DeS Model - Athena Wave III, Desulphurization

Project - JSW Digital Transformation

Purpose – Handover and Knowledge Transfer of DeS Model to JSW Team

Date – 13th April 2022

Table of Contents

1 Objective

The objective of the model is to determine the optimal addition of Aluminium material required for de-sulphurisation of liquid metal to reduce process time.
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.
The model considers the following parameters for decision making.

- 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

2 DeS Prediction Model

2.1 JSW Server Details

a. ID : 172.21.25.113, Password : admin@123
b. Codebase Location : C:\DeS\Codes

2.2 Database Table Structure

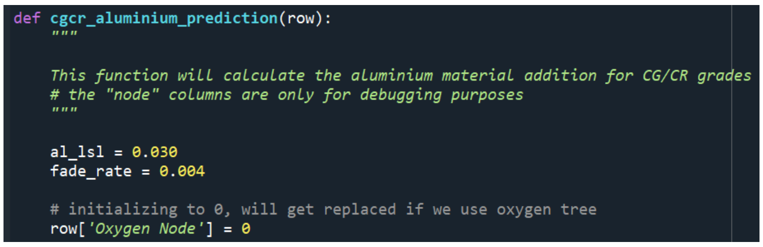
- Database Name – Des
- Input Tables-
    - heat_analysis
    - lf_data
    - grade_mapping
- Output Tables-
    - Intermin_output
    - Output_table

2.3 Node Rules (File: C:\DeS\Codes\node_rules.py)

This file contains the node rules (regression equations) determined using historic data. Based on these rules/equations, model will give predictions for live Heats.

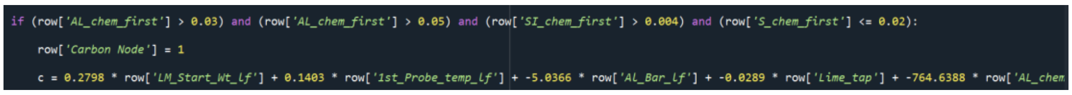
We have 16 different equations for each of CG/CR Grades and VAG Grades.

i. Def cgcr_aluminium_prediction(row)



```
def cgcr_aluminium_prediction(row):
    """

    This function will calculate the aluminium material addition for CG/CR grades
    # the "node" columns are only for debugging purposes
    """

    al_lsl = 0.030
    fade_rate = 0.004

    # initializing to 0, will get replaced if we use oxygen tree
    row['Oxygen Node'] = 0
```

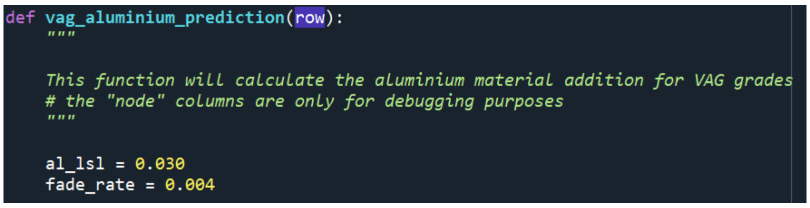
al_lsl = target aluminium, currently set at .03



```
if (row['AL_chem_first'] > 0.03) and (row['AL_chem_first'] > 0.05) and (row['SI_chem_first'] > 0.004) and (row['S_chem_first'] <= 0.02):

    row['Carbon Node'] = 1

    c = 0.2798 * row['LM_Start_Wt_Lf'] + 0.1403 * row['1st_Probe_temp_Lf'] + -5.0366 * row['AL_Bar_Lf'] + -0.0289 * row['Lime_tap'] + -764.6388 * row['AL_chem
```

Eg. Model checks the input parameters to determine which if/else statement does the heat lie in.
And then based on the equation (c=….), the aluminium value is calcuated

ii. Def vag_aluminium_prediction(row)



```
def vag_aluminium_prediction(row):
    """

    This function will calculate the aluminium material addition for VAG grades
    # the "node" columns are only for debugging purposes
    """

    al_lsl = 0.030
    fade_rate = 0.004
```

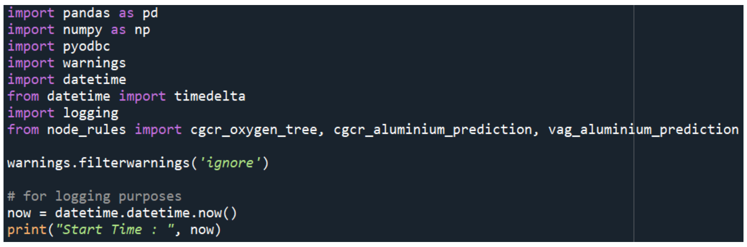
Once any of these functions is called, it checks which node does the current heat lie in based on input conditions and calculates Al requirement accordingly.

2.4 DeS (File: C:\DeS\Codes\al_prediction.py)

This is the main code which takes live input of heat from the database, executes node_rules to calculate aluminium required, and stores results in database

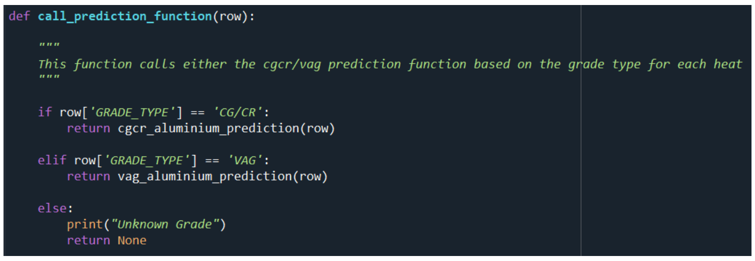
Steps:

i. **Import Required Libraries**



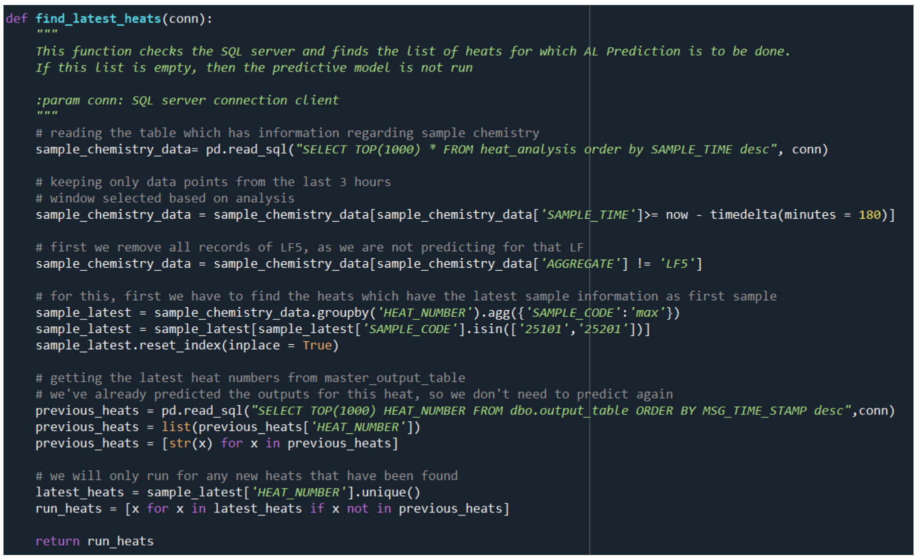
```python
import pandas as pd
import numpy as np
import pyodbc
import warnings
import datetime
from datetime import timedelta
import logging
from node_rules import cgcr_oxygen_tree, cgcr_aluminium_prediction, vag_aluminium_prediction

warnings.filterwarnings('ignore')

# for logging purposes
now = datetime.datetime.now()
print("Start Time : ", now)
```

ii. **Check Grade Type**
Check the type of Grade of the live heat and call the prediction function accordingly from node_rules.py



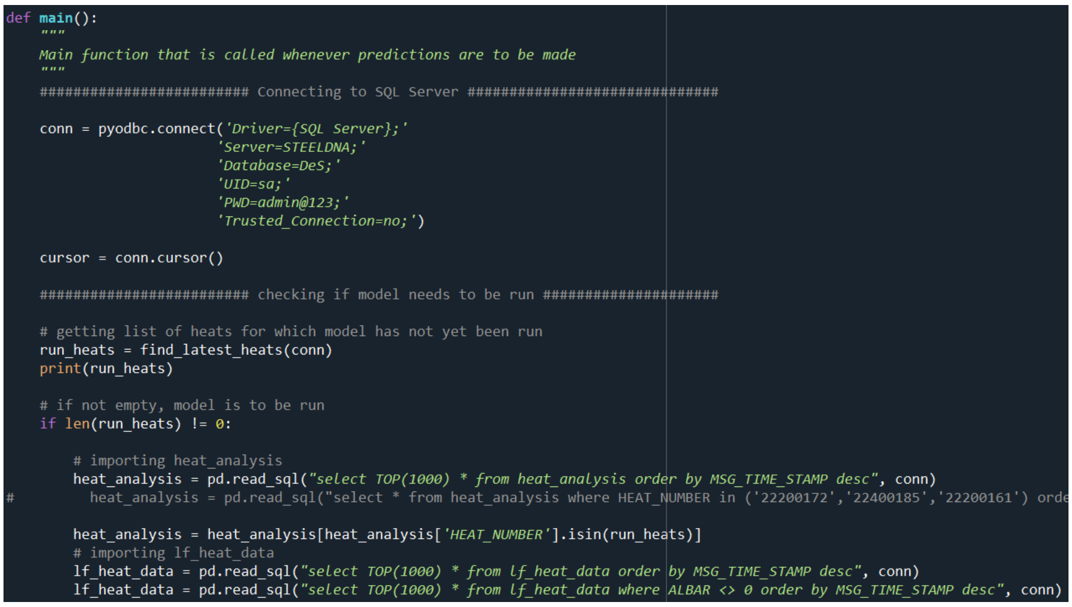
```python
def call_prediction_function(row):

    """
    This function calls either the cgcr/vag prediction function based on the grade type for each heat
    """

    if row['GRADE_TYPE'] == 'CG/CR':
        return cgcr_aluminium_prediction(row)

    elif row['GRADE_TYPE'] == 'VAG':
        return vag_aluminium_prediction(row)

    else:
        print("Unknown Grade")
        return None
```

iii. **Find Latest Heats**
   a. Connect to the JSW Database, and select latest heats from des.heat_analysis table.
   b. Filter out ‘LF5’ Shell since we don’t use it for DeS predictions
   c. Filter out the heats for which model has already generated results and stored in the des.output_table

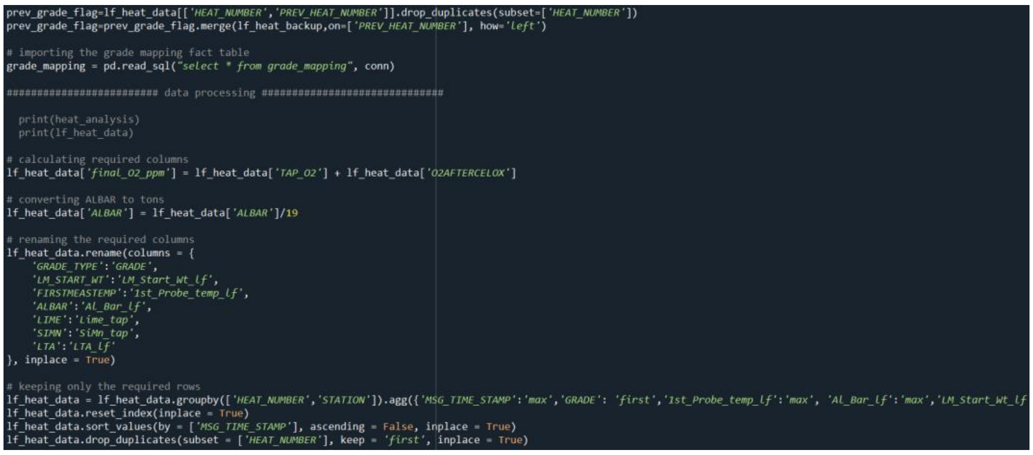


```python
def find_latest_heats(conn):
    """
    This function checks the SQL server and finds the list of heats for which AL Prediction is to be done.
    If this list is empty, then the predictive model is not run

    :param conn: SQL server connection client
    """
    # reading the table which has information regarding sample chemistry
    sample_chemistry_data= pd.read_sql("SELECT TOP(1000) * FROM heat_analysis order by SAMPLE_TIME desc", conn)

    # keeping only data points from the last 3 hours
    # window selected based on analysis
    sample_chemistry_data = sample_chemistry_data[sample_chemistry_data['SAMPLE_TIME']>= now - timedelta(minutes = 180)]

    # first we remove all records of LF5, as we are not predicting for that LF
    sample_chemistry_data = sample_chemistry_data[sample_chemistry_data['AGGREGATE'] != 'LF5']

    # for this, first we have to find the heats which have the latest sample information as first sample
    sample_latest = sample_chemistry_data.groupby('HEAT_NUMBER').agg({'SAMPLE_CODE':'max'})
    sample_latest = sample_latest[sample_latest['SAMPLE_CODE'].isin(['25101','25201'])]
    sample_latest.reset_index(inplace = True)

    # getting the latest heat numbers from master_output_table
    # we've already predicted the outputs for this heat, so we don't need to predict again
    previous_heats = pd.read_sql("SELECT TOP(1000) HEAT_NUMBER FROM dbo.output_table ORDER BY MSG_TIME_STAMP desc",conn)
    previous_heats = list(previous_heats['HEAT_NUMBER'])
    previous_heats = [str(x) for x in previous_heats]

    # we will only run for any new heats that have been found
    latest_heats = sample_latest['HEAT_NUMBER'].unique()
    run_heats = [x for x in latest_heats if x not in previous_heats]

    return run_heats
```

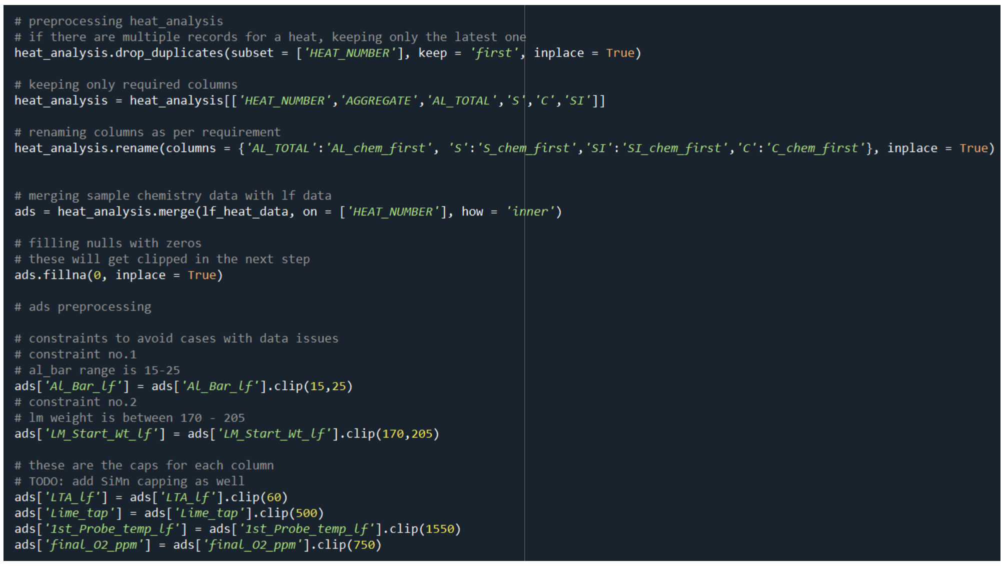
iv. **Main Function**
   a. Connect to JSW Database with the following credentials
      'Driver={SQL Server};'
      'Server=STEELDNA;'
      'Database=DeS;'
      'UID=sa;'
      'PWD=admin@123;'
      'Trusted_Connection=no;'
   b. Call latest_heats function to identify heats to run model on
   c. Get heat_analysis for the latest_heats
   d. Get lf_data for the latest_heats


```
def main():
    """
    Main function that is called whenever predictions are to be made
    """
    ######################### Connecting to SQL Server ###############################

    conn = pyodbc.connect('Driver={SQL Server};'
                          'Server=STEELDNA;'
                          'Database=DeS;'
                          'UID=sa;'
                          'PWD=admin@123;'
                          'Trusted_Connection=no;')

    cursor = conn.cursor()

    ######################### checking if model needs to be run #####################

    # getting list of heats for which model has not yet been run
    run_heats = find_latest_heats(conn)
    print(run_heats)

    # if not empty, model is to be run
    if len(run_heats) != 0:

        # importing heat_analysis
        heat_analysis = pd.read_sql("select TOP(1000) * from heat_analysis order by MSG_TIME_STAMP desc", conn)
#         heat_analysis = pd.read_sql("select * from heat_analysis where HEAT_NUMBER in ('22200172','22400185','22200161') orde

        heat_analysis = heat_analysis[heat_analysis['HEAT_NUMBER'].isin(run_heats)]
        # importing lf_heat_data
        lf_heat_data = pd.read_sql("select TOP(1000) * from lf_heat_data order by MSG_TIME_STAMP desc", conn)
        lf_heat_data = pd.read_sql("select TOP(1000) * from lf_heat_data where ALBAR <> 0 order by MSG_TIME_STAMP desc", conn)
```

v. **Preprocessing**
   a. Create Backup of lf_heat data to later fetch previous grade details for each heat.
   b. Get des.grade_mapping for finding type of grade
   c. lf_data processing
      i. Final O2 = Tap O2 + O2 after celox
      ii. Al Bar = Al Bar / 19
      iii. Getting max value of temp, al_bar, lm_start_wt, lta_lf, lime_tap, simn_tap, final_o2 for each heat and first grade


```
prev_grade_flag=lf_heat_data[['HEAT_NUMBER','PREV_HEAT_NUMBER']].drop_duplicates(subset=['HEAT_NUMBER'])
prev_grade_flag=prev_grade_flag.merge(lf_heat_backup,on=['PREV_HEAT_NUMBER'], how='left')

# importing the grade mapping fact table
grade_mapping = pd.read_sql("select * from grade_mapping", conn)

######################### data processing ###############################

  print(heat_analysis)
  print(lf_heat_data)

# calculating required columns
lf_heat_data['final_O2_ppm'] = lf_heat_data['TAP_O2'] + lf_heat_data['O2AFTERCELOX']

# converting ALBAR to tons
lf_heat_data['ALBAR'] = lf_heat_data['ALBAR']/19

# renaming the required columns
lf_heat_data.rename(columns = {
    'GRADE_TYPE':'GRADE',
    'LM_START_WT':'LM_Start_Wt_lf',
    'FIRSTMEASTEMP':'1st_Probe_temp_lf',
    'ALBAR':'Al_Bar_lf',
    'LIME':'Lime_tap',
    'SIMN':'SiMn_tap',
    'LTA':'LTA_lf'
}, inplace = True)

# keeping only the required rows
lf_heat_data = lf_heat_data.groupby(['HEAT_NUMBER','STATION']).agg({'MSG_TIME_STAMP':'max','GRADE': 'first','1st_Probe_temp_lf':'max', 'Al_Bar_lf':'max','LM_Start_Wt_lf
lf_heat_data.reset_index(inplace = True)
lf_heat_data.sort_values(by = ['MSG_TIME_STAMP'], ascending = False, inplace = True)
lf_heat_data.drop_duplicates(subset = ['HEAT_NUMBER'], keep = 'first', inplace = True)
```

   d. heat_analysis data cleaning
   e. merging heat_analysis, lf_heat_data and grade_mapping
   f. replacing outlier values for each parameter with their lower/upper bounds



```
# preprocessing heat_analysis
# if there are multiple records for a heat, keeping only the latest one
heat_analysis.drop_duplicates(subset = ['HEAT_NUMBER'], keep = 'first', inplace = True)

# keeping only required columns
heat_analysis = heat_analysis[['HEAT_NUMBER','AGGREGATE','AL_TOTAL','S','C','SI']]

# renaming columns as per requirement
heat_analysis.rename(columns = {'AL_TOTAL':'AL_chem_first', 'S':'S_chem_first','SI':'SI_chem_first','C':'C_chem_first'}, inplace = True)


# merging sample chemistry data with lf data
ads = heat_analysis.merge(lf_heat_data, on = ['HEAT_NUMBER'], how = 'inner')

# filling nulls with zeros
# these will get clipped in the next step
ads.fillna(0, inplace = True)

# ads preprocessing

# constraints to avoid cases with data issues
# constraint no.1
# al_bar range is 15-25
ads['Al_Bar_lf'] = ads['Al_Bar_lf'].clip(15,25)
# constraint no.2
# lm weight is between 170 - 205
ads['LM_Start_Wt_lf'] = ads['LM_Start_Wt_lf'].clip(170,205)

# these are the caps for each column
# TODO: add SiMn capping as well
ads['LTA_lf'] = ads['LTA_lf'].clip(60)
ads['Lime_tap'] = ads['Lime_tap'].clip(500)
ads['1st_Probe_temp_lf'] = ads['1st_Probe_temp_lf'].clip(1550)
ads['final_O2_ppm'] = ads['final_O2_ppm'].clip(750)
```

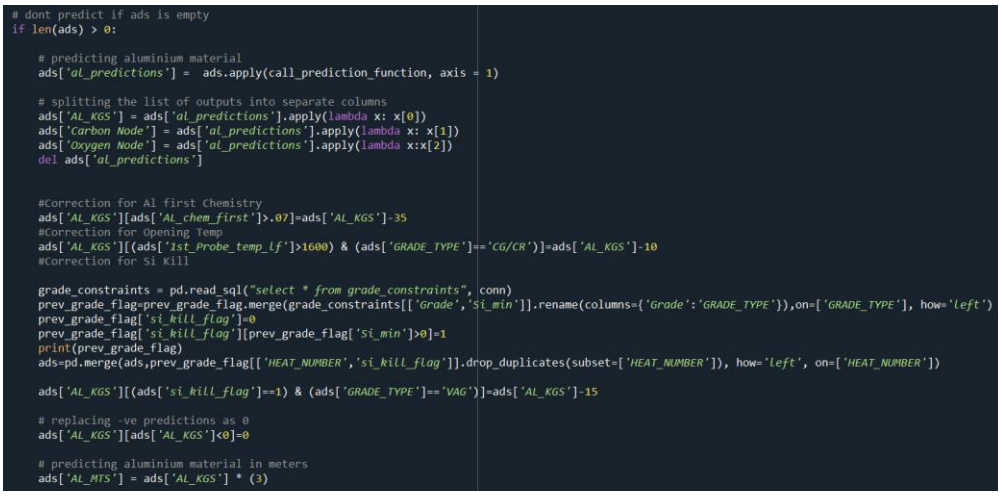
vi. Predicting Aluminium

a. Calling Prediction function to calculate Al required based on node rules and equations.

b. Applying Emperical corrections based on business logic and data trends

   i. If first Chemistry of Aluminium is > .07
   ii. If 1st Probe Temp of LF > 1600 for CG/CR Grades
   iii. Checking Si Kill based on Previous grade of the shell
   iv. Updating negative results to 0

c. Calculating Al (Meters) as Al (Kg) *3



```
# dont predict if ads is empty
if len(ads) > 0:

    # predicting aluminium material
    ads['al_predictions'] =  ads.apply(call_prediction_function, axis = 1)

    # splitting the list of outputs into separate columns
    ads['AL_KGS'] = ads['al_predictions'].apply(lambda x: x[0])
    ads['Carbon Node'] = ads['al_predictions'].apply(lambda x: x[1])
    ads['Oxygen Node'] = ads['al_predictions'].apply(lambda x:x[2])
    del ads['al_predictions']


    #Correction for Al first Chemistry
    ads['AL_KGS'][ads['AL_chem_first']>.07]=ads['AL_KGS']-35
    #Correction for Opening Temp
    ads['AL_KGS'][(ads['1st_Probe_temp_lf']>1600) & (ads['GRADE_TYPE']=='CG/CR')]=ads['AL_KGS']-10
    #Correction for Si Kill

    grade_constraints = pd.read_sql("select * from grade_constraints", conn)
    prev_grade_flag=prev_grade_flag.merge(grade_constraints[['Grade','Si_min']].rename(columns={'Grade':'GRADE_TYPE'}),on=['GRADE_TYPE'], how='left')
    prev_grade_flag['si_kill_flag']=0
    prev_grade_flag['si_kill_flag'][prev_grade_flag['Si_min']>0]=1
    print(prev_grade_flag)
    ads=pd.merge(ads,prev_grade_flag[['HEAT_NUMBER','si_kill_flag']].drop_duplicates(subset=['HEAT_NUMBER']), how='left', on=['HEAT_NUMBER'])

    ads['AL_KGS'][(ads['si_kill_flag']==1) & (ads['GRADE_TYPE']=='VAG')]=ads['AL_KGS']-15

    # replacing -ve predictions as 0
    ads['AL_KGS'][ads['AL_KGS']<0]=0

    # predicting aluminium material in meters
    ads['AL_MTS'] = ads['AL_KGS'] * (3)
```

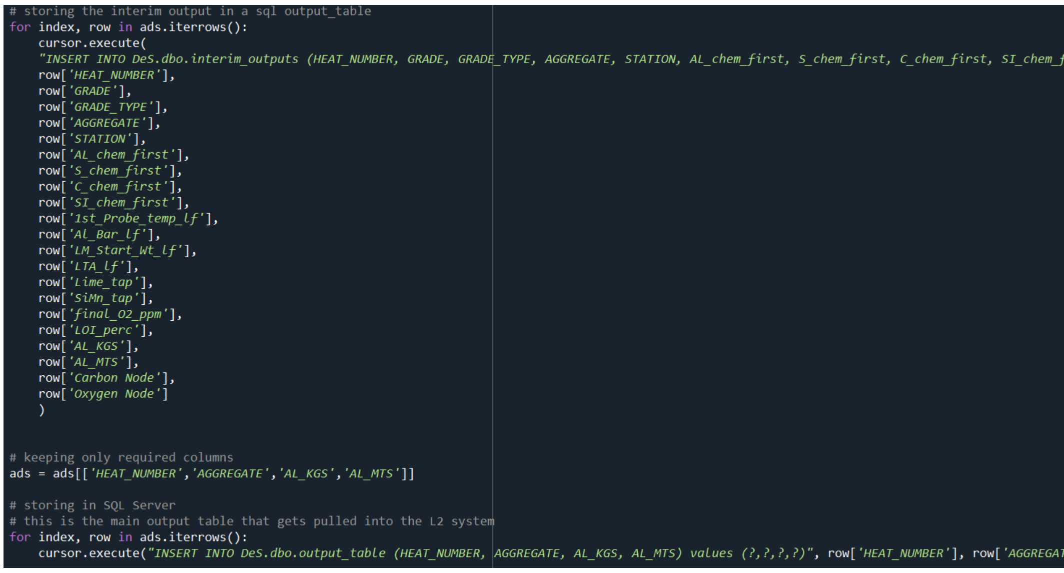
vii. Storing Results in Database

Storing the generated results in

a. des.interim_outputs
   This table stores all relavent information such as input data, node etc.

b. des.output_table

This table only stores Heat Number, Shell, Al (kgs), Al (Mtr) to present on L2 Terminal

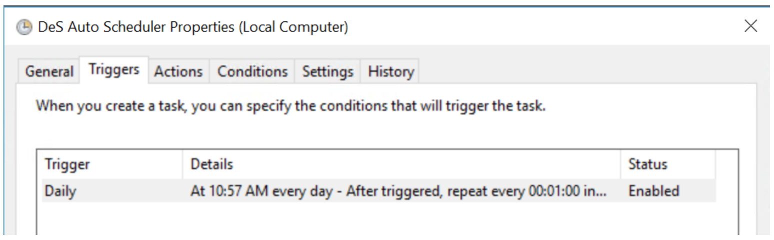


```
# storing the interim output in a sql output_table
for index, row in ads.iterrows():
    cursor.execute(
    "INSERT INTO DeS.dbo.interim_outputs (HEAT_NUMBER, GRADE, GRADE_TYPE, AGGREGATE, STATION, AL_chem_first, S_chem_first, C_chem_first, SI_chem_f
    row['HEAT_NUMBER'],
    row['GRADE'],
    row['GRADE_TYPE'],
    row['AGGREGATE'],
    row['STATION'],
    row['AL_chem_first'],
    row['S_chem_first'],
    row['C_chem_first'],
    row['SI_chem_first'],
    row['1st_Probe_temp_lf'],
    row['Al_Bar_lf'],
    row['LM_Start_Wt_lf'],
    row['LTA_lf'],
    row['Lime_tap'],
    row['SiMn_tap'],
    row['final_O2_ppm'],
    row['LOI_perc'],
    row['AL_KGS'],
    row['AL_MTS'],
    row['Carbon Node'],
    row['Oxygen Node']
    )


# keeping only required columns
ads = ads[['HEAT_NUMBER','AGGREGATE','AL_KGS','AL_MTS']]

# storing in SQL Server
# this is the main output table that gets pulled into the L2 system
for index, row in ads.iterrows():
    cursor.execute("INSERT INTO DeS.dbo.output_table (HEAT_NUMBER, AGGREGATE, AL_KGS, AL_MTS) values (?,?,?,?)", row['HEAT_NUMBER'], row['AGGREGAT
```

2.5 Scheduler (File: C:\DeS\Codes\scheduler.py)

This scheduler calls the al_prediction code in every 10 seconds to check for new heats.

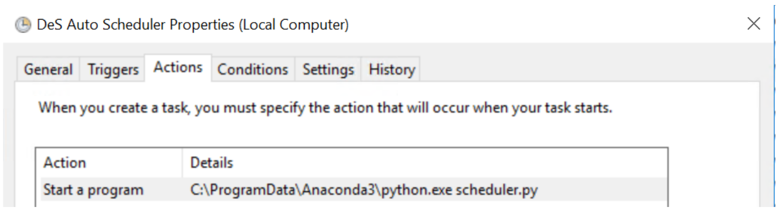
To run the scheduler.py code, we run Task every 1 minute



DeS Auto Scheduler Properties (Local Computer)

When you create a task, you can specify the conditions that will trigger the task.

Trigger	Details	Status
Daily	At 10:57 AM every day - After triggered, repeat every 00:01:00 in...	Enabled



DeS Auto Scheduler Properties (Local Computer)

When you create a task, you must specify the action that will occur when your task starts.

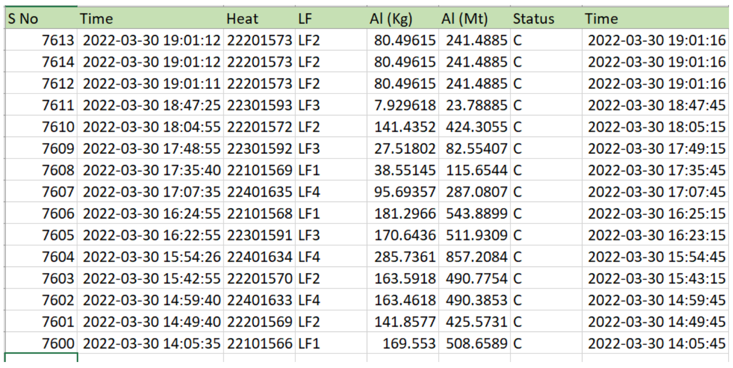
Action	Details
Start a program	C:\ProgramData\Anaconda3\python.exe scheduler.py

2.6 New Grades (Database entry des.dbo.grade_mapping)

Add any New Grade and Grade Type in the database table grade_mapping

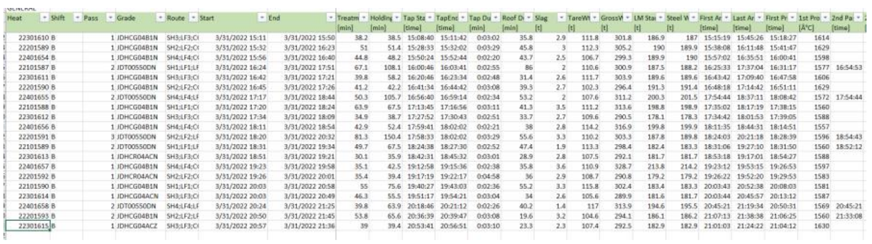
3 Generating Compliance (File name: C:\DeS\Codes\DeS Compliance\ DES Compliance Report.xlsx)

3.1 Paste weekly model output in sheet Model op under the existing data



S No	Time	Heat	LF	Al (Kg)	Al (Mt)	Status	Time
7613	2022-03-30 19:01:12	22201573	LF2	80.49615	241.4885	C	2022-03-30 19:01:16
7614	2022-03-30 19:01:12	22201573	LF2	80.49615	241.4885	C	2022-03-30 19:01:16
7612	2022-03-30 19:01:11	22201573	LF2	80.49615	241.4885	C	2022-03-30 19:01:16
7611	2022-03-30 18:47:25	22301593	LF3	7.929618	23.78885	C	2022-03-30 18:47:45
7610	2022-03-30 18:04:55	22201572	LF2	141.4352	424.3055	C	2022-03-30 18:05:15
7609	2022-03-30 17:48:55	22301592	LF3	27.51802	82.55407	C	2022-03-30 17:49:15
7608	2022-03-30 17:35:40	22101569	LF1	38.55145	115.6544	C	2022-03-30 17:35:45
7607	2022-03-30 17:07:35	22401635	LF4	95.69357	287.0807	C	2022-03-30 17:07:45
7606	2022-03-30 16:24:55	22101568	LF1	181.2966	543.8899	C	2022-03-30 16:25:15
7605	2022-03-30 16:22:55	22301591	LF3	170.6436	511.9309	C	2022-03-30 16:23:15
7604	2022-03-30 15:54:26	22401634	LF4	285.7361	857.2084	C	2022-03-30 15:54:45
7603	2022-03-30 15:42:55	22201570	LF2	163.5918	490.7754	C	2022-03-30 15:43:15
7602	2022-03-30 14:59:40	22401633	LF4	163.4618	490.3853	C	2022-03-30 14:59:45
7601	2022-03-30 14:49:40	22201569	LF2	141.8577	425.5731	C	2022-03-30 14:49:45
7600	2022-03-30 14:05:35	22101566	LF1	169.553	508.6589	C	2022-03-30 14:05:45

3.2 Collate weekly LF Csv Data and paste it under existing data in sheet DES Compliance_v2.2

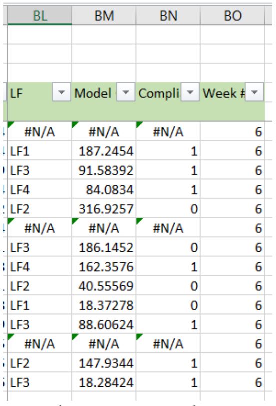


Heat	Shift	Pass	Grade	Route	Start	End	Treatm [min]	Holding [min]	Tap Sta [time]	TapEnd [time]
22301610	B	1	JDHCG04B1N	SH3;LF3;C	3/31/2022 15:11	3/31/2022 15:50	38.2	38.5	15:08:40	15:11:42
22201589	B	1	JDHCG04B1N	SH2;LF2;C	3/31/2022 15:32	3/31/2022 16:23	51	51.4	15:28:33	15:32:02
22401654	B	1	JDHCG04B1N	SH4;LF4;C	3/31/2022 15:56	3/31/2022 16:40	44.8	48.2	15:50:24	15:52:44
22101587	B	2	JDT00550DN	SH1;LF1;LF	3/31/2022 16:24	3/31/2022 17:51	67.1	108.1	16:00:46	16:03:41
22301611	B	1	JDHCG04B1N	SH3;LF3;C	3/31/2022 16:42	3/31/2022 17:21	39.8	58.2	16:20:46	16:23:34
22201590	B	1	JDHCG04B1N	SH2;LF2;C	3/31/2022 16:45	3/31/2022 17:26	41.2	42.2	16:41:34	16:44:42
22401655	B	2	JDT00550DN	SH4;LF4;LF	3/31/2022 17:17	3/31/2022 18:44	50.3	105.7	16:56:40	16:59:14
22101588	B	1	JDHCG04B1N	SH1;LF1;C	3/31/2022 17:20	3/31/2022 18:24	63.9	67.5	17:13:45	17:16:56
22301612	B	1	JDHCG04B1N	SH3;LF3;C	3/31/2022 17:34	3/31/2022 18:09	34.9	38.7	17:27:52	17:30:43
22401656	B	1	JDHCG04B1N	SH4;LF4;C	3/31/2022 18:11	3/31/2022 18:54	42.9	52.4	17:59:41	18:02:02
22201591	B	3	JDT00550DN	SH2;LF2;LF	3/31/2022 18:20	3/31/2022 20:32	81.3	150.4	17:58:33	18:02:02
22101589	B	2	JDT00550DN	SH1;LF1;LF	3/31/2022 18:31	3/31/2022 19:34	49.7	67.5	18:24:38	18:27:30
22301613	B	1	JDHCR04ACN	SH3;LF3;C	3/31/2022 18:51	3/31/2022 19:21	30.1	35.9	18:42:31	18:45:32
22401657	B	1	JDHCG04B1N	SH4;LF4;C	3/31/2022 19:23	3/31/2022 19:58	35.1	42.5	19:12:58	19:15:36
22201592	B	1	JDHCR04ACN	SH2;LF2;C	3/31/2022 19:26	3/31/2022 20:01	35.4	39.4	19:17:19	19:22:17
22101590	B	1	JDHCG04B1N	SH1;LF1;C	3/31/2022 20:03	3/31/2022 20:58	55	75.6	19:40:27	19:43:03
22301614	B	1	JDHCG04ACN	SH3;LF3;C	3/31/2022 20:03	3/31/2022 20:49	46.3	55.5	19:51:17	19:54:21
22401658	B	2	JDT00550DN	SH4;LF4;LF	3/31/2022 20:24	3/31/2022 21:25	39.8	63.9	20:18:46	20:21:12
22201593	B	2	JDHCG04B1N	SH2;LF2;LF	3/31/2022 20:50	3/31/2022 21:45	53.8	65.6	20:36:39	20:39:47
22301615	B	1	JDHCG04ACZ	SH3;LF3;C	3/31/2022 20:57	3/31/2022 21:36	39	39.4	20:53:41	20:56:51

Heat	Tap Du [min]	Roof D [min]	Slag [t]	TareWt [t]	GrossW [t]	LM Sta [t]	Steel W [t]	First Ar [time]	Last Ar [time]	First Pr [time]	1st Pro [Â°C]	2nd Pa [time]
22301610	0:03:02	35.8	2.9	111.8	301.8	186.9	187	15:15:19	15:45:26	15:18:27	1614	
22201589	0:03:29	45.8	3	112.3	305.2	190	189.9	15:38:08	16:11:48	15:41:47	1629	
22401654	0:02:20	43.7	2.5	106.7	299.3	189.9	190	15:57:02	16:35:51	16:00:41	1598	
22101587	0:02:55	86	2	110.6	300.9	187.5	188.2	16:25:33	17:37:04	16:31:17	1577	16:54:53
22301611	0:02:48	31.4	2.6	111.7	303.9	189.6	189.6	16:43:42	17:09:40	16:47:58	1606	
22201590	0:03:08	39.3	2.7	102.3	296.4	191.3	191.4	16:48:18	17:14:42	16:51:11	1629	
22401655	0:02:34	53.2	2	107.6	311.2	200.3	201.5	17:54:44	18:37:11	18:08:42	1572	17:54:44
22101588	0:03:11	41.3	3.5	111.2	313.6	198.8	198.9	17:35:02	18:17:19	17:38:15	1560	
22301612	0:02:51	33.7	2.7	109.6	290.5	178.1	178.3	17:34:42	18:01:53	17:39:05	1588	
22401656	0:02:21	38	2.8	114.2	316.9	199.8	199.9	18:11:35	18:44:31	18:14:51	1557	
22201591	0:03:29	55.6	3.3	110.2	303.3	187.8	189.8	18:24:03	20:21:18	18:28:39	1596	18:54:43
22101589	0:02:52	47.4	1.9	113.3	298.4	182.4	183.3	18:31:06	19:27:10	18:31:50	1560	18:52:12
22301613	0:03:01	28.9	2.8	107.5	292.1	181.7	181.7	18:53:18	19:17:01	18:54:27	1588	
22401657	0:02:38	35.8	3.6	110.9	328.7	213.8	214.2	19:23:12	19:53:15	19:26:53	1597	
22201592	0:04:58	36	2.9	108.7	290.8	179.2	179.2	19:26:22	19:52:20	19:29:53	1583	
22101590	0:02:36	55.2	3.3	115.8	302.4	183.4	183.3	20:03:43	20:52:38	20:08:03	1581	
22301614	0:03:04	34	2.6	105.6	289.9	181.6	181.7	20:03:44	20:45:57	20:13:12	1587	
22401658	0:02:26	40.2	1.4	117	313.9	194.6	195.5	20:45:21	21:19:34	20:50:31	1569	20:45:21
22201593	0:03:08	19.6	3.2	104.6	294.1	186.1	186.2	21:07:13	21:38:38	21:06:25	1560	21:33:08
22301615	0:03:10	23.3	2.3	107.4	292.5	182.9	182.9	21:01:03	21:24:22	21:04:12	1630	

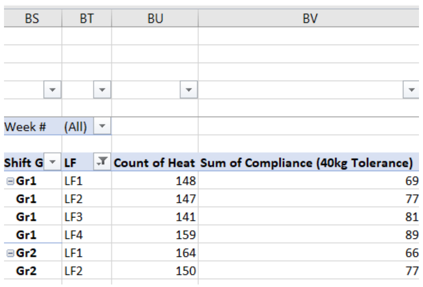
3.3 Calculate compliance -

i. Navigate to column BL
ii. Add LF Name manually
iii. Drag formula in column BM, BN to calculate compliance vs model results
iv. Add week number manually in column BO



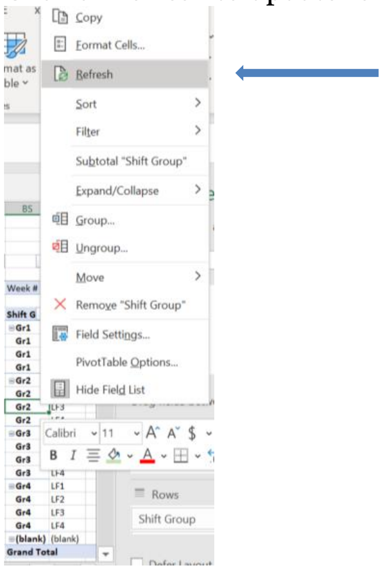
BL	BM	BN	BO
LF	Model	Compli	Week #
#N/A	#N/A	#N/A	6
LF1	187.2454	1	6
LF3	91.58392	1	6
LF4	84.0834	1	6
LF2	316.9257	0	6
#N/A	#N/A	#N/A	6
LF3	186.1452	0	6
LF4	162.3576	1	6
LF2	40.55569	0	6
LF1	18.37278	0	6
LF3	88.60624	1	6
#N/A	#N/A	#N/A	6
LF2	147.9344	1	6
LF3	18.28424	1	6

v. Navigate to column BS and right click on the pivot table.

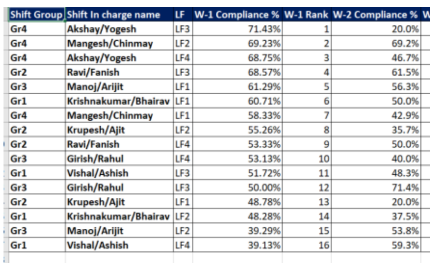


Shift G	LF	Count of Heat	Sum of Compliance (40kg Tolerance)
Gr1	LF1	148	69
Gr1	LF2	147	77
Gr1	LF3	141	81
Gr1	LF4	159	89
Gr2	LF1	164	66
Gr2	LF2	150	77

vi. Click on refresh to update for the current week



3.4 Update results in SMS – a - thon sheet



Shift Group	Shift In charge name	LF	W-1 Compliance %	W-1 Rank	W-2 Compliance %
Gr4	Akshay/Yogesh	LF3	71.43%	1	20.0%
Gr4	Mangesh/Chinmay	LF2	69.23%	2	69.2%
Gr4	Akshay/Yogesh	LF4	68.75%	3	46.7%
Gr2	Ravi/Fanish	LF3	68.57%	4	61.5%
Gr3	Manoj/Arijit	LF1	61.29%	5	56.3%
Gr1	Krishnakumar/Bhairav	LF1	60.71%	6	50.0%
Gr4	Mangesh/Chinmay	LF1	58.33%	7	42.9%
Gr2	Krupesh/Ajit	LF2	55.26%	8	35.7%
Gr2	Ravi/Fanish	LF4	53.33%	9	50.0%
Gr3	Girish/Rahul	LF4	53.13%	10	40.0%
Gr1	Vishal/Ashish	LF3	51.72%	11	48.3%
Gr3	Girish/Rahul	LF3	50.00%	12	71.4%
Gr2	Krupesh/Ajit	LF1	48.78%	13	20.0%
Gr1	Krishnakumar/Bhairav	LF2	48.28%	14	37.5%
Gr3	Manoj/Arijit	LF2	39.29%	15	53.8%
Gr1	Vishal/Ashish	LF4	39.13%	16	59.3%

4 Retraining Model

To build Machine Learning model, following parameters are considered to impact the output Aluminium –

i. final_O2_ppm
ii. LM_Start_Wt_lf
iii. 1st_Probe_temp_lf
iv. Al_Bar_lf
v. Lime_tap
vi. AL_chem_first
vii. S_chem_first

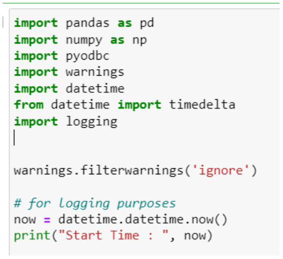
4.1 Create Input Data

Collate Heat wise data from the following data sources in (Des Input.xlsx)-

a. Heat Analysis
b. Lf Data
c. Grade Mapping
d. Actual Al Added

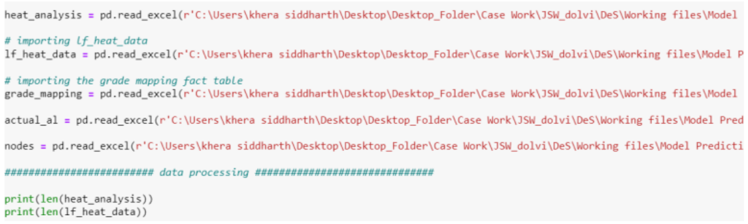
4.2 Run file ‘DeS Model vag ‘ and ‘DeS Model cg/cr ‘ Model training folder

1. Import required libraries



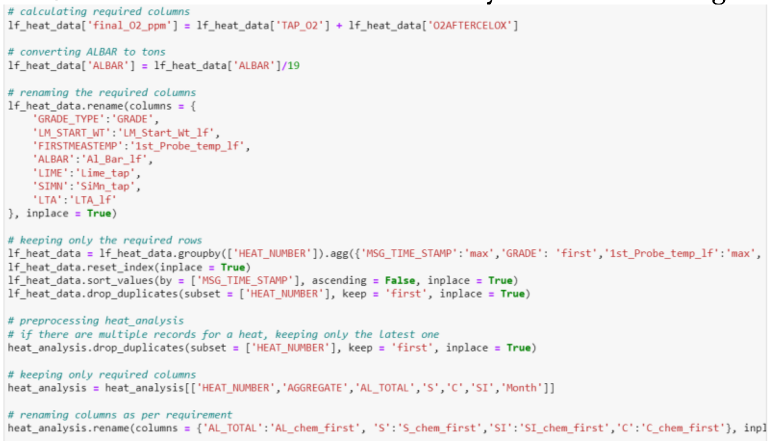
```
import pandas as pd
import numpy as np
import pyodbc
import warnings
import datetime
from datetime import timedelta
import logging


warnings.filterwarnings('ignore')

# for logging purposes
now = datetime.datetime.now()
print("Start Time : ", now)
```

2. Upload Input Data



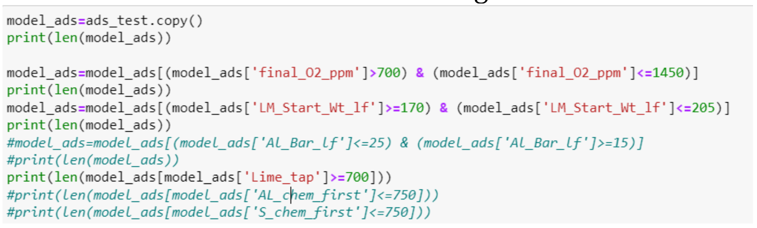
```
heat_analysis = pd.read_excel(r'C:\Users\khera siddharth\Desktop\Desktop_Folder\Case Work\JSW_dolvi\DeS\Working files\Model

# importing lf_heat_data
lf_heat_data = pd.read_excel(r'C:\Users\khera siddharth\Desktop\Desktop_Folder\Case Work\JSW_dolvi\DeS\Working files\Model P

# importing the grade mapping fact table
grade_mapping = pd.read_excel(r'C:\Users\khera siddharth\Desktop\Desktop_Folder\Case Work\JSW_dolvi\DeS\Working files\Model

actual_al = pd.read_excel(r'C:\Users\khera siddharth\Desktop\Desktop_Folder\Case Work\JSW_dolvi\DeS\Working files\Model Pred

nodes = pd.read_excel(r'C:\Users\khera siddharth\Desktop\Desktop_Folder\Case Work\JSW_dolvi\DeS\Working files\Model Predicti

######################### data processing ##############################

print(len(heat_analysis))
print(len(lf_heat_data))
```

3. Process Lf Data and heat analysis data and merge



```
# calculating required columns
lf_heat_data['final_O2_ppm'] = lf_heat_data['TAP_O2'] + lf_heat_data['O2AFTERCELOX']

# converting ALBAR to tons
lf_heat_data['ALBAR'] = lf_heat_data['ALBAR']/19

# renaming the required columns
lf_heat_data.rename(columns = {
    'GRADE_TYPE':'GRADE',
    'LM_START_WT':'LM_Start_Wt_lf',
    'FIRSTMEASTEMP':'1st_Probe_temp_lf',
    'ALBAR':'Al_Bar_lf',
    'LIME':'Lime_tap',
    'SIMN':'SiMn_tap',
    'LTA':'LTA_lf'
}, inplace = True)

# keeping only the required rows
lf_heat_data = lf_heat_data.groupby(['HEAT_NUMBER']).agg({'MSG_TIME_STAMP':'max','GRADE': 'first','1st_Probe_temp_lf':'max',
lf_heat_data.reset_index(inplace = True)
lf_heat_data.sort_values(by = ['MSG_TIME_STAMP'], ascending = False, inplace = True)
lf_heat_data.drop_duplicates(subset = ['HEAT_NUMBER'], keep = 'first', inplace = True)

# preprocessing heat_analysis
# if there are multiple records for a heat, keeping only the latest one
heat_analysis.drop_duplicates(subset = ['HEAT_NUMBER'], keep = 'first', inplace = True)

# keeping only required columns
heat_analysis = heat_analysis[['HEAT_NUMBER','AGGREGATE','AL_TOTAL','S','C','SI','Month']]

# renaming columns as per requirement
heat_analysis.rename(columns = {'AL_TOTAL':'AL_chem_first', 'S':'S_chem_first','SI':'SI_chem_first','C':'C_chem_first'}, inpl
```

4. Outlier Treatment – removing heats which lie outside the range of bounds



```
model_ads=ads_test.copy()
print(len(model_ads))

model_ads=model_ads[(model_ads['final_O2_ppm']>700) & (model_ads['final_O2_ppm']<=1450)]
print(len(model_ads))
model_ads=model_ads[(model_ads['LM_Start_Wt_lf']>=170) & (model_ads['LM_Start_Wt_lf']<=205)]
print(len(model_ads))
#model_ads=model_ads[(model_ads['Al_Bar_lf']<=25) & (model_ads['Al_Bar_lf']>=15)]
#print(len(model_ads))
print(len(model_ads[model_ads['Lime_tap']>=700]))
#print(len(model_ads[model_ads['AL_chem_first']<=750]))
#print(len(model_ads[model_ads['S_chem_first']<=750]))
```

5. Data cleansing and replacing missing values from data

```python
def outlier_treatment(df,col_list,lv_imp,uv_imp):
    """

    Intended to treat outlier values based on defined thresholds lv and uv
    returns : dataframe with values treated for outliers
    """
    for col,lv,uv in zip(col_list,lv_imp,uv_imp):
        df[col+'_treated'] = [uv if x>uv else x for x in df[col]]
        df[col+'_treated'] = [lv if x<lv else x for x in df[col+'_treated']]

        #print(df[col+'_treated'])
    return df

def impute_missing(df,cols=[],simple_impute=True,impute_value=0):
    """

    Intended to impute missing values for the specified columns by the specified value
    returns : dataframe with imputed values
    """
    if simple_impute == 1:
            if len(cols)>0:
                df[cols] = df[cols].fillna(value=impute_value)
            else:
                df.fillna(impute_value,inplace=True)
    else:
        df.fillna(df[cols].median(),inplace=True)
        print("Column and Median to be replace : ",cols, df[cols].median())

    return df
```

6. Running Random Forest Machine Learning Algorithm to generate nodes

```python
model_ads_train_test_df=model_ads.copy()

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
# Split the training set to train and evaluate samples
X_train, X_test, y_train, y_test = train_test_split(model_ads_train_test_df[features],
                                                    model_ads_train_test_df[target],
                                                    test_size=0.25,
                                                    random_state=97)

#Merging the datasets
train_df = pd.concat([X_train,y_train], axis = 1).reset_index(drop=True)
test_df = pd.concat([X_test,y_test], axis = 1).reset_index(drop=True)

def train_decision_tree(df_features,df_target,n_=200,leaf_=10,split_=25,md_=4, random_state = 97):

    # Initialise dataframe
    weights=pd.DataFrame(columns=['Variable','Importance_gain','iteration','rand_index'])

    # Instantiate the model with required parameters
    model_=DecisionTreeRegressor(max_depth=md_, random_state=random_state,criterion='mse',min_samples_leaf=leaf_,
                                min_samples_split=split_)

    print(model_)
    model_.fit(df_features,df_target)
    weights = pd.DataFrame(model_.feature_importances_,
                               index = df_features.columns,
                                columns=['importance']).sort_values('importance',ascending=False)
    return model_,weights
```

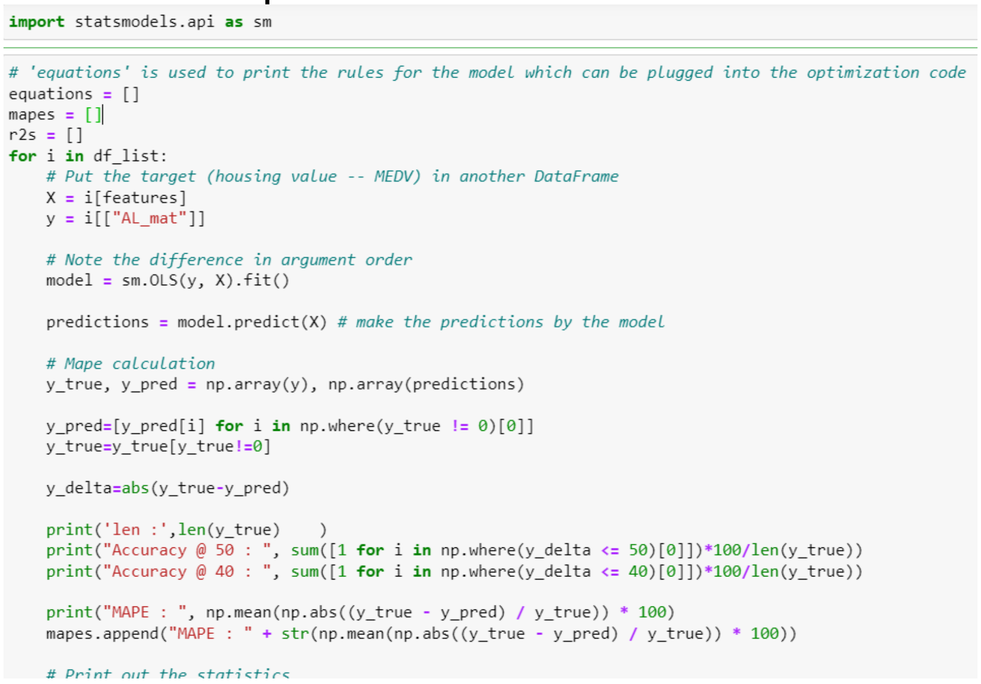
7. Extracting rules in a readable form from Random forest model

```python
def get_rules(tree, feature_names, class_names):
    tree_ = tree.tree_
    feature_name = [
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined!"
        for i in tree_.feature
    ]

    paths = []
    path = []

    def recurse(node, path, paths):

        if tree_.feature[node] != _tree.TREE_UNDEFINED:
            name = feature_name[node]
            threshold = tree_.threshold[node]
            p1, p2 = list(path), list(path)
            p1 += [f"({name} <= {np.round(threshold, 3)})"]
            recurse(tree_.children_left[node], p1, paths)
            p2 += [f"({name} > {np.round(threshold, 3)})"]
            recurse(tree_.children_right[node], p2, paths)
        else:
            path += [(tree_.value[node], tree_.n_node_samples[node])]
            paths += [path]

    recurse(0, path, paths)

    # sort by samples count
    samples_count = [p[-1][1] for p in paths]
    ii = list(np.argsort(samples_count))
    paths = [paths[i] for i in reversed(ii)]
```

8. Copy and Paste the rules in the next step of the code to generate relavent equations for each node

```
df1 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] >
0.046) & (model_ads_train_test_df['Lime_tap'] <= 2927.5) & (model_ads_train_test_df['S_chem_first'] <= 0.027)]
df2 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] <=
0.046) & (model_ads_train_test_df['S_chem_first'] > 0.02) & (model_ads_train_test_df['final_O2_ppm'] > 980.5)]
df3 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first'] <
= 0.032) & (model_ads_train_test_df['S_chem_first'] > 0.027) & (model_ads_train_test_df['S_chem_first'] <= 0.035)]
df4 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] <=
0.046) & (model_ads_train_test_df['S_chem_first'] <= 0.02) & (model_ads_train_test_df['Lime_tap'] <= 3088.5)]
df5 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first'] <
= 0.032) & (model_ads_train_test_df['S_chem_first'] <= 0.027) & (model_ads_train_test_df['final_O2_ppm'] <= 1144.5)]
df6 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first'] <
= 0.032) & (model_ads_train_test_df['S_chem_first'] > 0.027) & (model_ads_train_test_df['S_chem_first'] > 0.035)]
df7 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] >
0.046) & (model_ads_train_test_df['Lime_tap'] <= 2927.5) & (model_ads_train_test_df['S_chem_first'] > 0.027)]
df8 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first'] >
0.032) & (model_ads_train_test_df['Lime_tap'] > 995.5) & (model_ads_train_test_df['S_chem_first'] <= 0.029)]
df9 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] <=
0.046) & (model_ads_train_test_df['S_chem_first'] > 0.02) & (model_ads_train_test_df['final_O2_ppm'] <= 980.5)]
df10 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first']
<= 0.032) & (model_ads_train_test_df['S_chem_first'] <= 0.027) & (model_ads_train_test_df['final_O2_ppm'] > 1144.5)]
df11 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first']
> 0.032) & (model_ads_train_test_df['Lime_tap'] <= 995.5) & (model_ads_train_test_df['LM_Start_Wt_lf'] > 185.5)]
df12 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first']
> 0.032) & (model_ads_train_test_df['Lime_tap'] > 995.5) & (model_ads_train_test_df['S_chem_first'] > 0.029)]
df13 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] <= 15.026) & (model_ads_train_test_df['AL_chem_first']
> 0.032) & (model_ads_train_test_df['Lime_tap'] <= 995.5) & (model_ads_train_test_df['LM_Start_Wt_lf'] <= 185.5)]
df14 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] >
0.046) & (model_ads_train_test_df['AL_chem_first'] > 0.046) & (model_ads_train_test_df['Lime_tap'] > 2927.5)]
df15 = model_ads_train_test_df[(model_ads_train_test_df['Al_Bar_lf'] > 15.026) & (model_ads_train_test_df['AL_chem_first'] <
= 0.046) & (model_ads_train_test_df['S_chem_first'] <= 0.02) & (model_ads_train_test_df['Lime_tap'] > 3088.5)]
```

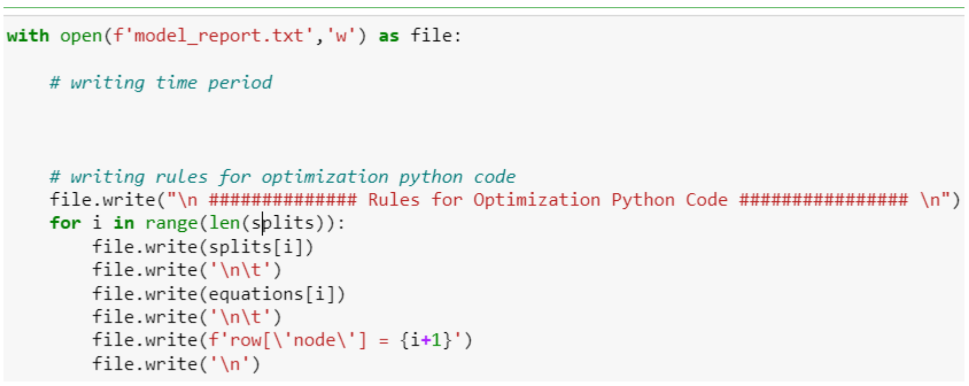
9. Generate equations for each of the nodes



```
import statsmodels.api as sm
```

```
# 'equations' is used to print the rules for the model which can be plugged into the optimization code
equations = []
mapes = []
r2s = []
for i in df_list:
    # Put the target (housing value -- MEDV) in another DataFrame
    X = i[features]
    y = i[["AL_mat"]]

    # Note the difference in argument order
    model = sm.OLS(y, X).fit()

    predictions = model.predict(X) # make the predictions by the model

    # Mape calculation
    y_true, y_pred = np.array(y), np.array(predictions)

    y_pred=[y_pred[i] for i in np.where(y_true != 0)[0]]
    y_true=y_true[y_true!=0]

    y_delta=abs(y_true-y_pred)

    print('len :',len(y_true)     )
    print("Accuracy @ 50 : ", sum([1 for i in np.where(y_delta <= 50)[0]])*100/len(y_true))
    print("Accuracy @ 40 : ", sum([1 for i in np.where(y_delta <= 40)[0]])*100/len(y_true))

    print("MAPE : ", np.mean(np.abs((y_true - y_pred) / y_true)) * 100)
    mapes.append("MAPE : " + str(np.mean(np.abs((y_true - y_pred) / y_true)) * 100))

    # Print out the statistics
```

10. Save the rules in a notepad and from there, Copy and Paste the generated equations in node_rules.py



```
with open(f'model_report.txt','w') as file:

    # writing time period



    # writing rules for optimization python code
    file.write("\n ############## Rules for Optimization Python Code ################ \n")
    for i in range(len(splits)):
        file.write(splits[i])
        file.write('\n\t')
        file.write(equations[i])
        file.write('\n\t')
        file.write(f'row[\'node\'] = {i+1}')
        file.write('\n')
```

11. Run the process for cg/cr grades separately