additionally, it is also more efficient as it does away with RAM/Cache issue involved in running the same instance of the python script indefinitely. Task Scheduler can only repeat an action at a minimum interval of 1 minute, and hence the need for a python-based scheduler as well.

The DeS scheduler task can be accessed/modified by following the steps shown below:

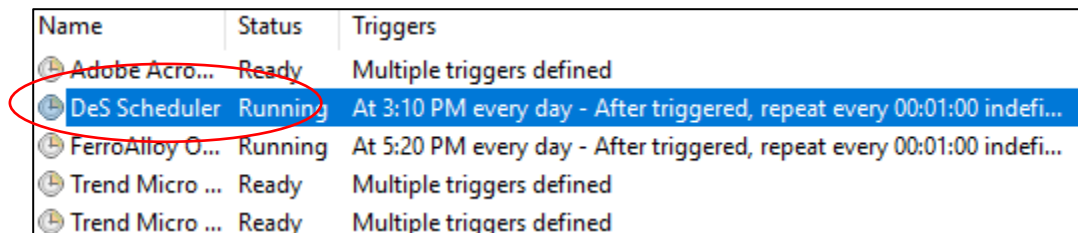
1. Open the Windows Task Scheduler (Start -> Windows Task Scheduler).

2. In the list of Active Tasks, select the task named “**DeS Scheduler**”. Double click to open.



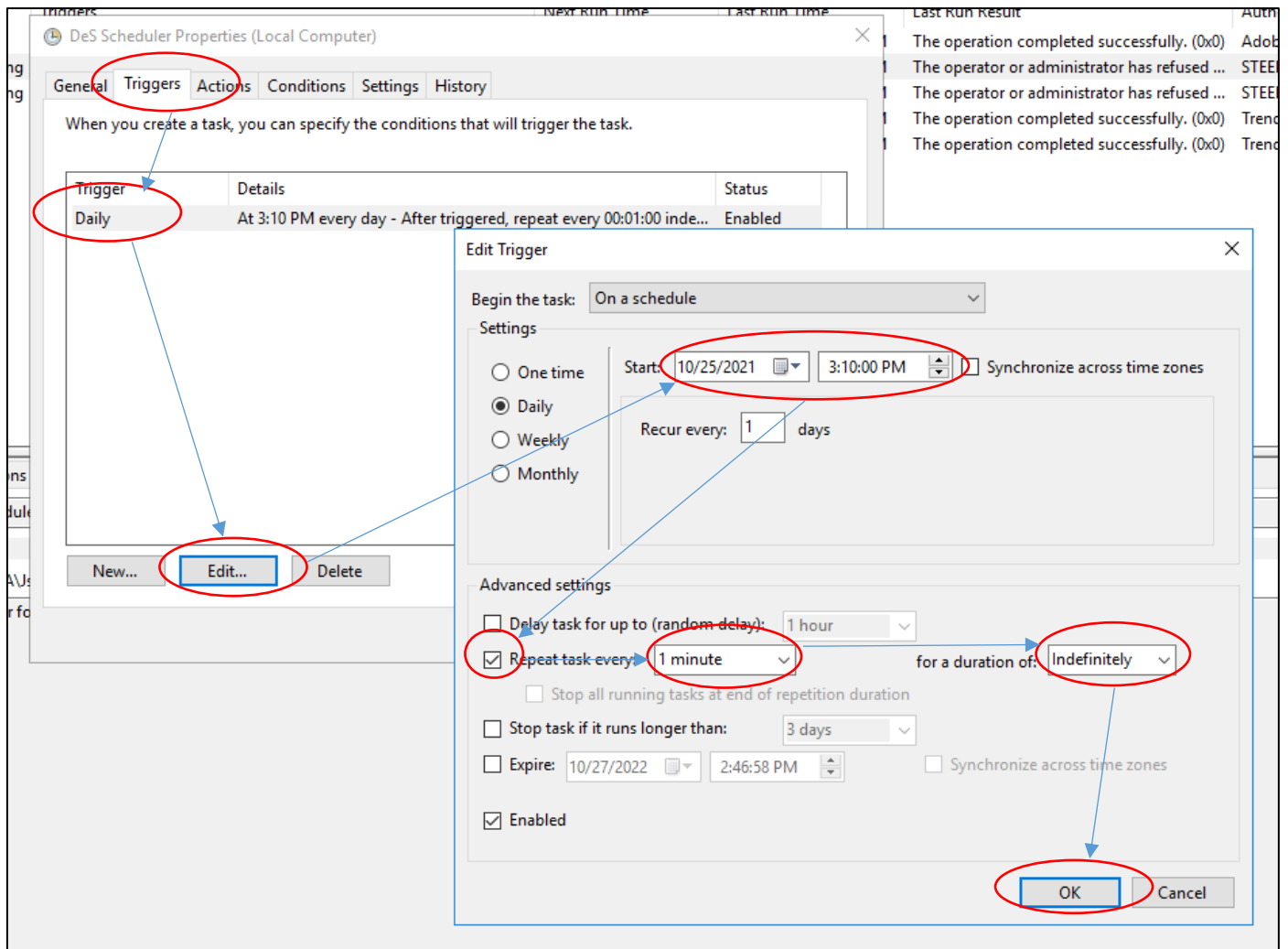
| Task Name | Next Run Time |
|--------------------------------------|-----------------------|
| DeS Scheduler | 10/27/2021 2:35:00 PM |
| FerroAlloy Optimization Scheduler | 10/27/2021 2:35:00 PM |
| Trend Micro Apex One Security Ser... | 10/27/2021 3:21:18 PM |

3. In the opened window, right click on ‘DeS Scheduler’ and choose “Properties”.

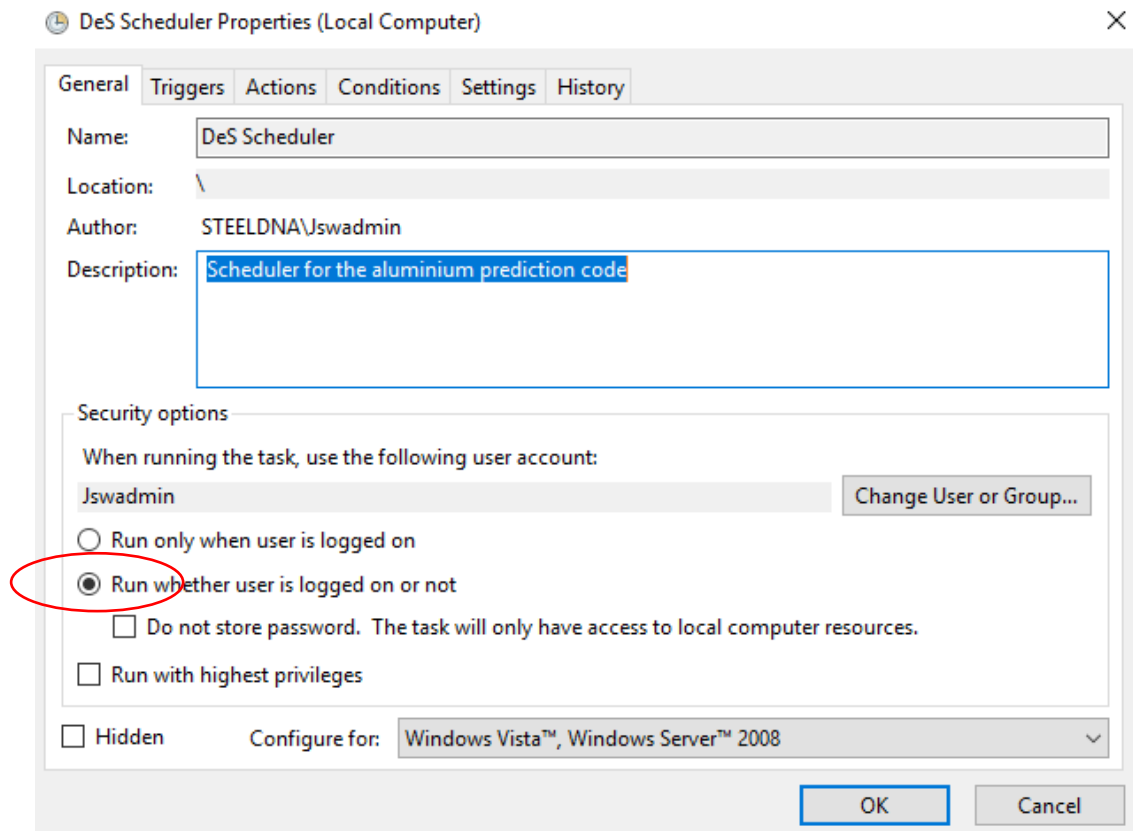


| Name | Status | Triggers |
|-----------------|---------|---|
| Adobe Acro... | Ready | Multiple triggers defined |
| DeS Scheduler | Running | At 3:10 PM every day - After triggered, repeat every 00:01:00 indefi... |
| FerroAlloy O... | Running | At 5:20 PM every day - After triggered, repeat every 00:01:00 indefi... |
| Trend Micro ... | Ready | Multiple triggers defined |
| Trend Micro ... | Ready | Multiple triggers defined |

4. In the properties window, you can modify things like when to trigger the python code, how often to repeat etc. For example, the steps involved in changing the starting time/repetition time is shown in the below image.



5. To ensure that the Task Scheduler runs the task in the background and does not pop out a terminal every time it starts the task, tick the option “Run whether user is logged on or not”.



7 Architecture used for the Backend

The model is developed in collaboration with the IT team of JSW Dolvi.

- **Python Version Used:** Python 3.8.8
- **SQL Server Used:**
 - o SQL Server Management Studio: 15.0.18384.0
 - o SQL Server Management Objects (SMO): 16.100.46367.54
 - o Microsoft Analysis Services Client Tools: 15.0.19535.0
 - o Microsoft Data Access Components (MDAC): 10.0.14393.0
 - o Microsoft MSXML: 3.0 6.0
 - o Microsoft .NET Framework: 4.0.30319.42000
 - o Operating System: 10.0.14393