Università degli Studi di Napoli
Federico II

Machine Learning Minicontest #2

Anna Lamboglia

What will we see?

Topics of this presentation

01 Problem

02 Dataset Study

03 Preprocessing

04 Feature Selection

05 Algorithm Choice

06 Evaluation

07 Testing

Domain of interest

Background

Modern society strongly relies on oil.

A possible solution is the use of biofuels, namely fuels compatible with machinery designed for fossil fuels but obtained by processing other sources instead of oil. Among all, raw feedstocks processing is a promising approach to produce biofuels from raw vegetable wastes

Problem

The high variability of the transformation efficiency, which is strongly affected by factors associated with the characteristics of the source material, pre-processing stages, storage type, manipulation etc.

Goal

The aim of the second Machine Learning Mini-Contest (MC2) for the academic year 2021/2022 is to *predict the Oxygen/Carbon ratio (numeric prediction) for a given raw feedstock sample described by its properties and characteristics*.

Dataset Study

Material_1	Type_1	Pretreatment_1	Moisture_1	Volatiles_1	Fixed Carbon_1	Ash_1	HHV, MJ/Kg_1	C_1	H_1	N_1	O_1	S_1	Cl_1	H/C(0)_1	O/C(0)_1	Cellulose_1	Hemicellulose_1	Lignin_1	Ex
japonica alga	algae	acid_washed	5.03	NaN	NaN	7.32	14.27	43.76	6.0	2.76	55.38	NaN	NaN	0.137111517367459	1.26553930530165	NaN	NaN	NaN	
bamboo	agricultural_waste	none	7.3	90.9	0.1	1.7	17.7	46.9	5.85	0.21	47.02	0.02	NaN	0.124733475479744	1.00255863539446	41.0	26.5	25.3	
japonica alga	algae	none	6.9	NaN	NaN	20.21	16.96	41.22	7.73	1.17	49.88	NaN	NaN	0.18753032508491	1.21009218825813	NaN	NaN	NaN	
beech wood	hardwood	none	8.0	NaN	NaN	0.54	NaN	48.44	5.83	NaN	45.73	NaN	NaN	0.120355078447564	0.944054500412882	NaN	NaN	NaN	
rice straw	agricultural_waste	none	10.8	66.89	14.57	7.56	NaN	39.98	2.45	4.43	52.61	0.53	NaN	0.0612806403201601	1.31590795397699	NaN	NaN	NaN	
mallee	NaN	none	NaN	81.9	17.6	0.5	NaN	48.4	6.3	0.1	45.2	NaN	NaN	0.130165289256198	0.933884297520661	NaN	NaN	NaN	
beech wood	hardwood	none	NaN	NaN	NaN	2.8	18.2	46.3	6.3	0.1	36.1	NaN	NaN	0.136069114470842	0.779697624190065	NaN	NaN	NaN	
olive husk	industrial waste	none	3.66	NaN	NaN	1.88	21.57	51.8	6.83	1.32	40.05	NaN	NaN	0.131853281853282	0.773166023166023	NaN	NaN	NaN	
napier grass	grass	none	75.3	81.5	16.7	1.75	18.1	48.6	6.01	0.99	44.1	0.32	NaN	0.123662551440329	0.907407407407407	38.8	19.8	27.0	
sugarcane bagasse	agricultural_waste	none	16.07	65.0	14.59	4.34	18.61	58.14	6.05	0.69	34.57	0.19	0.36	0.10405916752666	0.594599243206054	NaN	NaN	NaN	
beech wood	hardwood	none	6.0	85.3	14.3	0.4	NaN	50.8	5.9	0.3	42.9	0.02	NaN	0.116141732283465	0.844488188976378	NaN	NaN	NaN	

```
[ ]  #Look at the data types whitch columns need to be encoded
     df.dtypes
```

```
Material_1             object
Type_1                 object
Pretreatment_1         object
Moisture_1            float64
Volatiles_1           float64
Fixed Carbon_1        float64
Ash_1                 float64
HHV, MJ/Kg_1          float64
C_1                   float64
H_1                   float64
N_1                   float64
O_1                   float64
S_1                   float64
Cl_1                  float64
H/C(0)_1              float64
O/C(0)_1              float64
Cellulose_1           float64
Hemicellulose_1       float64
Lignin_1              float64
Extractives_1         float64
Particle Size, mm_1    object
O/C oil               float64
dtype: object
```

Encoding

```
     df.shape
```

```
(177, 22)
```

```
[7]  #Count the number of empty values of each columns in dataset
     df.isna().sum()
```

```
Material_1              0
Type_1                  8
Pretreatment_1          1
Moisture_1             51
Volatiles_1            43
Fixed Carbon_1         49
Ash_1                  12
HHV, MJ/Kg_1           94
C_1                    13
H_1                    13
N_1                    36
O_1                    14
S_1                   111
Cl_1                  172
H/C(0)_1               13
O/C(0)_1               13
Cellulose_1            93
Hemicellulose_1       107
Lignin_1              102
Extractives_1         128
Particle Size, mm_1   143
O/C oil                 0
dtype: int64
```

A lot of missing values!

Preprocessing

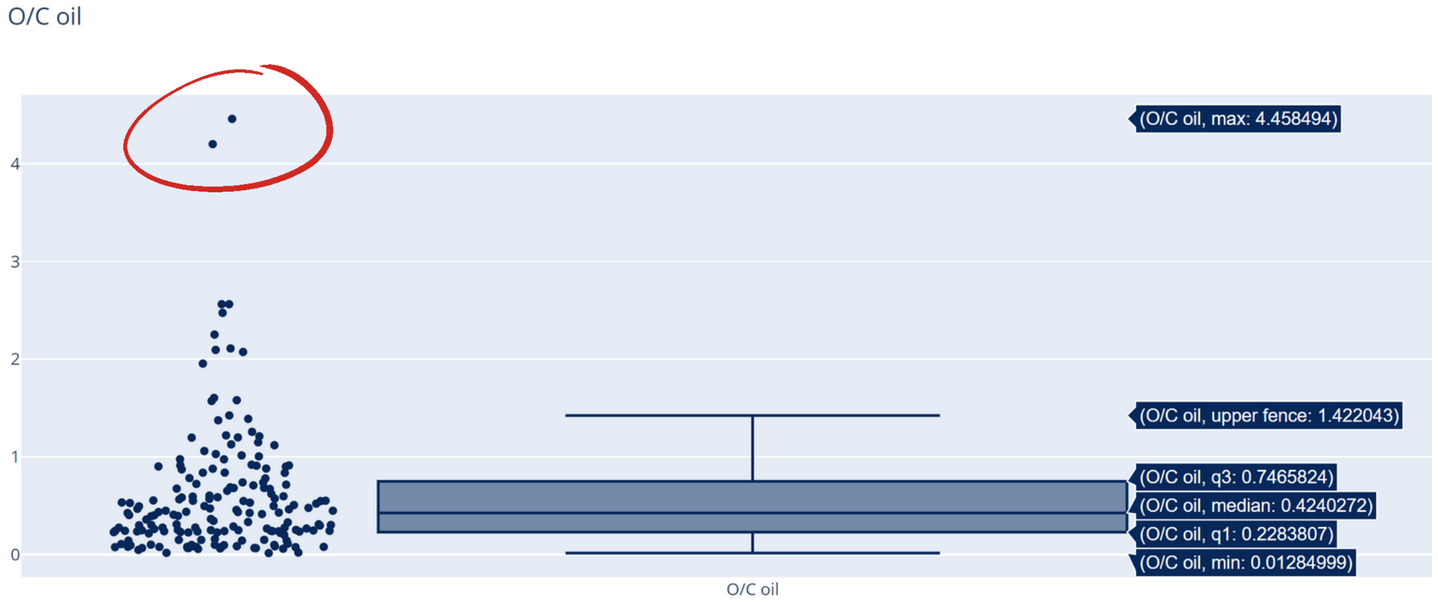
```
#o/c oil range for distribution column
min=0.0128
q1=0.2283
mediana=0.4240
q3=0.7465
upper=1.422043

list_value=[]

for value in df['O/C oil']:
  if value<=q1:
    list_value.append(1)
  elif value>q1 and value<mediana:
    list_value.append(2)
  elif value>=mediana and value<q3:
    list_value.append(3)
  elif value>=q3 and value<upper:
    list_value.append(4)
  else:
    list_value.append(5)
```

```
[9] len(list_value)
```

```
177
```



HHV, MJ/Kg_1	C_1	H_1	N_1	O_1	S_1	Cl_1	H/C(0)_1	O/C(0)_1	Cellulose_1	Hemicellulose_1	Lignin_1	Extractives_1	Particle Size, mm_1	O/C oil
18.61	58.14	6.05	0.69	34.57	0.19	0.36	0.104059	0.594599	NaN	NaN	NaN	NaN	<0.50 mm	4.458494
22.83	63.45	6.73	0.43	28.27	0.17	0.95	0.106068	0.445548	NaN	NaN	NaN	NaN	<0.50 mm	4.200194

Preprocessing

```
[12] new_column=pd.DataFrame(list_value, columns=['Distribution'])

     new_column
```

	Distribution
0	3
1	5
2	2
3	2
4	5
...	...
172	2
173	3
174	1
175	1
176	1

177 rows × 1 columns

```
df = pd.concat([df, new_column], axis=1)
```

Cellulose_1	Hemicellulose_1	Lignin_1	Extractives_1	Particle Size, mm_1	O/C oil	Distribution
NaN	NaN	NaN	NaN	NaN	0.469887	3
41.0	26.5	25.3	NaN	0.6	2.561132	5
NaN	NaN	NaN	NaN	NaN	0.393652	2
NaN	NaN	NaN	NaN	NaN	0.236315	2
NaN	NaN	NaN	NaN	0.6 mm <particle diameter < 1.18 mm	1.952824	5
...	...	...	...	...	...	...
37.8	25.3	23.3	NaN	NaN	0.307387	2
NaN	NaN	NaN	NaN	NaN	0.494919	3
NaN	NaN	NaN	NaN	NaN	0.018292	1
NaN	NaN	NaN	NaN	NaN	0.015102	1
35.0	29.0	28.0	NaN	0	0.083324	1

Preprocessing

```
filtered_df = df[df['Type_1']=='pure']
filtered_df
```

	Material_1	Type_1	Pretreatment_1	O/C(0)_1	Cellulose_1	Hemicellulose_1	Lignin_1	Extractives_1	Particle Size, mm_1	O/C oil	Distribution
52	cellulose	pure	none	1.228991	100.0	NaN	NaN	NaN	NaN	0.669970	3
89	cellulose	pure	none	1.228991	100.0	NaN	NaN	NaN	NaN	0.836571	4
121	cellulose	pure	none	1.123241	100.0	NaN	NaN	NaN	NaN	0.147731	1
122	cellulose	pure	none	1.123241	100.0	NaN	NaN	NaN	NaN	0.147731	1
132	lignin	pure	none	0.603578	NaN	NaN	100.0	NaN	NaN	0.423023	2
162	xylan	pure	none	1.332174	NaN	100.0	NaN	NaN	NaN	0.105183	1

```
filtered_df = df[df['Type_1'].isna()]
filtered_df
```

	Material_1	Type_1	Pretreatment_1
5	mallee	NaN	none
30	cellulose	NaN	none
35	cellulose	NaN	none
43	camelina straw pellets	NaN	none
55	lignin	NaN	none

Preprocessing

	Material_1	Type_1	Pretreatment_1	Moisture_1	Volatiles_1	Fixed Carbon_1	Ash_1	HHV, MJ/Kg_1	C_1	H_1	N_1	O_1	S_1	Cl_1	H/C(0)_1	O/C(0)_1	Cellulose_1	Hemicellulose_1	Lignin_1	Ex
0	japonica alga	algae	acid_washed	5.03	NaN	NaN	7.32	14.27	43.76	6.00	2.76	55.38	NaN	NaN	0.137112	1.265539	NaN	NaN	NaN	
2	japonica alga	algae	none	6.90	NaN	NaN	20.21	16.96	41.22	7.73	1.17	49.88	NaN	NaN	0.187530	1.210092	NaN	NaN	NaN	
3	beech wood	hardwood	none	8.00	NaN	NaN	0.54	NaN	48.44	5.83	NaN	45.73	NaN	NaN	0.120355	0.944055	NaN	NaN	NaN	
6	beech wood	hardwood	none	NaN	NaN	NaN	2.80	18.20	46.30	6.30	0.10	36.10	NaN	NaN	0.136069	0.779698	NaN	NaN	NaN	
7	olive husk	industrial waste	none	3.66	NaN	NaN	1.88	21.57	51.80	6.83	1.32	40.05	NaN	NaN	0.131853	0.773166	NaN	NaN	NaN	
12	bamboo	herbaceous	none	6.56	NaN	NaN	2.13	17.95	45.55	7.02	0.13	47.30	NaN	NaN	0.154116	1.038419	NaN	NaN	NaN	
13	poplar	hardwood	none	6.20	NaN	NaN	3.40	18.80	46.60	6.00	0.60	43.50	NaN	NaN	0.128755	0.933476	49.50	NaN	26.80	
16	pine wood	softwood	none	NaN	NaN	NaN	0.20	NaN	50.40	6.40	0.70	42.50	NaN	NaN	0.126984	0.843254	NaN	NaN	NaN	
21	pine wood	softwood	none	NaN	NaN	NaN	0.20	NaN	50.40	6.40	0.70	42.50	NaN	NaN	0.126984	0.843254	NaN	NaN	NaN	
74	pine wood	softwood	none	12.00	83.90	NaN	0.40	20.40	51.40	5.90	0.10	42.20	NaN	NaN	0.114786	0.821012	NaN	NaN	NaN	
77	olive husk	industrial waste	none	3.66	NaN	NaN	1.88	21.57	51.80	6.83	1.32	40.05	NaN	NaN	0.131853	0.773166	NaN	NaN	NaN	
83	bamboo	herbaceous	none	6.56	NaN	NaN	2.13	17.95	45.55	7.02	0.13	47.30	NaN	NaN	0.154116	1.038419	NaN	NaN	NaN	
84	japonica alga	algae	none	6.90	NaN	NaN	20.21	16.96	41.22	7.73	1.17	49.88	NaN	NaN	0.187530	1.210092	NaN	NaN	NaN	
95	beech wood	hardwood	none	NaN	NaN	NaN	2.80	18.20	46.30	6.30	0.10	36.10	NaN	NaN	0.136069	0.779698	NaN	NaN	NaN	
97	japonica alga	algae	acid_washed	5.03	NaN	NaN	7.32	14.27	43.76	6.00	2.76	55.38	NaN	NaN	0.137112	1.265539	NaN	NaN	NaN	
101	lignin	pure	none	NaN	NaN	NaN	NaN	NaN	48.30	5.10	NaN	46.60	6.40	NaN	0.105590	0.964803	0.00	0.00	100.00	
106	olive kernels	industrial waste	none	3.66	NaN	NaN	1.88	21.57	51.80	6.83	1.32	40.05	NaN	NaN	0.131853	0.773166	NaN	NaN	NaN	
112	poplar	hardwood	none	NaN	NaN	NaN	NaN	NaN	43.10	5.40	NaN	51.50	NaN	NaN	0.125290	1.194896	NaN	NaN	NaN	
120	beech wood	hardwood	none	8.00	NaN	NaN	1.35	NaN	45.98	6.39	NaN	46.28	NaN	NaN	0.138973	1.006525	39.25	33.91	21.75	
126	emal	agricultural_waste	none	NaN	NaN	NaN	2.80	20.35	50.42	6.24	0.35	40.10	0.09	NaN	0.123760	0.795319	NaN	NaN	NaN	
131	pine wood	softwood	none	8.10	76.70	NaN	2.10	NaN	51.40	6.00	0.50	40.00	0.04	NaN	0.116732	0.778210	NaN	NaN	NaN	
148	pine wood	softwood	none	8.00	NaN	NaN	NaN	NaN	46.58	6.34	0.04	46.98	0.06	NaN	0.136110	1.008587	35.00	29.00	28.00	
149	pine wood	softwood	none	NaN	NaN	NaN	0.37	NaN	47.40	6.10	0.10	46.40	NaN	NaN	0.128692	0.978903	NaN	NaN	NaN	
165	coconut shell	agricultural_waste	none	11.26	85.36	NaN	3.38	22.83	63.45	6.73	0.43	28.27	0.17	0.95	0.106068	0.445548	NaN	NaN	NaN	
168	rice straw	agricultural_waste	none	13.61	76.85	NaN	9.54	16.35	50.93	6.04	0.83	41.61	0.23	0.36	0.118594	0.817004	NaN	NaN	NaN	
169	rice straw	agricultural_waste	none	6.48	82.30	NaN	11.22	NaN	39.20	4.84	1.60	53.69	0.67	NaN	0.123469	1.369643	54.67	32.53	12.83	
171	rice straw	agricultural_waste	none	NaN	NaN	NaN	13.50	15.69	40.24	5.44	3.40	37.22	0.17	NaN	0.135189	0.924950	34.40	19.60	22.90	
173	sugarcane bagasse	agricultural_waste	none	16.07	79.59	NaN	4.34	18.61	58.14	6.05	0.69	34.57	0.19	0.36	0.104059	0.594599	NaN	NaN	NaN	
176	pine wood	softwood	step-wise pyrolysis (torrefaction)	8.00	NaN	NaN	NaN	NaN	46.58	6.34	0.04	46.98	0.06	NaN	0.136110	1.008587	35.00	29.00	28.00	

The sum is about 100

```
#Duplicate raw
df=df.drop(121)
```

```
for i in df.index:
  if df['Material_1'][i]=='cellulose':
    df['Type_1'][i]='pure'
    df['Cellulose_1'][i]=100.0
    df['Hemicellulose_1'][i]=0.0
    df['Lignin_1'][i]=0.0
    df['Extractives_1'][i]=0.0

  elif df['Material_1'][i]=='lignin':
    df['Type_1'][i]='pure'
    df['Cellulose_1'][i]=0.0
    df['Hemicellulose_1'][i]=0.0
    df['Lignin_1'][i]=100.0
    df['Extractives_1'][i]=0.0

  elif df['Material_1'][i]=='xylan':
    df['Type_1'][i]='pure'
    df['Cellulose_1'][i]=0.0
    df['Hemicellulose_1'][i]=100.0
    df['Lignin_1'][i]=0.0
    df['Extractives_1'][i]=0.0
```

How to fix

- *Delete duplicate rows*
- *Change Type value to Pure*
- *Change other material values to 0.0*
- *Make the difference between 100 and the other values where it is possible otherwise put 0 (slide ->)*

How to fix

```
for i in df.index:
  if np.isnan(df['Moisture_1'][i]) and df['Volatiles_1'][i]!='nan' and df['Fixed Carbon_1'][i]!='nan' and df['Ash_1'][i]!='nan':
    df['Moisture_1'][i]=100-(df['Volatiles_1'][i]+df['Fixed Carbon_1'][i]+df['Ash_1'][i])

  elif np.isnan(df['Volatiles_1'][i]) and  df['Fixed Carbon_1'][i]!='nan' and df['Ash_1'][i]!='nan' and df['Moisture_1'][i]!='nan':
    df['Volatiles_1'][i]=100-(df['Moisture_1'][i]+df['Fixed Carbon_1'][i]+df['Ash_1'][i])

  elif np.isnan(df['Fixed Carbon_1'][i]) and  df['Volatiles_1'][i]!='nan' and df['Ash_1'][i]!='nan'and df['Moisture_1'][i]!='nan':
    df['Fixed Carbon_1'][i]=100-(df['Moisture_1'][i]+df['Volatiles_1'][i]+df['Ash_1'][i])

  elif np.isnan(df['Ash_1'][i]) and  df['Fixed Carbon_1'][i]!='nan' and df['Volatiles_1'][i]!='nan'and df['Moisture_1'][i]!='nan':
    df['Ash_1'][i]=100-(df['Moisture_1'][i]+df['Fixed Carbon_1'][i]+df['Volatiles_1'][i])
```

```
for i in df.index:
  if np.isnan(df['C_1'][i]) and df['H_1'][i]!='nan' and df['N_1'][i]!='nan' and df['S_1'][i]!='nan' and df['O_1'][i]!='nan' and df['Cl_1'][i]!='nan':
    df['C_1'][i]=100-(df['H_1'][i]+df['N_1'][i]+df['O_1'][i]+df['S_1'][i]+df['Cl_1'][i])

  elif np.isnan(df['H_1'][i]) and df['C_1'][i]!='nan' and df['N_1'][i]!='nan' and df['S_1'][i]!='nan' and df['O_1'][i]!='nan' and df['Cl_1'][i]!='nan':
    df['H_1'][i]=100-(df['C_1'][i]+df['N_1'][i]+df['O_1'][i]+df['S_1'][i]+df['Cl_1'][i])

  elif np.isnan(df['N_1'][i]) and df['C_1'][i]!='nan' and df['H_1'][i]!='nan' and df['S_1'][i]!='nan' and df['O_1'][i]!='nan' and df['Cl_1'][i]!='nan':
    df['N_1'][i]=100-(df['C_1'][i]+df['H_1'][i]+df['O_1'][i]+df['S_1'][i]+df['Cl_1'][i])

  elif np.isnan(df['O_1'][i]) and df['C_1'][i]!='nan' and df['N_1'][i]!='nan' and df['S_1'][i]!='nan' and df['H_1'][i]!='nan' and df['Cl_1'][i]!='nan':
    df['O_1'][i]=100-(df['C_1'][i]+df['N_1'][i]+df['H_1'][i]+df['S_1'][i]+df['Cl_1'][i])

  elif np.isnan(df['S_1'][i]) and df['C_1'][i]!='nan' and df['N_1'][i]!='nan' and df['H_1'][i]!='nan' and df['O_1'][i]!='nan' and df['Cl_1'][i]!='nan':
    df['S_1'][i]=100-(df['C_1'][i]+df['N_1'][i]+df['O_1'][i]+df['H_1'][i]+df['Cl_1'][i])

  elif np.isnan(df['Cl_1'][i]) and df['C_1'][i]!='nan' and df['N_1'][i]!='nan' and df['S_1'][i]!='nan' and df['O_1'][i]!='nan' and df['H_1'][i]!='nan':
    df['Cl_1'][i]=100-(df['C_1'][i]+df['N_1'][i]+df['O_1'][i]+df['S_1'][i]+df['H_1'][i])
```

How to fix

```
for i in df.index:
  if np.isnan(df['Cellulose_1'][i]) and df['Hemicellulose_1'][i]!='nan' and df['Lignin_1'][i]!='nan' and df['Extractives_1'][i]!='nan':
    df['Cellulose_1'][i]=100-(df['Hemicellulose_1'][i]+df['Lignin_1'][i]+df['Extractives_1'][i])

  elif np.isnan(df['Hemicellulose_1'][i]) and df['Lignin_1'][i]!='nan' and df['Extractives_1'][i]!='nan' and df['Cellulose_1'][i]!='nan':
    df['Hemicellulose_1'][i]=100-(df['Lignin_1'][i]+df['Extractives_1'][i]+df['Cellulose_1'][i])

  elif np.isnan(df['Lignin_1'][i]) and df['Extractives_1'][i]!='nan' and df['Cellulose_1'][i]!='nan' and df['Hemicellulose_1'][i]!='nan':
    df['Lignin_1'][i]=100-(df['Extractives_1'][i]+df['Cellulose_1'][i]+df['Hemicellulose_1'][i])

  elif np.isnan(df['Extractives_1'][i]) and df['Cellulose_1'][i]!='nan' and df['Hemicellulose_1'][i]!='nan' and df['Lignin_1'][i]!='nan':
    df['Extractives_1'][i]=100-(df['Cellulose_1'][i]+df['Hemicellulose_1'][i]+df['Lignin_1'][i])
```

```
colonne=["C_1", "H_1",  "N_1",  "O_1",  "S_1",  "Cl_1", "Moisture_1", "Volatiles_1",  "Fixed Carbon_1", "Ash_1",
         "Cellulose_1", "Hemicellulose_1",  "Lignin_1", "Extractives_1"]

for column in colonne:
  df[column] = df[column].fillna(0)
```

Results

```
#Count the number of empty values of each columns in dataset
df.isna().sum()
```

```
Material_1              0
Type_1                  8
Pretreatment_1          1
Moisture_1             51
Volatiles_1            43
Fixed Carbon_1         49
Ash_1                  12
HHV, MJ/Kg_1           94
C_1                    13
H_1                    13
N_1                    36
O_1                    14
S_1                   111
Cl_1                  172
H/C(0)_1               13
O/C(0)_1               13
Cellulose_1            93
Hemicellulose_1       107
Lignin_1              102
Extractives_1         128
Particle Size, mm_1   143
O/C oil                 0
dtype: int64
```

Before

```
#Count the number of empty values of each columns in dataset
df.isna().sum()
```

```
Material_1              0
Type_1                  4
Pretreatment_1          0
Moisture_1              0
Volatiles_1             0
Fixed Carbon_1          0
Ash_1                   0
HHV, MJ/Kg_1           92
C_1                     0
H_1                     0
N_1                     0
O_1                     0
S_1                     0
Cl_1                    0
H/C(0)_1               12
O/C(0)_1               12
Cellulose_1             0
Hemicellulose_1         0
Lignin_1                0
Extractives_1           0
Particle Size, mm_1   141
O/C oil                 0
Distribution            0
dtype: int64
```

After

Results

```
#Count the number of empty values of each columns in dataset
df.isna().sum()
```

```
Material_1              0
Type_1                  8
Pretreatment_1          1
Moisture_1             51
Volatiles_1            43
Fixed Carbon_1         49
Ash_1                  12
HHV, MJ/Kg_1           94
C_1                    13
H_1                    13
N_1                    36
O_1                    14
S_1                   111
Cl_1                  172
H/C(0)_1               13
O/C(0)_1               13
Cellulose_1            93
Hemicellulose_1       107
Lignin_1              102
Extractives_1         128
Particle Size, mm_1   143
O/C oil                 0
dtype: int64
```

I deleted these two columns

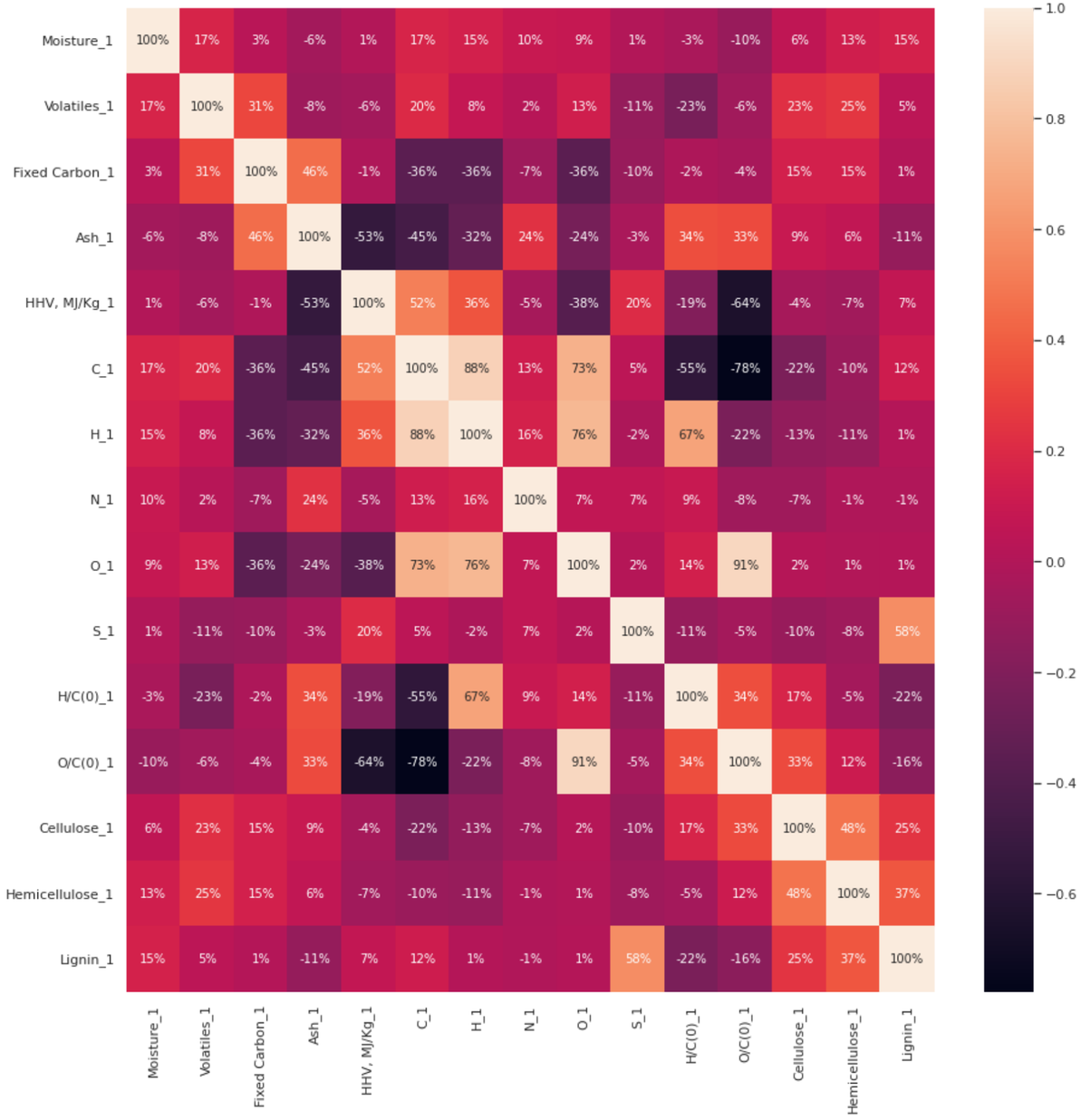
Before

```
#Count the number of empty values of each columns in dataset
df.isna().sum()
```

```
Material_1              0
Type_1                  4
Pretreatment_1          0
Moisture_1              0
Volatiles_1             0
Fixed Carbon_1          0
Ash_1                   0
HHV, MJ/Kg_1           92
C_1                     0
H_1                     0
N_1                     0
O_1                     0
S_1                     0
Cl_1                    0
H/C(0)_1               12
O/C(0)_1               12
Cellulose_1             0
Hemicellulose_1         0
Lignin_1                0
Extractives_1           0
Particle Size, mm_1   141
O/C oil                 0
Distribution            0
dtype: int64
```

After

Feature Selection



I deleted the columns that had more than 80% correlation

```
# Dropping of features with a correlation greater than 0.8
# Code readapted from Chris Albon
# Create the correlation matrix with absolute values
corr_matrix = df_features.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.8
to_drop = [column for column in upper.columns if any(upper[column] > 0.80)]
```

```
to_drop
```

```
['H_1', 'O/C(0)_1']
```

```
df=df.drop(columns=to_drop)
```

Encoding

```
categorical_cols=df.columns[df.dtypes==object].tolist()
categorical_cols
```

```
['Material_1', 'Type_1', 'Pretreatment_1', 'Particle Size, mm_1']
```

```
df["Material_1_cat"]=df["Material_1"].astype('category').cat.codes
df["Type_1_cat"]=df["Type_1"].astype('category').cat.codes
df["Pretreatment_1_cat"]=df["Pretreatment_1"].astype('category').cat.codes
df["Particle Size, mm_1_cat"]=df["Particle Size, mm_1"].astype('category').cat.codes
df.head()
```

HHV, MJ/Kg_1	C_1	N_1	O_1	S_1	H/C(0)_1	Cellulose_1	Hemicellulose_1	Lignin_1	Particle Size, mm_1	O/C oil	Distribution	Material_1_cat	Type_1_cat	Pretreatment_1_cat	Particle Size, mm_1_cat
14.27	43.76	2.76	55.38	0.00	0.137112	0.0	0.0	0.0	NaN	0.469887	3	22	1	0	-1
17.70	46.90	0.21	47.02	0.02	0.124733	41.0	26.5	25.3	0.6	2.561132	5	5	0	2	11
16.96	41.22	1.17	49.88	0.00	0.187530	0.0	0.0	0.0	NaN	0.393652	2	22	1	2	-1
NaN	48.44	0.00	45.73	0.00	0.120355	0.0	0.0	0.0	NaN	0.236315	2	6	5	2	-1
									0.6 mm						

Disclaimer

I know that I have defined a numerical ordering between the categorical rows that was not there before, but doing various tests by deleting these columns, the performances weren't so good.

Splitting Data

```
X=df.drop(columns=["O/C oil", 'Material_1', 'Type_1', 'Pretreatment_1','Particle Size, mm_1',"Distribution"])
Y=df["O/C oil"]
```

```
X.columns
```

```
Index(['Moisture_1', 'Volatiles_1', 'Fixed Carbon_1', 'Ash_1', 'HHV, MJ/Kg_1',
       'C_1', 'N_1', 'O_1', 'S_1', 'H/C(0)_1', 'Cellulose_1',
       'Hemicellulose_1', 'Lignin_1', 'Material_1_cat', 'Type_1_cat',
       'Pretreatment_1_cat', 'Particle Size, mm_1_cat'],
      dtype='object')
```

```
#Split the training e validation 70% and 30% testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, stratify=df["Distribution"], random_state=0)
```

```
#Split the training e validation 75% and 25% testing
from sklearn.model_selection import train_test_split

X_train, X_validation, Y_train, Y_validation = train_test_split(X_train, Y_train, test_size=0.25, random_state=0)
```

Last Missing values

```
import statistics
mediane={}
mediane_tot={}

for column in X_train.columns:
  mediana=X_train.groupby(column)[column].transform("median")
  X_train[column].fillna(mediana, inplace=True)
  mediane.update({column: mediana})
  print(str(column) + ": " + str(statistics.median(X_train[column])))
```

```
Moisture_1: 5.18
Volatiles_1: 74.76
Fixed Carbon_1: 14.27
Ash_1: 2.45
HHV, MJ/Kg_1: nan
C_1: 46.9
N_1: 0.3
O_1: 45.2
S_1: 0.0
H/C(0)_1: nan
Cellulose_1: 20.35
Hemicellulose_1: 0.0
Lignin_1: 0.0
Material_1_cat: 27
Type_1_cat: 5
Pretreatment_1_cat: 2
Particle Size, mm_1_cat: -1
```

I calculated the values of the medians in the train dataset and then replaced it in the validation and in the test →

Remaining Nan values were replaced the with -1 →

```
for column in X_train.columns[3:-1]:

  X_train[column]=X_train[column].fillna(mediane[column])
  X_validation[column]=X_validation[column].fillna(mediane[column])
  X_test[column]=X_test[column].fillna(mediane[column])

X_train=X_train.fillna(-1)
X_validation=X_validation.fillna(-1)
X_test=X_test.fillna(-1)
```

Normalization

```
#Scale the data (Feature Scaling Z Score) z = (x - u) / s
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_validation=sc.transform(X_validation)
X_test = sc.transform(X_test)
```

```
X_train = np.array(X_train)
X_validation=np.array(X_validation)
X_test = np.array(X_test)

Y_train = np.array(Y_train)
Y_validation=np.array(Y_validation)
Y_test = np.array(Y_test)
```

Algorithm Choice

Regressors tested

```
regressors = [
    KNeighborsRegressor(),
    GradientBoostingRegressor(),
    ExtraTreesRegressor(),
    RandomForestRegressor(),
    DecisionTreeRegressor(),
    LinearRegression(),
    Lasso(),
    Ridge(),
    SVR(),
    SVR(kernel="poly"),
    SVR(kernel="rbf")
]
```

Best three

```
GradientBoostingRegressor()
        Training time: 0.044s
        Prediction time: 0.000s
        Explained variance: 0.7913680428484982
        Mean absolute error: 0.26884599489044514
        R2 score: 0.773793003873987

ExtraTreesRegressor()
        Training time: 0.100s
        Prediction time: 0.008s
        Explained variance: 0.7714371633188151
        Mean absolute error: 0.29764231114507095
        R2 score: 0.7634724234233365

RandomForestRegressor()
        Training time: 0.146s
        Prediction time: 0.008s
        Explained variance: 0.7371090995718285
        Mean absolute error: 0.3302963751432704
        R2 score: 0.7334674808400609
```

Algorithm Choice

Gradient Boosting Regressor

Gradient boosting builds an additive mode by using multiple decision trees of fixed size as weak learners or weak predictive models. The parameter, n_estimators, decides the number of decision trees which will be used in the boosting stages.

ExtraTrees Regressor

The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

Random Forest Regressor

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time

Tuning

GradientBoostingRegressor

```
parameters = { 'loss' : ['squared_error', 'absolute_error', 'huber', 'quantile'],
               'learning_rate' : (0.05,0.25,0.50,1),
               'criterion' : ['squared_error', 'absolute_error'],
               'max_features' : ['auto', 'sqrt', 'log2']
              }
```

```
grid = GridSearchCV(GradientBoostingRegressor(), parameters, scoring='neg_mean_absolute_error', cv=5)
model = grid.fit(X_train_all,Y_train_all)
```

```
print(model.best_params_,'\n')
print(model.best_estimator_,'\n')
```

```
{'criterion': 'absolute_error', 'learning_rate': 0.25, 'loss': 'squared_error', 'max_features': 'log2'}

GradientBoostingRegressor(criterion='absolute_error', learning_rate=0.25,
                          max_features='log2')
```

Tuning

ExtraTreesRegressor

```
parameters = {'max_depth':[9,10,11,12,13,14], 'criterion':['squared_error','absolute_error'], 'n_estimators': [100,200,300,1000]}
```

```
grid = GridSearchCV(ExtraTreesRegressor(), parameters, scoring='neg_mean_absolute_error', cv=5)
model = grid.fit(X_train_all,Y_train_all)
```

```
print(model.best_params_,'\n')
print(model.best_estimator_,'\n')
```

```
{'criterion': 'absolute_error', 'max_depth': 11, 'n_estimators': 100}

ExtraTreesRegressor(criterion='absolute_error', max_depth=11)
```

Tuning

RandomForestRegressor

```
parameters = { 'max_depth' : [100,200,300,500,1000], 'min_samples_split': [3,4,5]}
```

```
grid = GridSearchCV(RandomForestRegressor(), parameters, scoring='neg_mean_absolute_error', cv=5)
model = grid.fit(X_train_all,Y_train_all)
```

```
print(model.best_params_,'\n')
print(model.best_estimator_,'\n')
```

```
{'max_depth': 100, 'min_samples_split': 3}

RandomForestRegressor(max_depth=100, min_samples_split=3)
```

Local Comparison

Gradient Boosting Regressor

Mean Absolute Error

Validation: 0.287

Test: 0.308

ExtraTrees Regressor

Mean Absolute Error

Validation: 0.267

Test: 0.286

Random Forest Regressor

Mean Absolute Error

Validation: 0.337

Test: 0.337

Merge Train+Validation

```
df_1=pd.DataFrame(X_train)
df_2=pd.DataFrame(X_validation)

X_n=df_1.append(df_2, ignore_index=True)

df_1=pd.DataFrame(Y_train)
df_2=pd.DataFrame(Y_validation)

Y_n=df_1.append(df_2, ignore_index=True)

#Scale the data (Feature Scaling Z Score) z = (x - u) / s
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_n = sc.fit_transform(X_n)

X_n=np.array(X_n)
Y_n=np.array(Y_n)

X_n_test=X_test
Y_n_test=Y_test
```

Train + Validation - Local Comparison

Gradient Boosting Regressor

Mean Absolute Error

Test: 0.339

ExtraTrees Regressor

Mean Absolute Error

Test: 0.294

Random Forest Regressor

Mean Absolute Error

Test: 0.363

File Test Preprocessing

```
for i in df_test.index:
  if df_test['Material_1'][i]=='cellulose':
    df_test['Type_1'][i]='pure'
    df_test['Cellulose_1'][i]=100.0
    df_test['Hemicellulose_1'][i]=0.0
    df_test['Lignin_1'][i]=0.0
    df_test['Extractives_1'][i]=0.0

  elif df_test['Material_1'][i]=='lignin':
    df_test['Type_1'][i]='pure'
    df_test['Cellulose_1'][i]=0.0
    df_test['Hemicellulose_1'][i]=0.0
    df_test['Lignin_1'][i]=100.0
    df_test['Extractives_1'][i]=0.0

  elif df_test['Material_1'][i]=='xylan':
    df_test['Type_1'][i]='pure'
    df_test['Cellulose_1'][i]=0.0
    df_test['Hemicellulose_1'][i]=100.0
    df_test['Lignin_1'][i]=0.0
    df_test['Extractives_1'][i]=0.0
```

All preprocessing operations done on the training set were carried over to the test set

File Test Preprocessing

```
#Moisture_1 Volatiles_1 Fixed Carbon_1  Ash_1
for i in df_test.index:
  if np.isnan(df_test['Moisture_1'][i]) and df_test['Volatiles_1'][i]!='nan' and df_test['Fixed Carbon_1'][i]!='nan' and df_test['Ash_1'][i]!='nan':
    df_test['Moisture_1'][i]=100-(df_test['Volatiles_1'][i]+df_test['Fixed Carbon_1'][i]+df_test['Ash_1'][i])

  elif np.isnan(df_test['Volatiles_1'][i]) and  df_test['Fixed Carbon_1'][i]!='nan' and df_test['Ash_1'][i]!='nan' and df_test['Moisture_1'][i]!='nan':
    df_test['Volatiles_1'][i]=100-(df_test['Moisture_1'][i]+df_test['Fixed Carbon_1'][i]+df_test['Ash_1'][i])

  elif np.isnan(df_test['Fixed Carbon_1'][i]) and  df_test['Volatiles_1'][i]!='nan' and df_test['Ash_1'][i]!='nan'and df_test['Moisture_1'][i]!='nan':
    df_test['Fixed Carbon_1'][i]=100-(df_test['Moisture_1'][i]+df['Volatiles_1'][i]+df_test['Ash_1'][i])

  elif np.isnan(df_test['Ash_1'][i]) and  df_test['Fixed Carbon_1'][i]!='nan' and df_test['Volatiles_1'][i]!='nan'and df_test['Moisture_1'][i]!='nan':
    df_test['Ash_1'][i]=100-(df_test['Moisture_1'][i]+df_test['Fixed Carbon_1'][i]+df_test['Volatiles_1'][i])
```

```
#C_1  H_1 N_1 O_1 S_1 Cl_1
for i in df_test.index:
  if np.isnan(df_test['C_1'][i]) and df_test['H_1'][i]!='nan' and df_test['N_1'][i]!='nan' and df_test['S_1'][i]!='nan' and df_test['O_1'][i]!='nan' and df_test['Cl_1'][i]!='nan':
    df_test['C_1'][i]=100-(df_test['H_1'][i]+df_test['N_1'][i]+df_test['O_1'][i]+df_test['S_1'][i]+df_test['Cl_1'][i])

  elif np.isnan(df_test['H_1'][i]) and df_test['C_1'][i]!='nan' and df_test['N_1'][i]!='nan' and df_test['S_1'][i]!='nan' and df_test['O_1'][i]!='nan' and df_test['Cl_1'][i]!='nan':
    df_test['H_1'][i]=100-(df_test['C_1'][i]+df_test['N_1'][i]+df_test['O_1'][i]+df_test['S_1'][i]+df_test['Cl_1'][i])

  elif np.isnan(df_test['N_1'][i]) and df_test['C_1'][i]!='nan' and df_test['H_1'][i]!='nan' and df_test['S_1'][i]!='nan' and df_test['O_1'][i]!='nan' and df_test['Cl_1'][i]!='nan':
    df_test['N_1'][i]=100-(df_test['C_1'][i]+df_test['H_1'][i]+df_test['O_1'][i]+df_test['S_1'][i]+df_test['Cl_1'][i])

  elif np.isnan(df_test['O_1'][i]) and df_test['C_1'][i]!='nan' and df_test['N_1'][i]!='nan' and df_test['S_1'][i]!='nan' and df_test['H_1'][i]!='nan' and df_test['Cl_1'][i]!='nan':
    df_test['O_1'][i]=100-(df_test['C_1'][i]+df_test['N_1'][i]+df_test['H_1'][i]+df_test['S_1'][i]+df_test['Cl_1'][i])

  elif np.isnan(df_test['S_1'][i]) and df_test['C_1'][i]!='nan' and df_test['N_1'][i]!='nan' and df_test['H_1'][i]!='nan' and df_test['O_1'][i]!='nan' and df_test['Cl_1'][i]!='nan':
    df_test['S_1'][i]=100-(df_test['C_1'][i]+df_test['N_1'][i]+df_test['O_1'][i]+df_test['H_1'][i]+df_test['Cl_1'][i])

  elif np.isnan(df_test['Cl_1'][i]) and df_test['C_1'][i]!='nan' and df_test['N_1'][i]!='nan' and df_test['S_1'][i]!='nan' and df_test['O_1'][i]!='nan' and df_test['H_1'][i]!='nan':
    df_test['Cl_1'][i]=100-(df_test['C_1'][i]+df_test['N_1'][i]+df_test['O_1'][i]+df_test['S_1'][i]+df_test['H_1'][i])
```

File Test Preprocessing

```
#Cellulose_1  Hemicellulose_1 Lignin_1  Extractives_1
for i in df_test.index:
  if np.isnan(df_test['Cellulose_1'][i]) and df_test['Hemicellulose_1'][i]!='nan' and df_test['Lignin_1'][i]!='nan' and df_test['Extractives_1'][i]!='nan':
    df_test['Cellulose_1'][i]=100-(df_test['Hemicellulose_1'][i]+df_test['Lignin_1'][i]+df_test['Extractives_1'][i])

  elif np.isnan(df_test['Hemicellulose_1'][i]) and df_test['Lignin_1'][i]!='nan' and df_test['Extractives_1'][i]!='nan' and df_test['Cellulose_1'][i]!='nan':
    df_test['Hemicellulose_1'][i]=100-(df_test['Lignin_1'][i]+df_test['Extractives_1'][i]+df_test['Cellulose_1'][i])

  elif np.isnan(df_test['Lignin_1'][i]) and df_test['Extractives_1'][i]!='nan' and df_test['Cellulose_1'][i]!='nan' and df_test['Hemicellulose_1'][i]!='nan':
    df_test['Lignin_1'][i]=100-(df_test['Extractives_1'][i]+df_test['Cellulose_1'][i]+df_test['Hemicellulose_1'][i])

  elif np.isnan(df_test['Extractives_1'][i]) and df_test['Cellulose_1'][i]!='nan' and df_test['Hemicellulose_1'][i]!='nan' and df_test['Lignin_1'][i]!='nan':
    df_test['Extractives_1'][i]=100-(df_test['Cellulose_1'][i]+df_test['Hemicellulose_1'][i]+df_test['Lignin_1'][i])
```

```
#C_1  H_1 N_1 O_1 S_1 Cl_1
#Moisture_1 Volatiles_1 Fixed Carbon_1  Ash_1
#Cellulose_1  Hemicellulose_1 Lignin_1  Extractives_1

colonne=["C_1", "H_1",  "N_1",  "O_1",  "S_1",  "Cl_1", "Moisture_1", "Volatiles_1",  "Fixed Carbon_1", "Ash_1",
         "Cellulose_1", "Hemicellulose_1",  "Lignin_1", "Extractives_1"]

for column in colonne:
  df_test[column] = df_test[column].fillna(0)
```

File Test Preprocessing

```
df_test["Material_1_cat"]=df_test["Material_1"].astype('category').cat.codes
df_test["Type_1_cat"]=df_test["Type_1"].astype('category').cat.codes
df_test["Pretreatment_1_cat"]=df_test["Pretreatment_1"].astype('category').cat.codes
df_test["Particle Size, mm_1_cat"]=df_test["Particle Size, mm_1"].astype('category').cat.codes
```

```
df_test=df_test.drop(columns=categorical_cols)
```

```
df_test=df_test.drop(columns=["Cl_1","Extractives_1"])
df_test=df_test.drop(columns=to_drop)
```

```
for column in df_test.columns[:-5]:

  df_test[column]=df_test[column].fillna(mediane[column])

df_test=df_test.fillna(-1)
```

Testing

```
#Dataset train
X=df.drop(columns=["O/C oil", 'Material_1', 'Type_1', 'Pretreatment_1', 'Particle Size, mm_1'])
Y=df["O/C oil"]
```

```
for column in X.columns[3:-1]:

  X[column]=X[column].fillna(mediane[column])

X=X.fillna(-1)
```

```
#Scale the data (Feature Scaling Z Score) z = (x - u) / s
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X = sc.fit_transform(X)
X_test = sc.transform(df_test)
```

The dataset for training the final model consists of all instances of the training file

Kaggle Results Comparison

Gradient Boosting Regressor

Mean Absolute Error

Public: *0.246*

Private: *0.357*

ExtraTrees Regressor

Mean Absolute Error

Public: *0.180*

Private: *0.349*

Random Forest Regressor

Mean Absolute Error

Public: *0.190*

Private: *0.358*

Conclusion

Algorithm and tuning found

Three machine learning algorithms and their tuning have been found to perform well with the available data

Not necessarily the best

For time and computational reasons, not many combinations of hyperparameters were tested during tuning.

Suggestions for the future

- Try to increase the number of combinations by inserting additional hyper-parameters
- Try to modify the feature selection in order to assess whether the choices made are actually the best ones

ANNA LAMBOGLIA

Thanks for your attention!

Tell me your questions.

Code

Identification number

M63001219

E-mail

ann.lamboglia@studenti.unina.it

Linkedin

https://www.linkedin.com/in/annalamboglia/