Predicting Ultimate Tensile Strength (UTS) on austenitic stainless steel

1. Problem Statement:

- Predicting mechanical properties of metals from big data is of great importance to materials engineering.
- The present work aims at applying supervised machine learning model to predict the tensile properties such as ultimate tensile strength (UTS) on austenitic stainless steel as a function of chemical composition, heat treatment and test temperature.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns",None)
```

In [2]:
```
data=pd.read_csv(r"D:\Github\Ultimate Tensile strength prediction\data\Data.csv",index_col="Unnamed: 0")
```

In [3]:
```
data.head()
```

Out[3]:

	Cr	Ni	Mo	Mn	Si	Nb	Ti	Zr	Ta	V	W	Cu	N	C	B	P	S	Co	Al	Sn	Pb	Solution_treatment_te
0	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
1	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
2	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
3	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
4	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	

2. Dataset Description:

2.1. Features:

Column 1 Chromium wt%

Column 2 Nickel wt%

Column 3 Molybdenum wt%

Column 4 Manganese wt%

Column 5 Silicon wt%

Column 6 Niobium wt%

Column 7 Titanium wt%

Column 8 Zirconium wt%

Column 9 Tantalum wt%

Column 10 Vanadium wt%

Column 11 Tungsten wt%

Column 12 Copper wt%

Column 13 Nitrogen wt%

Column 14 Carbon wt%

Column 15 Boron wt%

Column 16 Phosphorus wt%

Column 17 Sulphur wt%

Column 18 Cobalt wt%

Column 19 Aluminium wt%

Column 20 Tin wt%

Column 21 Lead wt%

Column 22 Solution treatment temperature / K

Column 23 Solution treatment time /s

Column 24 Water quenched after solution treatment

Column 25 Air quenched after solution treatment

Column 26 Grains mm-2

Column 27 Type of melting

Column 28 Size of ingot

Column 29 Product form

Column 30 Temperature / K

Column 31 0.2% proof stress / MPa

Column 32 UTS / MPa

Column 33 Elongation (%)

Column 34 Area_reduction (%)

Column 35 Comments

2.2. Information Given:

- Dataset contains the chemical composition of the `2180` steels studied, and their mechanical properties at different heat treatment and test temperatures.
- The presence of an `na` indicates that the value was not reported in the dataset.

In [4]:
```
df=data.copy()
```

- Removing Unwanted columns which cannot be used for model building

In [5]:
```
df.drop(columns=["0.2%proof_stress (M Pa)","Elongation (%)","Area_reduction (%)","Comments"],inplace=True)
```

- As presence of an na indicates that the value was not reported in the dataset, so first we need to convert it into null values

In [6]:
```
df=df.replace("Na", np.NaN)
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2180 entries, 0 to 2179
Data columns (total 31 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Cr                              2180 non-null   float64
 1   Ni                              2180 non-null   float64
 2   Mo                              2180 non-null   float64
 3   Mn                              2180 non-null   float64
 4   Si                              2180 non-null   float64
 5   Nb                              2180 non-null   float64
 6   Ti                              2180 non-null   float64
 7   Zr                              2180 non-null   int64  
 8   Ta                              2180 non-null   int64  
 9   V                               2180 non-null   float64
 10  W                               2179 non-null   float64
 11  Cu                              2180 non-null   float64
 12  N                               2180 non-null   float64
 13  C                               2180 non-null   float64
 14  B                               2180 non-null   float64
 15  P                               2180 non-null   float64
 16  S                               2180 non-null   float64
 17  Co                              2180 non-null   float64
 18  Al                              2180 non-null   float64
 19  Sn                              2180 non-null   int64  
 20  Pb                              2180 non-null   int64  
 21  Solution_treatment_temperature  1996 non-null   object 
 22  Solution_treatment_time(s)      1083 non-null   object 
 23  Water_Quenched_after_s.t.       1916 non-null   object 
 24  Air_Quenched_after_s.t.         1916 non-null   object 
 25  Grains mm-2                     663 non-null    object 
 26  Type of melting                 1963 non-null   object 
 27  Size of ingot                   663 non-null    object 
 28  Product form                    2180 non-null   object 
 29  Temperature (K)                 2180 non-null   int64  
 30  UTS (M Pa)                      2180 non-null   float64
dtypes: float64(18), int64(5), object(8)
memory usage: 545.0+ KB
```

- Converting Datatypes of columns `Solution_treatment_temperature` , `Solution_treatment_time(s)` , `Grains mm-2` , `Size of ingot` to float which is by default object

In [8]:
```
df["Solution_treatment_temperature"]=df["Solution_treatment_temperature"].astype(float)
df["Solution_treatment_time(s)"]=df["Solution_treatment_time(s)"].astype(float)
df["Grains mm-2"]=df["Grains mm-2"].astype(float)
df["Size of ingot"]=df["Size of ingot"].astype(float)
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2180 entries, 0 to 2179
Data columns (total 31 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Cr                              2180 non-null   float64
 1   Ni                              2180 non-null   float64
 2   Mo                              2180 non-null   float64
 3   Mn                              2180 non-null   float64
 4   Si                              2180 non-null   float64
 5   Nb                              2180 non-null   float64
 6   Ti                              2180 non-null   float64
 7   Zr                              2180 non-null   int64  
 8   Ta                              2180 non-null   int64  
 9   V                               2180 non-null   float64
 10  W                               2179 non-null   float64
 11  Cu                              2180 non-null   float64
 12  N                               2180 non-null   float64
 13  C                               2180 non-null   float64
 14  B                               2180 non-null   float64
 15  P                               2180 non-null   float64
 16  S                               2180 non-null   float64
 17  Co                              2180 non-null   float64
 18  Al                              2180 non-null   float64
 19  Sn                              2180 non-null   int64  
 20  Pb                              2180 non-null   int64  
 21  Solution_treatment_temperature  1996 non-null   float64
 22  Solution_treatment_time(s)      1083 non-null   float64
 23  Water_Quenched_after_s.t.       1916 non-null   object 
 24  Air_Quenched_after_s.t.         1916 non-null   object 
 25  Grains mm-2                     663 non-null    float64
 26  Type of melting                 1963 non-null   object 
 27  Size of ingot                   663 non-null    float64
 28  Product form                    2180 non-null   object 
 29  Temperature (K)                 2180 non-null   int64  
 30  UTS (M Pa)                      2180 non-null   float64
dtypes: float64(22), int64(5), object(4)
memory usage: 545.0+ KB
```

3. Data Cleaning

In [10]:
```
df.size
```

Out[10]:
```
67580
```

In [11]:
```
df.shape
```

Out[11]:
```
(2180, 31)
```

In [12]:
```
df.head()
```

Out[12]:

	Cr	Ni	Mo	Mn	Si	Nb	Ti	Zr	Ta	V	W	Cu	N	C	B	P	S	Co	Al	Sn	Pb	Solution_treatment_te
0	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
1	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
2	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
3	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	
4	18.7	10.69	0.47	1.56	0.62	0.01	0.04	0	0	0.0	0.0	0.17	0.031	0.062	0.0007	0.025	0.013	0.0	0.047	0	0	

3.1. Deal with missing data columns

In [13]:
```
df.isnull().sum()[df.isnull().sum()>0]
```

Out[13]:
```
W                                  1
Solution_treatment_temperature   184
Solution_treatment_time(s)      1097
Water_Quenched_after_s.t.        264
Air_Quenched_after_s.t.          264
Grains mm-2                     1517
Type of melting                  217
Size of ingot                   1517
dtype: int64
```

In [14]:
```
(df.isnull().sum()*100/df.shape[0])[df.isnull().sum()*100/df.shape[0]>0]
```

Out[14]:
```
W                                  0.045872
Solution_treatment_temperature     8.440367
Solution_treatment_time(s)        50.321101
Water_Quenched_after_s.t.         12.110092
Air_Quenched_after_s.t.           12.110092
Grains mm-2                       69.587156
Type of melting                    9.954128
Size of ingot                     69.587156
dtype: float64
```

- Columns such as `Grains mm-2` , `Size of ingot` contains large missing values. So these columns needs to be dropped.

In [15]:
```
df.drop(columns=["Grains mm-2","Size of ingot"],inplace=True)
```

- Columns such as `Product Form` , `Type of Melting` does not play any role for ultimate tensile strength. So these Columns also needs to be dropped

In [16]:
```
df.drop(columns=["Product form","Type of melting"],inplace=True)
```

In [17]:
```
df.describe()
```

Out[17]:

	Cr	Ni	Mo	Mn	Si	Nb	Ti	Zr	Ta	V	W	
count	2180.000000	2180.000000	2180.000000	2180.000000	2180.000000	2180.000000	2180.000000	2180.0	2180.0	2180.000000	2179.0	2180.0
mean	17.808335	12.580528	1.015940	1.463771	0.499528	0.095143	0.145684	0.0	0.0	0.002547	0.0	0.0
std	0.991134	5.152322	1.164922	0.235216	0.140637	0.256143	0.202533	0.0	0.0	0.009969	0.0	0.0
min	15.900000	8.400000	0.000000	0.610000	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.0
25%	17.110000	10.430000	0.000000	1.430000	0.400000	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.0
50%	17.700000	11.600000	0.110000	1.520000	0.490000	0.000000	0.014000	0.0	0.0	0.000000	0.0	0.0
75%	18.200000	12.210000	2.370000	1.610000	0.590000	0.010000	0.390000	0.0	0.0	0.000000	0.0	0.0
max	21.060000	34.450000	2.910000	1.820000	1.150000	0.950000	0.560000	0.0	0.0	0.057000	0.0	0.3

- Columns such as `Zr` , `TA` , `W` , `Sn` , `Pb` contains all values as `0` . So these Columns also needs to be dropped

In [18]:
```
df.drop(columns=["Zr","Ta","W","Sn","Pb"],inplace=True)
```

In [19]:
```
(df.isnull().sum()*100/df.shape[0])[df.isnull().sum()*100/df.shape[0]>0]
```

Out[19]:
```
Solution_treatment_temperature     8.440367
Solution_treatment_time(s)        50.321101
Water_Quenched_after_s.t.         12.110092
Air_Quenched_after_s.t.           12.110092
dtype: float64
```

- `Solution_treatment_time(s)` , `Water_Quenched_after_s.t.` , `Air_Quenched_after_s.t.` plays an `Important Role` as Properties of steel mainly depends on it. So records having `Null` value in `Solution_treatment_time(s)` , `Water_Quenched_after_s.t.` , `Air_Quenched_after_s.t.` cannot be used.

In [20]:
```
df.dropna(axis=0, subset=['Solution_treatment_time(s)'],inplace=True)
```

In [21]:
```
df.dropna(axis=0, subset=['Water_Quenched_after_s.t.'],inplace=True)
```

In [22]:
```
(df.isnull().sum()*100/df.shape[0])[df.isnull().sum()*100/df.shape[0]>0]
```

Out[22]:
```
Series([], dtype: float64)
```

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1037 entries, 66 to 2167
Data columns (total 22 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Cr                            1037 non-null   float64
 1   Ni                            1037 non-null   float64
 2   Mo                            1037 non-null   float64
 3   Mn                            1037 non-null   float64
 4   Si                            1037 non-null   float64
 5   Nb                            1037 non-null   float64
 6   Ti                            1037 non-null   float64
 7   V                             1037 non-null   float64
 8   Cu                            1037 non-null   float64
 9   N                             1037 non-null   float64
 10  C                             1037 non-null   float64
 11  B                             1037 non-null   float64
 12  P                             1037 non-null   float64
 13  S                             1037 non-null   float64
 14  Co                            1037 non-null   float64
 15  Al                            1037 non-null   float64
 16  Solution_treatment_temperature  1037 non-null   float64
 17  Solution_treatment_time(s)    1037 non-null   float64
 18  Water_Quenched_after_s.t.     1037 non-null   object 
 19  Air_Quenched_after_s.t.       1037 non-null   object 
 20  Temperature (K)               1037 non-null   int64  
 21  UTS (M Pa)                    1037 non-null   float64
dtypes: float64(19), int64(1), object(2)
memory usage: 186.3+ KB
```

In [24]:
```
df["Water_Quenched_after_s.t."].replace({"No":0,"Yes":1},inplace=True)
df["Air_Quenched_after_s.t."].replace({"No":0,"Yes":1},inplace=True)
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1037 entries, 66 to 2167
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Cr                              1037 non-null   float64
 1   Ni                              1037 non-null   float64
 2   Mo                              1037 non-null   float64
 3   Mn                              1037 non-null   float64
 4   Si                              1037 non-null   float64
 5   Nb                              1037 non-null   float64
 6   Ti                              1037 non-null   float64
 7   V                               1037 non-null   float64
 8   Cu                              1037 non-null   float64
 9   N                               1037 non-null   float64
 10  C                               1037 non-null   float64
 11  B                               1037 non-null   float64
 12  P                               1037 non-null   float64
 13  S                               1037 non-null   float64
 14  Co                              1037 non-null   float64
 15  Al                              1037 non-null   float64
 16  Solution_treatment_temperature  1037 non-null   float64
 17  Solution_treatment_time(s)      1037 non-null   float64
 18  Water_Quenched_after_s.t.       1037 non-null   int64  
 19  Air_Quenched_after_s.t.         1037 non-null   int64  
 20  Temperature (K)                 1037 non-null   int64  
 21  UTS (M Pa)                      1037 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 186.3 KB
```

- After data cleaning, no nulls are remained. So Exploratory Data analysis can be done

In [26]:
```
df.head(1)
```

Out[26]:

	Cr	Ni	Mo	Mn	Si	Nb	Ti	V	Cu	N	C	B	P	S	Co	Al	Solution_treatment_temperature	Solution_t
66	18.16	9.8	0.05	1.47	0.58	0.04	0.031	0.0	0.14	0.031	0.07	0.0003	0.022	0.013	0.0	0.015	1343.0	

4. Exploratory Data Analysis

4.1. Univariate Data Analysis

In [27]:
```
df.describe()
```

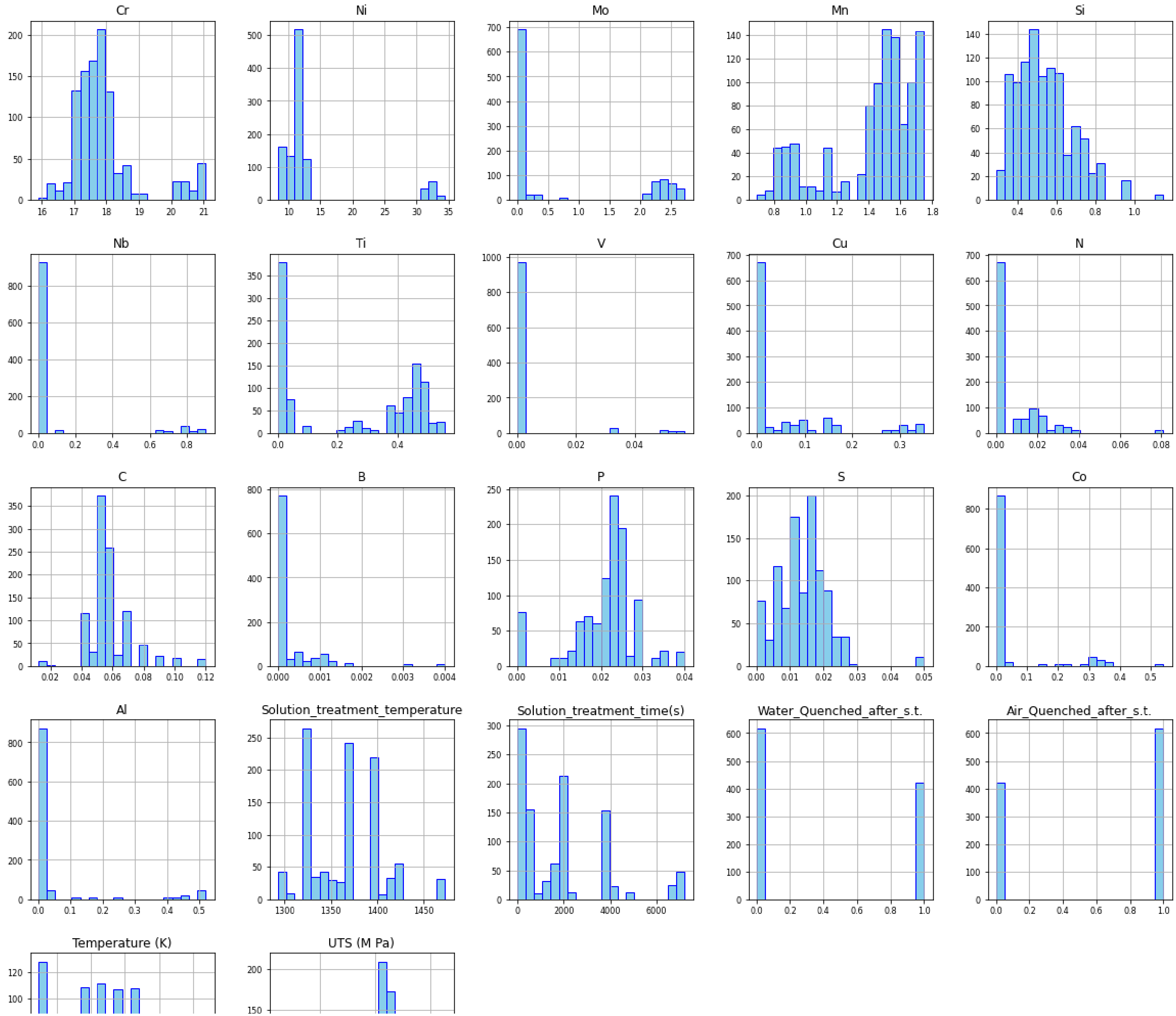
Out[27]:

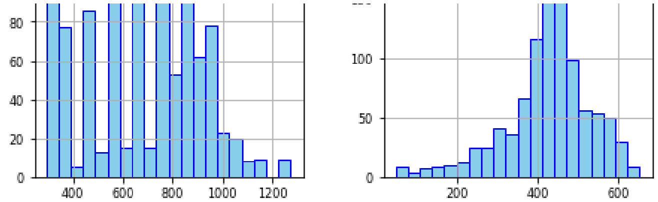
	Cr	Ni	Mo	Mn	Si	Nb	Ti	V	Cu	N	
count	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	1037.000000	10:
mean	17.901726	13.293983	0.705361	1.424079	0.538312	0.078581	0.244722	0.002847	0.055757	0.007875	
std	1.024191	6.324274	1.060158	0.280454	0.143843	0.226181	0.217176	0.011074	0.097583	0.013193	
min	15.900000	8.400000	0.000000	0.690000	0.290000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	17.270000	10.440000	0.000000	1.360000	0.440000	0.000000	0.011000	0.000000	0.000000	0.000000	
50%	17.710000	11.700000	0.040000	1.520000	0.520000	0.000000	0.290000	0.000000	0.000000	0.000000	
75%	18.020000	12.200000	2.190000	1.620000	0.610000	0.010000	0.460000	0.000000	0.080000	0.016000	
max	21.060000	34.450000	2.720000	1.750000	1.150000	0.900000	0.560000	0.057000	0.350000	0.081000	

a=[col for col in df.columns if df[col].dtypes!="object"] for i in a: skew = df[i].skew() plt.hist(df[i], label='Skew = %.3f' %(skew), bins=30,color='green') plt.title("Distribution Plot of "+i) plt.legend(loc='best') plt.show();

A. Distribution Plot

In [28]:
```
df.hist(bins=20,figsize=(20,20),color='skyblue',edgecolor='blue',xlabelsize=8,ylabelsize=8,);
plt.savefig('results/Distibution.png', bbox_inches='tight')
```


Cr
Ni
Mo
Mn
Si
Nb
Ti
V
Cu
N
C
B
P
S
Co
Al
Solution_treatment_temperature
Solution_treatment_time(s)
Water_Quenched_after_s.t.
Air_Quenched_after_s.t.
Temperature (K)
UTS (M Pa)

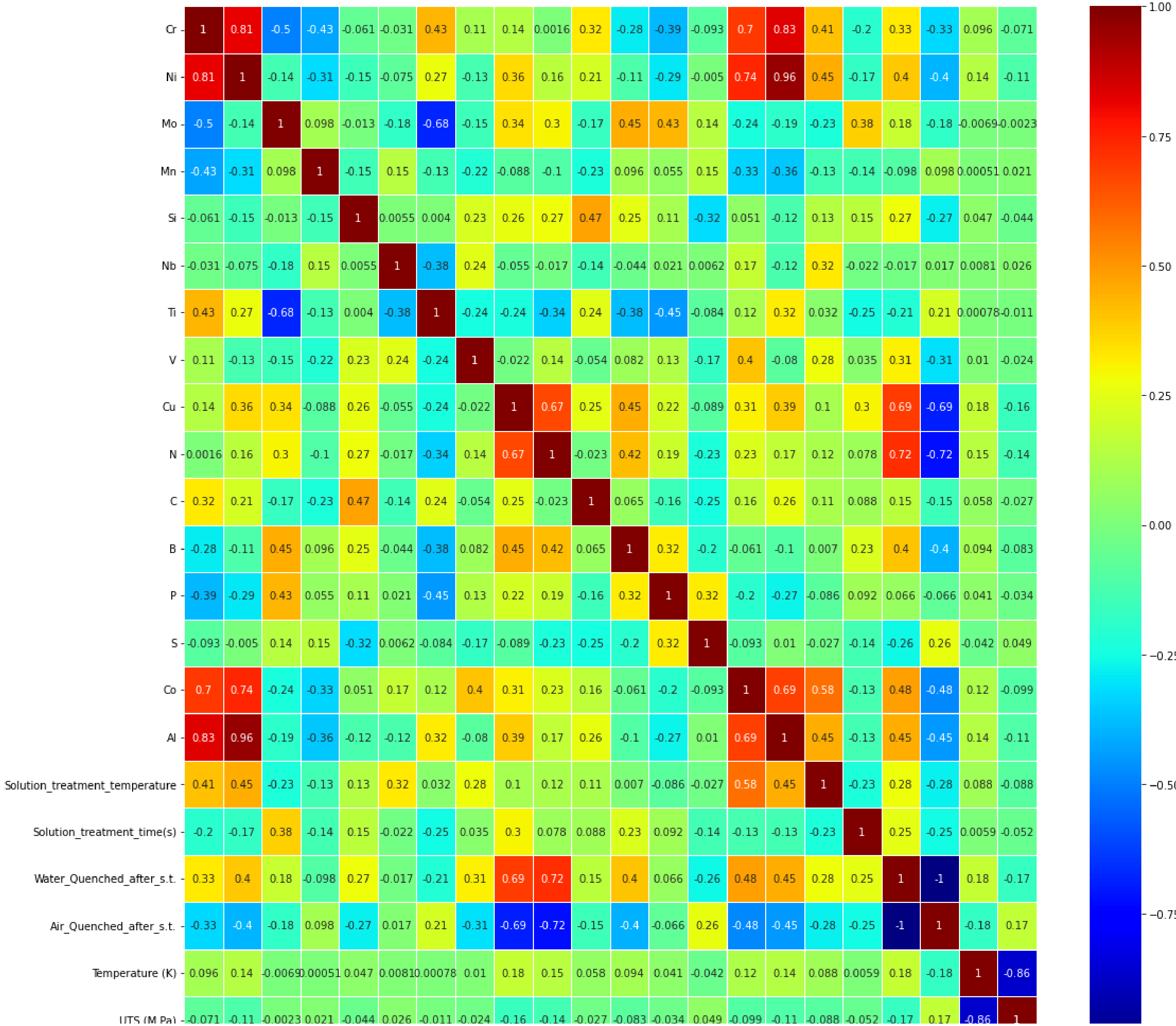


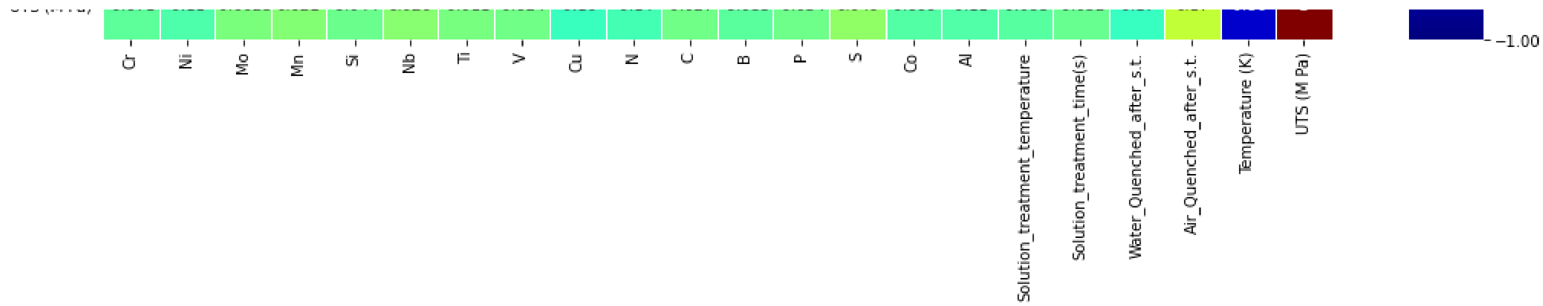
4.2. Bivariate Analysis

A. Pearson's Correlation Coefficient:

- Helps you find out the relationship between two quantities. It gives you the measure of the strength of association between two variables.
- The value of Pearson's Correlation Coefficient can be between -1 to +1. 1 means that they are highly correlated and 0 means no correlation.

In [29]:
```
plt.figure(figsize=(18,18))
sns.heatmap(data=df.corr(), cmap="jet", annot=True,linewidths=1, linecolor='white');
plt.savefig('results/correlation.png', bbox_inches='tight')
```


Cr 1 0.81 -0.5 -0.43 -0.061 -0.031 0.43 0.11 0.14 0.0016 0.32 -0.28 -0.39 -0.093 0.7 0.83 0.41 -0.2 0.33 -0.33 0.096 -0.071
Ni 0.81 1 -0.14 -0.31 -0.15 -0.075 0.27 -0.13 0.36 0.16 0.21 -0.11 -0.29 -0.005 0.74 0.96 0.45 -0.17 0.4 -0.4 0.14 -0.11
Mo -0.5 -0.14 1 0.098 -0.013 -0.18 -0.68 -0.15 0.34 0.3 -0.17 0.45 0.43 0.14 -0.24 -0.19 -0.23 0.38 0.18 -0.18 -0.0069 -0.0023
Mn -0.43 -0.31 0.098 1 -0.15 0.15 -0.13 -0.22 -0.088 -0.1 -0.23 0.096 0.055 0.15 -0.33 -0.36 -0.13 -0.14 -0.098 0.098 0.00051 0.021
Si -0.061 -0.15 -0.013 -0.15 1 0.0055 0.004 0.23 0.26 0.27 0.47 0.25 0.11 -0.32 0.051 -0.12 0.13 0.15 0.27 -0.27 0.047 -0.044
Nb -0.031 -0.075 -0.18 0.15 0.0055 1 -0.38 0.24 -0.055 -0.017 -0.14 -0.044 0.021 0.0062 0.17 -0.12 0.32 -0.022 -0.017 0.017 0.0081 0.026
Ti 0.43 0.27 -0.68 -0.13 0.004 -0.38 1 -0.24 -0.24 -0.34 0.24 -0.38 -0.45 -0.084 0.12 0.32 0.032 -0.25 -0.21 0.21 0.00078 -0.011
V 0.11 -0.13 -0.15 -0.22 0.23 0.24 -0.24 1 -0.022 0.14 -0.054 0.082 0.13 -0.17 0.4 -0.08 0.28 0.035 0.31 -0.31 0.01 -0.024
Cu 0.14 0.36 0.34 -0.088 0.26 -0.055 -0.24 -0.022 1 0.67 0.25 0.45 0.22 -0.089 0.31 0.39 0.1 0.3 0.69 -0.69 0.18 -0.16
N 0.0016 0.16 0.3 -0.1 0.27 -0.017 -0.34 0.14 0.67 1 -0.023 0.42 0.19 -0.23 0.23 0.17 0.12 0.078 0.72 -0.72 0.15 -0.14
C 0.32 0.21 -0.17 -0.23 0.47 -0.14 0.24 -0.054 0.25 -0.023 1 0.065 -0.16 -0.25 0.16 0.26 0.11 0.088 0.15 -0.15 0.058 -0.027
B -0.28 -0.11 0.45 0.096 0.25 -0.044 -0.38 0.082 0.45 0.42 0.065 1 0.32 -0.2 -0.061 -0.1 0.007 0.23 0.4 -0.4 0.094 -0.083
P -0.39 -0.29 0.43 0.055 0.11 0.021 -0.45 0.13 0.22 0.19 -0.16 0.32 1 0.32 -0.2 -0.27 -0.086 0.092 0.066 -0.066 0.041 -0.034
S -0.093 -0.005 0.14 0.15 -0.32 0.0062 -0.084 -0.17 -0.089 -0.23 -0.25 -0.2 0.32 1 -0.093 0.01 -0.027 -0.14 -0.26 0.26 -0.042 0.049
Co 0.7 0.74 -0.24 -0.33 0.051 0.17 0.12 0.4 0.31 0.23 0.16 -0.061 -0.2 -0.093 1 0.69 0.58 -0.13 0.48 -0.48 0.12 -0.099
Al 0.83 0.96 -0.19 -0.36 -0.12 -0.12 0.32 -0.08 0.39 0.17 0.26 -0.1 -0.27 0.01 0.69 1 0.45 -0.13 0.45 -0.45 0.14 -0.11
Solution_treatment_temperature 0.41 0.45 -0.23 -0.13 0.13 0.32 0.032 0.28 0.1 0.12 0.11 0.007 -0.086 -0.027 0.58 0.45 1 -0.23 0.28 -0.28 0.088 -0.088
Solution_treatment_time(s) -0.2 -0.17 0.38 -0.14 0.15 -0.022 -0.25 0.035 0.3 0.078 0.088 0.23 0.092 -0.14 -0.13 -0.13 -0.23 1 0.25 -0.25 0.0059 -0.052
Water_Quenched_after_s.t. 0.33 0.4 0.18 -0.098 0.27 -0.017 -0.21 0.31 0.69 0.72 0.15 0.4 0.066 -0.26 0.48 0.45 0.28 0.25 1 -1 0.18 -0.17
Air_Quenched_after_s.t. -0.33 -0.4 -0.18 0.098 -0.27 0.017 0.21 -0.31 -0.69 -0.72 -0.15 -0.4 -0.066 0.26 -0.48 -0.45 -0.28 -0.25 -1 1 -0.18 0.17
Temperature (K) 0.096 0.14 -0.0069 0.00051 0.047 0.0081 0.00078 0.01 0.18 0.15 0.058 0.094 0.041 -0.042 0.12 0.14 0.088 0.0059 0.18 -0.18 1 -0.86
UTS (M Pa) -0.071 -0.11 -0.0023 0.021 -0.044 0.026 -0.011 -0.024 -0.16 -0.14 -0.027 -0.083 -0.034 0.049 -0.099 -0.11 -0.088 -0.052 -0.17 0.17 -0.86 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



4.3. Feature Selection Technique

- dropping `Cr`, `Ni`, `Air_Quenched_after_s.t.` as there is high correlation greater than 0.75 or less than -0.75

In [30]:
```
df.drop(columns=["Cr","Ni","Air_Quenched_after_s.t."],inplace=True)
```

In [31]:
```
df.corr()["UTS (M Pa)"].sort_values(ascending=False)
```

Out[31]:
```
UTS (M Pa)                        1.000000
S                                 0.049276
Nb                                0.025647
Mn                                0.020746
Mo                               -0.002268
Ti                               -0.010942
V                                -0.024455
C                                -0.027047
P                                -0.034012
Si                               -0.044445
Solution_treatment_time(s)       -0.051794
B                                -0.083284
Solution_treatment_temperature   -0.088376
Co                               -0.098533
Al                               -0.108020
N                                -0.140141
Cu                               -0.157979
Water_Quenched_after_s.t.        -0.166155
Temperature (K)                  -0.860521
Name: UTS (M Pa), dtype: float64
```

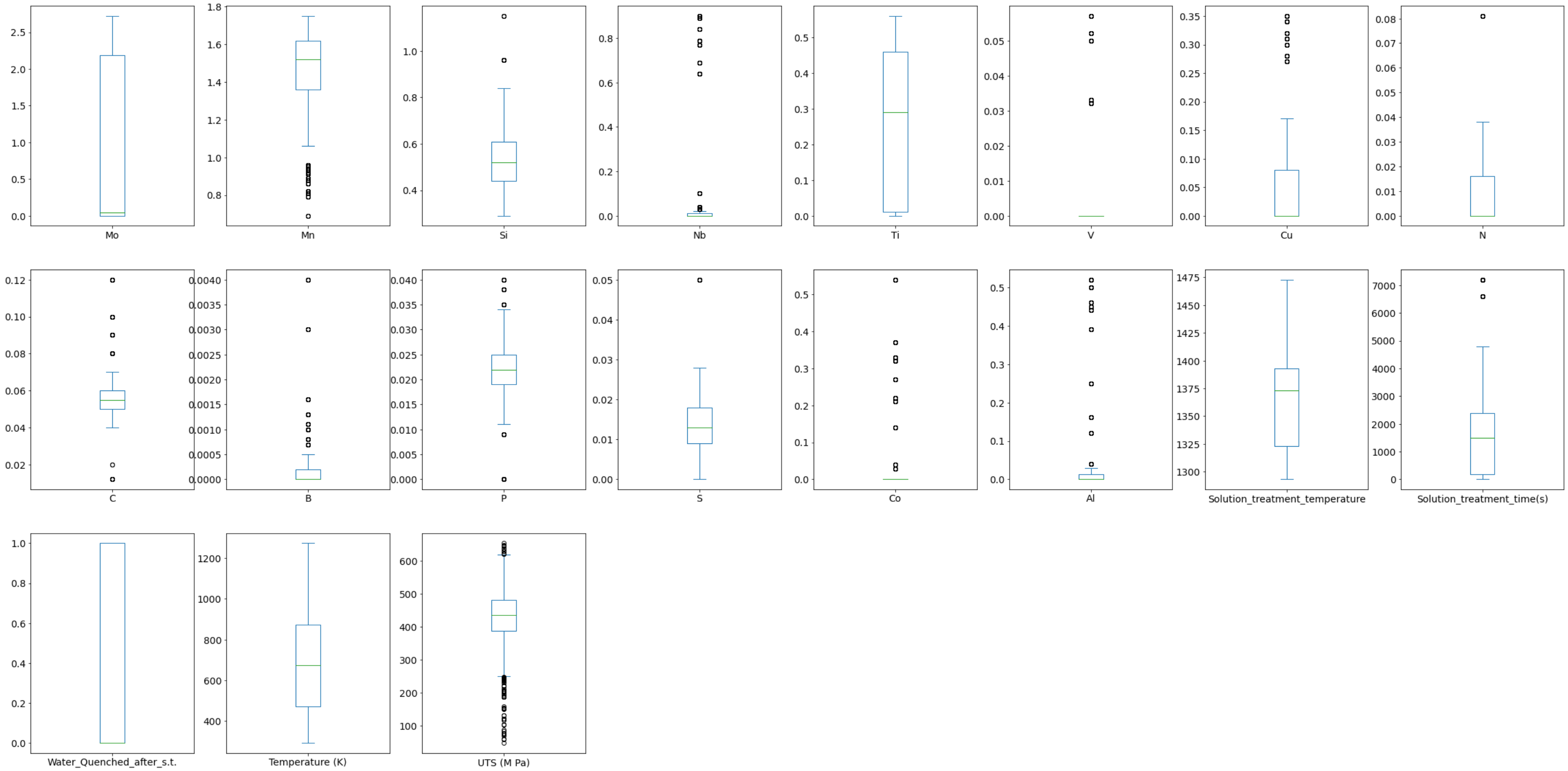
In [32]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1037 entries, 66 to 2167
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Mo                              1037 non-null   float64
 1   Mn                              1037 non-null   float64
 2   Si                              1037 non-null   float64
 3   Nb                              1037 non-null   float64
 4   Ti                              1037 non-null   float64
 5   V                               1037 non-null   float64
 6   Cu                              1037 non-null   float64
 7   N                               1037 non-null   float64
 8   C                               1037 non-null   float64
 9   B                               1037 non-null   float64
 10  P                               1037 non-null   float64
 11  S                               1037 non-null   float64
 12  Co                              1037 non-null   float64
 13  Al                              1037 non-null   float64
 14  Solution_treatment_temperature  1037 non-null   float64
 15  Solution_treatment_time(s)      1037 non-null   float64
 16  Water_Quenched_after_s.t.       1037 non-null   int64  
 17  Temperature (K)                 1037 non-null   int64  
 18  UTS (M Pa)                      1037 non-null   float64
dtypes: float64(17), int64(2)
memory usage: 162.0 KB
```

In [33]:
```
df["Water_Quenched_after_s.t."]=df["Water_Quenched_after_s.t."].astype(float)
df["Temperature (K)"]=df["Temperature (K)"].astype(float)
```

4.4. Outlier Detection

In [34]:
```
df.plot(kind='box',figsize=(40,20), layout=(3,8),subplots=True,fontsize="14");
plt.savefig("results/boxplot.png", bbox_inches='tight')
```



5. Machine Learning Algorithm

5.1. Applying Standard Scaler

- For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1 (or the variance).
- x_scaled = x – mean/std_dev
- However, Standard Scaler assumes that the distribution of the variable is normal. Thus, in case, the variables are not normally distributed, we either choose a different scaler or first, convert the variables to a normal distribution and then apply this scaler

In [35]:
```
# standardization
X=df.iloc[:,0:-1]
y=df.iloc[:,-1]
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
std=scaler.fit_transform(X)
```

In [36]:
```
std[1]   # even after applying Standard Scaler, there is no change in accuracy.going with data as it is
```

Out[36]:
```
array([-0.6184707 ,  0.16381658,  0.28995365, -0.17066026, -0.98457038,
       -0.25719297,  0.86370895,  1.75367   ,  0.795903  ,  0.11268594,
        0.14703692, -0.066944  , -0.39820798, -0.25943189, -0.54442346,
       -0.61881505,  1.21204275, -1.25936014])
```

5.2. Train Test Split

In [37]:
```
from sklearn.model_selection import train_test_split
```

In [38]:
```
df_train_x,df_test_x,df_train_y,df_test_y=train_test_split(X,y,test_size=0.2,random_state=42)
```

5.3. Defing Function for all Algorithm

In [39]:
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import r2_score
from sklearn import metrics
```

In [40]:
```
result=pd.DataFrame()
train_MAE=[]
train_MSE=[]
train_RMSE=[]
train_R2=[]
test_MAE=[]
test_MSE=[]
test_RMSE=[]
test_R2=[]
CV_train_R2=[]
CV_test_R2=[]


def print_score(model, X_train, y_train, X_test, y_test, train=True,test=True):
    '''
    print the r2_score
    '''
    '''
        training performance
    '''
```

```
    print('\033[1m'+"Train Result:"+'\033[0m')

    print("mean absolute error: {0:.4f}".format(metrics.mean_absolute_error(y_train, model.predict(X_train))))
    train_MAE.append(metrics.mean_absolute_error(y_train, model.predict(X_train)))
    print("mean squared error: {0:.4f}".format(metrics.mean_squared_error(y_train, model.predict(X_train))))
    train_MSE.append(metrics.mean_squared_error(y_train, model.predict(X_train)))
    print("RMSE: {0:.4f}".format(np.sqrt(metrics.mean_squared_error(y_train, model.predict(X_train)))))
    train_RMSE.append(metrics.mean_squared_error(y_train, model.predict(X_train)))
    print("r2_score: {0:.4f}".format(r2_score(y_train, model.predict(X_train))))
    train_R2.append(r2_score(y_train, model.predict(X_train)))
    print("\n")
    '''
    test performance
    '''
    print('\033[1m'+"Test Result:"+'\033[0m')

    print("mean absolute error: {0:.4f}".format(metrics.mean_absolute_error(y_test, model.predict(X_test))))
    test_MAE.append(metrics.mean_absolute_error(y_test, model.predict(X_test)))
    print("mean squared error: {0:.4f}".format(metrics.mean_squared_error(y_test, model.predict(X_test))))
    test_MSE.append(metrics.mean_squared_error(y_test, model.predict(X_test)))
    print("RMSE: {0:.4f}".format(np.sqrt(metrics.mean_squared_error(y_test, model.predict(X_test)))))
    test_RMSE.append(np.sqrt(metrics.mean_squared_error(y_test, model.predict(X_test))))
    print("r2_score: {0:.4f}\n".format(r2_score(y_test, model.predict(X_test))))
    test_R2.append(r2_score(y_test, model.predict(X_test)))

    '''
    Cross Validation
    '''
    print('\033[1m'+"r2_Score after Cross Validation\n"+'\033[0m')

    folds_train = KFold(n_splits = 5, shuffle = True, random_state = 42)
    scores_train = cross_val_score(model, X_train, y_train, scoring='r2', cv=folds_train)
    print("Training r2_score: \t {0:.4f}".format(np.mean(scores_train)))
    CV_train_R2.append(np.mean(scores_train))
    folds_test = KFold(n_splits = 5, shuffle = True, random_state = 42)
    scores_test = cross_val_score(model, X_test, y_test, scoring='r2', cv=folds_test)
    print("Testing r2_score: \t {0:.4f}".format(np.mean(scores_test)))
    CV_test_R2.append(np.mean(scores_test))
```

5.4. Applying Function to all Algorithms

In [41]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost.sklearn import XGBRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
```

In [42]:

```
ml_model={  "Linear Regression" : LinearRegression(),
            "DecisionTree Regressor": DecisionTreeRegressor(max_depth=2),
            "RandomForest Regressor" :  RandomForestRegressor(n_estimators=100),
            "KNeighbors Regressor" : KNeighborsRegressor(),
            "Support Vector Regressor" : SVR(),
            "Bagging Regressor with DT Regressor" : BaggingRegressor(DecisionTreeRegressor(),n_estimators=500,bootstrap
            "AdaBoostRegressor" : AdaBoostRegressor(random_state=42),
            "GradientBoosting Regressor" :GradientBoostingRegressor(random_state=40,learning_rate=0.1),
            "XGBRegressor":XGBRegressor(random_state=42,learning_rate=0.1),
            "Ridge":Ridge(),
            "Lasso":Lasso()
         }

for i in range(len(list(ml_model))):
    model = list(ml_model.values())[i]
    model_fit=model.fit(df_train_x, df_train_y)
    print("<------------------------------------------------------------------------------->")
    print('\033[1m' +str(i+1)+")"+ list(ml_model.keys())[i] + '\033[0m')
    print("<------------------------------------------------------------------------------->")
    print("\n")
    print_score(model_fit,df_train_x,df_train_y,df_test_x,df_test_y,train=True,test=True)
    print("\n")
```

```
<----------------------------------------------------------------------------->
1)Linear Regression
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 37.2805
mean squared error: 2573.1351
RMSE: 50.7261
r2_score: 0.7517


Test Result:
mean absolute error: 38.4006
mean squared error: 2860.0636
RMSE: 53.4796
r2_score: 0.7287

r2_Score after Cross Validation

Training r2_score:       0.7319
Testing r2_score:        0.6324


<----------------------------------------------------------------------------->
2)DecisionTree Regressor
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 35.7230
mean squared error: 2151.6449
RMSE: 46.3858
r2_score: 0.7923


Test Result:
mean absolute error: 37.0037
mean squared error: 2283.5113
RMSE: 47.7861
r2_score: 0.7834

r2_Score after Cross Validation

Training r2_score:       0.7885
```

```
Testing r2_score:        0.7394


<----------------------------------------------------------------------------->
3)RandomForest Regressor
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 5.7710
mean squared error: 133.8778
RMSE: 11.5706
r2_score: 0.9871


Test Result:
mean absolute error: 15.9355
mean squared error: 793.4335
RMSE: 28.1680
r2_score: 0.9247

r2_Score after Cross Validation

Training r2_score:       0.9108
Testing r2_score:        0.9054


<----------------------------------------------------------------------------->
4)KNeighbors Regressor
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 19.1740
mean squared error: 947.0786
RMSE: 30.7746
r2_score: 0.9086


Test Result:
mean absolute error: 24.7172
mean squared error: 1495.3877
RMSE: 38.6702
r2_score: 0.8581
```

```
r2_Score after Cross Validation

Training r2_score:       0.8311
Testing r2_score:        0.6605


<------------------------------------------------------------------------------>
5)Support Vector Regressor
<------------------------------------------------------------------------------>


Train Result:
mean absolute error: 68.6099
mean squared error: 9583.2247
RMSE: 97.8939
r2_score: 0.0751


Test Result:
mean absolute error: 71.9338
mean squared error: 9956.7689
RMSE: 99.7836
r2_score: 0.0554

r2_Score after Cross Validation

Training r2_score:       0.0515
Testing r2_score:        -0.0232


<------------------------------------------------------------------------------>
6)Bagging Regressor with DT Regressor
<------------------------------------------------------------------------------>


Train Result:
mean absolute error: 5.6520
mean squared error: 123.7761
RMSE: 11.1255
r2_score: 0.9881


Test Result:
mean absolute error: 15.5861
mean squared error: 738.7336
```

```
RMSE: 27.1797
r2_score: 0.9299

r2_Score after Cross Validation

Training r2_score:       0.9109
Testing r2_score:        0.9087


<----------------------------------------------------------------------------->
7)AdaBoostRegressor
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 30.5947
mean squared error: 1572.7056
RMSE: 39.6574
r2_score: 0.8482


Test Result:
mean absolute error: 31.7162
mean squared error: 1876.9970
RMSE: 43.3243
r2_score: 0.8219

r2_Score after Cross Validation

Training r2_score:       0.8159
Testing r2_score:        0.8698


<----------------------------------------------------------------------------->
8)GradientBoosting Regressor
<----------------------------------------------------------------------------->


Train Result:
mean absolute error: 11.5615
mean squared error: 429.9853
RMSE: 20.7361
r2_score: 0.9585
```

```
Test Result:
mean absolute error: 15.8894
mean squared error: 801.8524
RMSE: 28.3170
r2_score: 0.9239

r2_Score after Cross Validation

Training r2_score:       0.9247
Testing r2_score:        0.9258


<------------------------------------------------------------------------------>
9)XGBRegressor
<------------------------------------------------------------------------------>


Train Result:
mean absolute error: 5.2992
mean squared error: 87.3888
RMSE: 9.3482
r2_score: 0.9916


Test Result:
mean absolute error: 12.1371
mean squared error: 416.1206
RMSE: 20.3990
r2_score: 0.9605

r2_Score after Cross Validation

Training r2_score:       0.9213
Testing r2_score:        0.9201


<------------------------------------------------------------------------------>
10)Ridge
<------------------------------------------------------------------------------>


Train Result:
mean absolute error: 37.4549
mean squared error: 2594.9724
RMSE: 50.9409
```

```
r2_score: 0.7496


Test Result:
mean absolute error: 38.0568
mean squared error: 2799.0412
RMSE: 52.9060
r2_score: 0.7345

r2_Score after Cross Validation

Training r2_score:       0.7390
Testing r2_score:        0.6788


<------------------------------------------------------------------------------>
11)Lasso
<------------------------------------------------------------------------------>


Train Result:
mean absolute error: 38.0894
mean squared error: 2642.8191
RMSE: 51.4084
r2_score: 0.7449


Test Result:
mean absolute error: 38.3576
mean squared error: 2798.7509
RMSE: 52.9032
r2_score: 0.7345

r2_Score after Cross Validation

Training r2_score:       0.7412
Testing r2_score:        0.6827
```

In [43]:

```
result["model"]=list(ml_model.keys())
result["train_MAE"]=train_MAE
result["train_MSE"]=train_MSE
result["train_RMSE"]=train_RMSE
result["train_R2"]=train_R2
```

```
result["test_MAE"]=test_MAE
result["test_MSE"]=test_MSE
result["test_RMSE"]=test_RMSE
result["test_R2"]=test_R2
result["CV_train_R2"]=CV_train_R2
result["CV_test_R2"]=CV_test_R2
result
```

Out[43]:

	model	train_MAE	train_MSE	train_RMSE	train_R2	test_MAE	test_MSE	test_RMSE	test_R2	CV_train_R2	CV_test_R2
0	Linear Regression	37.280469	2573.135140	2573.135140	0.751662	38.400633	2860.063605	53.479563	0.728678	0.731934	0.632437
1	DecisionTree Regressor	35.722974	2151.644932	2151.644932	0.792341	37.003714	2283.511321	47.786100	0.783373	0.788510	0.739387
2	RandomForest Regressor	5.770987	133.877751	133.877751	0.987079	15.935493	793.433453	28.167951	0.924730	0.910818	0.905360
3	KNeighbors Regressor	19.174042	947.078589	947.078589	0.908596	24.717249	1495.387665	38.670243	0.858139	0.831096	0.660543
4	Support Vector Regressor	68.609873	9583.224717	9583.224717	0.075104	71.933820	9956.768856	99.783610	0.055445	0.051469	-0.023250
5	Bagging Regressor with DT Regressor	5.652013	123.776065	123.776065	0.988054	15.586060	738.733578	27.179654	0.929920	0.910900	0.908735
6	AdaBoostRegressor	30.594711	1572.705564	1572.705564	0.848215	31.716215	1876.996977	43.324323	0.821938	0.815922	0.869760
7	GradientBoosting Regressor	11.561496	429.985285	429.985285	0.958501	15.889350	801.852391	28.316998	0.923932	0.924669	0.925824
8	XGBRegressor	5.299249	87.388818	87.388818	0.991566	12.137144	416.120565	20.399033	0.960524	0.921277	0.920149
9	Ridge	37.454883	2594.972355	2594.972355	0.749554	38.056790	2799.041177	52.905965	0.734467	0.739009	0.678840
10	Lasso	38.089424	2642.819086	2642.819086	0.744936	38.357630	2798.750877	52.903222	0.734495	0.741187	0.682731

In [44]:

```
test_R2
```

Out[44]:
```
[0.7286782745320621,
 0.7833732681489545,
 0.9247304384752162,
 0.8581391124571388,
 0.055444885555567584,
 0.9299195763524047,
 0.8219375059817797,
 0.9239317706656405,
 0.9605244619149088,
 0.7344672053711345,
 0.7344947449129862]
```

5.5. Observations:

- We got pretty good results for all model but it that `XGBRegressor` is the best model.
- Train and test errors are also quiet similar, which means our model is not overfitted or underfitted.

6. Hyper Parameter Tuning of `XGBRegressor` Model

6.1. Without Tuning

In [45]:
```
xgboost=XGBRegressor(random_state=42,learning_rate=0.10)
xgboost.fit(df_train_x,df_train_y)
```

Out[45]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

In [46]:
```
print_score(xgboost, df_train_x, df_train_y, df_test_x, df_test_y, train=True,test=True)
```

```
Train Result:
mean absolute error: 5.2992
mean squared error: 87.3888
RMSE: 9.3482
r2_score: 0.9916


Test Result:
mean absolute error: 12.1371
mean squared error: 416.1206
RMSE: 20.3990
r2_score: 0.9605

r2_Score after Cross Validation

Training r2_score:       0.9213
Testing r2_score:        0.9201
```

6.2 With Tuning

In [47]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [48]:
```
base_score=[0.25,0.5,0.75,1]
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster=['gbtree','gblinear']
learning_rate=[0.05,0.1,0.15,0.20]
min_child_weight=[1,2,3,4]

# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
    'min_child_weight':min_child_weight,
    'booster':booster,
    'base_score':base_score
    }

# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=XGBRegressor(),
            param_distributions=hyperparameter_grid,
            cv=5, n_iter=50,
            scoring = 'neg_mean_absolute_error',n_jobs = 4,
```

```
            verbose = 5,
            return_train_score = True,
            random_state=42)

random_cv.fit(df_train_x,df_train_y)

random_cv.best_estimator_
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

Out[48]:

```
XGBRegressor(base_score=1, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=15, max_leaves=None,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=900, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [49]:

```
xgboost1=XGBRegressor(base_score=0.25, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=10, max_leaves=None,
             min_child_weight=3, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None)
xgboost1.fit(df_train_x,df_train_y)
```

Out[49]:

```
XGBRegressor(base_score=0.25, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=10, max_leaves=None,
             min_child_weight=3, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [50]:
```
print_score(xgboost1, df_train_x, df_train_y, df_test_x, df_test_y, train=True,test=True)
```

```
Train Result:
mean absolute error: 2.9894
mean squared error: 50.9728
RMSE: 7.1395
r2_score: 0.9951


Test Result:
mean absolute error: 11.8445
mean squared error: 475.7602
RMSE: 21.8119
r2_score: 0.9549

r2_Score after Cross Validation

Training r2_score:       0.9286
Testing r2_score:        0.9143
```

- After tuning R2 score is decreasing, we will keep XGBRegressor model without tuning

In [51]:
```
#Actual value and the predicted value
mlr_diff = pd.DataFrame({'Actual value': df_test_y, 'Predicted value': xgboost.predict(df_test_x)})
mlr_diff["error test"]=mlr_diff['Actual value']-mlr_diff['Predicted value']
mlr_diff.head(10)
```

Out[51]:

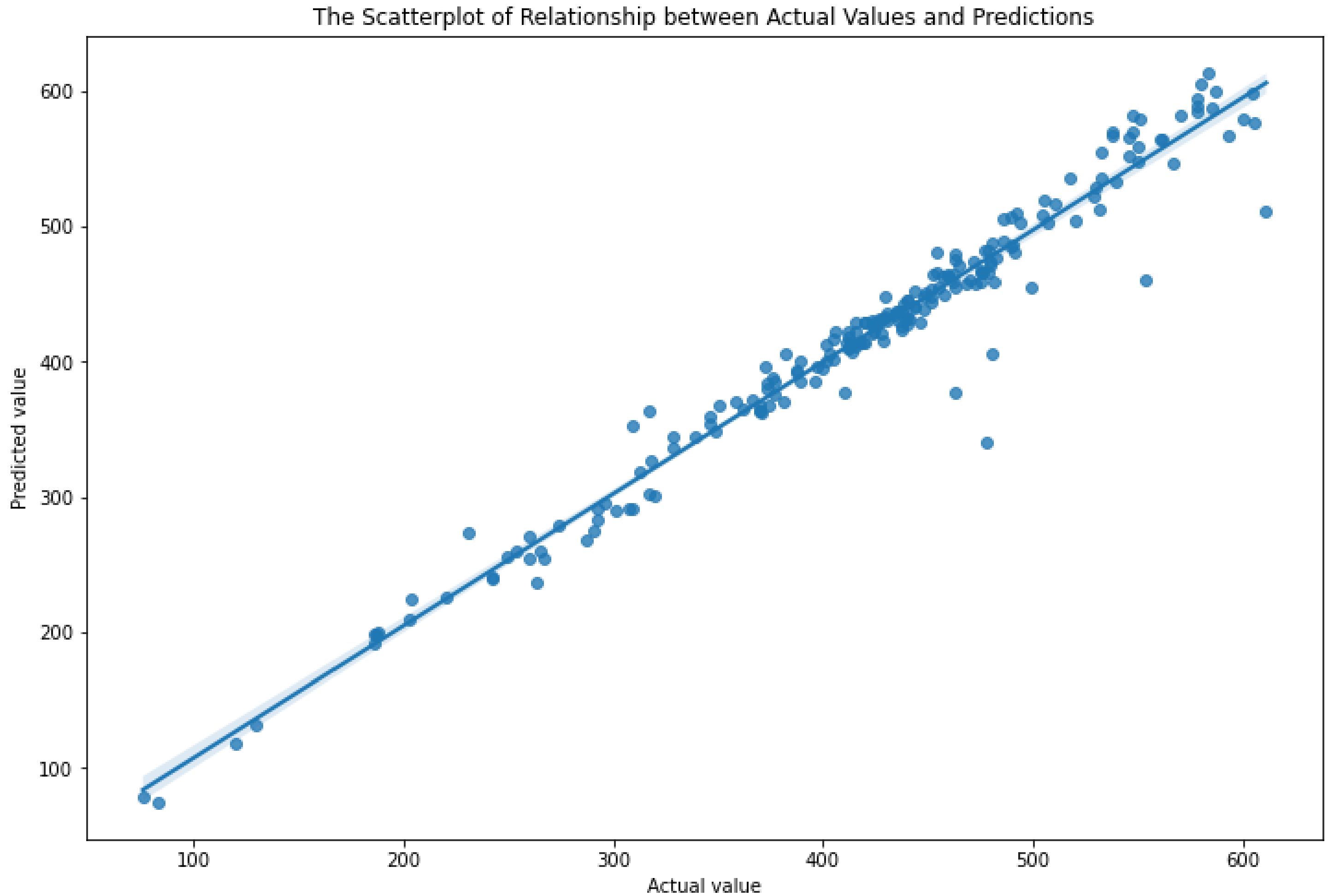
	Actual value	Predicted value	error test
1650	401.091985	400.132721	0.959264
307	242.000000	241.348618	0.651382
1712	517.791120	535.467041	-17.675921
1070	454.047895	465.588684	-11.540789
1569	316.754795	302.235809	14.518986
1506	456.989890	463.654358	-6.664468
299	504.000000	508.318787	-4.318787
1554	585.457005	587.062805	-1.605800
1781	586.437670	599.719604	-13.281934
2063	443.260580	439.846344	3.414236

7. Checking model with and without outlier

7.1. With Outliers

In [52]:

```
plt.figure(figsize=(12,8))
plt.xlabel("Actual Values")
plt.ylabel("Predicted values")
plt.title("The Scatterplot of Relationship between Actual Values and Predictions")
sns.regplot(data=mlr_diff,x=mlr_diff["Actual value"],y=mlr_diff["Predicted value"])
plt.savefig('results/Actual Vs Predicted with outliers.png', bbox_inches='tight')
```

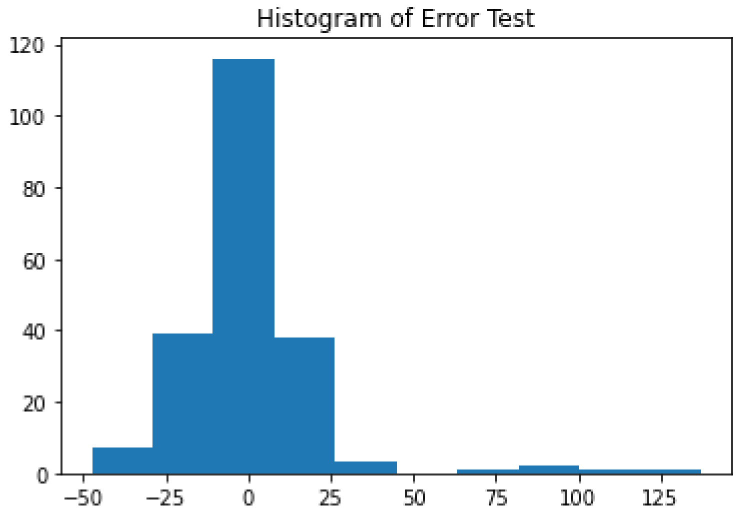


In [53]:
```
print(np.mean(mlr_diff["error test"]))
print(np.median(mlr_diff["error test"]))
```

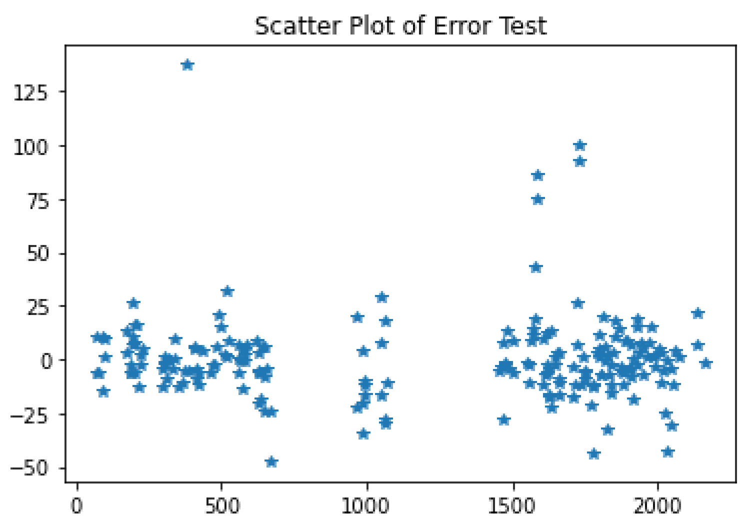
```
0.3169631383103575
-2.285627612304694
```

- mean of error is nearly equal to zero

In [54]:
```
plt.title("Histogram of Error Test")
plt.hist(mlr_diff["error test"])
plt.savefig('results/Histogram of Error Test with outliers.png', bbox_inches='tight')
```



In [55]:
```
plt.title("Scatter Plot of Error Test")
plt.plot(mlr_diff["error test"],"*")
plt.savefig('results/Histogram of Error Test with outliers.png', bbox_inches='tight')
```



7.2. Without Outliers

In [56]:
```
def remove_outliers(df,  col, k  ):
    mean = df[col].mean()
    global df1
```

```
    sd = df[col].std()
    final_list = [x for x in df[col] if (x > mean - k * sd)]
    final_list = [x for x in final_list if (x < mean + k * sd)]
    df1 = df.loc[df[col].isin(final_list)] ; print(df1.shape)
    print("Number of outliers removed == >"    ,  df.shape[0]  - df1.shape[0])
```

In [57]:
```
remove_outliers(df,"UTS (M Pa)",3)
```

```
(1024, 19)
Number of outliers removed == > 13
```

In [58]:
```
# standardization
X1=df1.iloc[:,0:-1]
y1=df1.iloc[:,-1]
```

In [59]:
```
from sklearn.model_selection import train_test_split
```

In [60]:
```
df_train_x1,df_test_x1,df_train_y1,df_test_y1=train_test_split(X1,y1,test_size=0.2,random_state=42)
```

In [61]:
```
xgboost2=XGBRegressor(random_state=42,learning_rate=0.30)
xgboost2.fit(df_train_x1,df_train_y1)
```

Out[61]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.3, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

In [62]:
```
print_score(xgboost2, df_train_x1, df_train_y1, df_test_x1, df_test_y1, train=True,test=True)
```

```
Train Result:
mean absolute error: 1.5961
mean squared error: 11.9956
RMSE: 3.4635
r2_score: 0.9987


Test Result:
mean absolute error: 11.4476
mean squared error: 395.7321
RMSE: 19.8930
r2_score: 0.9512

r2_Score after Cross Validation

Training r2_score:       0.9553
Testing r2_score:        0.9197
```

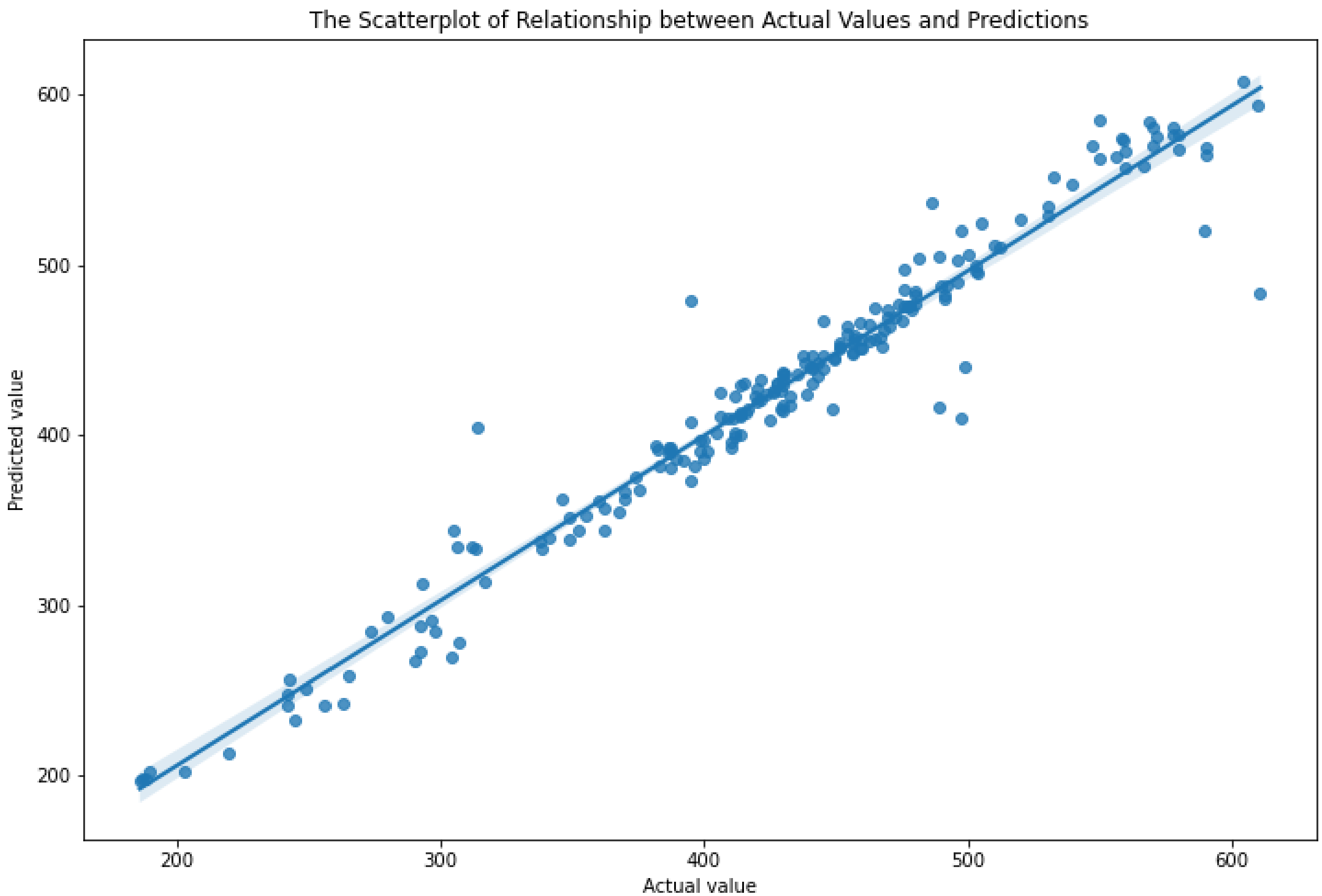
In [63]:
```
#Actual value and the predicted value
mlr_diff2 = pd.DataFrame({'Actual value': df_test_y1, 'Predicted value': xgboost2.predict(df_test_x1)})
mlr_diff2["error test"]=mlr_diff2['Actual value']-mlr_diff2['Predicted value']
mlr_diff2.head(10)
```

Out[63]:

	Actual value	Predicted value	error test
1566	456.989890	456.531891	0.457999
667	382.459350	391.125153	-8.665803
1069	467.777205	451.902863	15.874342
97	292.000000	272.238220	19.761780
1659	375.594695	367.559113	8.035582
1628	405.014645	400.644196	4.370449
1072	441.299250	430.190155	11.109095
1810	437.376590	446.642273	-9.265683
208	307.000000	278.298737	28.701263
1791	423.647280	423.986542	-0.339262

In [64]:
```
plt.figure(figsize=(12,8))
plt.xlabel("Actual Values")
```

```
plt.ylabel("Predicted values")
plt.title("The Scatterplot of Relationship between Actual Values and Predictions")
sns.regplot(data=mlr_diff2,x=mlr_diff2["Actual value"],y=mlr_diff2["Predicted value"])
plt.savefig('results/Actual Vs Predicted without outliers.png', bbox_inches='tight')
```

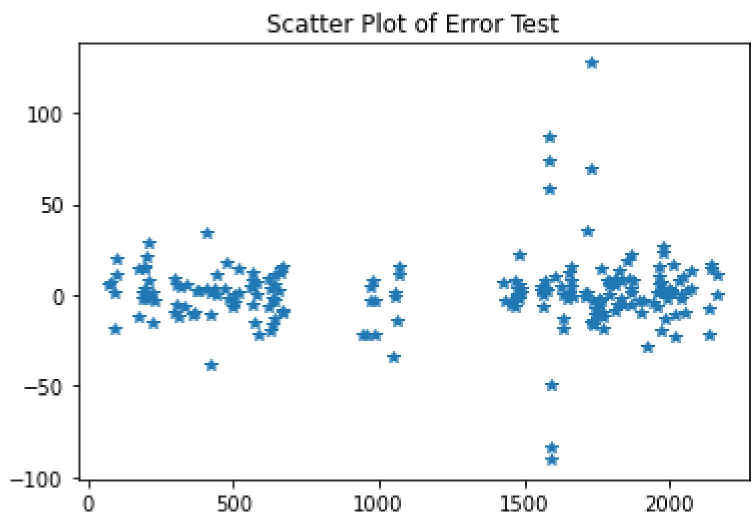


In [65]:
```
print(np.mean(mlr_diff2["error test"]))
print(np.median(mlr_diff2["error test"]))
```

```
1.3841781638957689
1.1009521484375
```

In [66]:
```
plt.title("Histogram of Error Test")
plt.hist(mlr_diff2["error test"])
plt.savefig('results/Histogram of Error Test without outliers.png', bbox_inches='tight')
```

In [67]:
```
plt.title("Scatter Plot of Error Test")
plt.plot(mlr_diff2["error test"],"*")
plt.savefig('results/Histogram of Error Test without outliers.png', bbox_inches='tight')
```



In [68]:
```
df_test_x1.head(1)
```

Out[68]:

	Mo	Mn	Si	Nb	Ti	V	Cu	N	C	B	P	S	Co	Al	Solution_treatment_temperature	Solution_treatment_time(s)	Wa
1566	0.0	0.81	0.63	0.0	0.5	0.0	0.0	0.0	0.06	0.0	0.028	0.007	0.0	0.0	1403.0	3600.0	

In [69]:
```
df_test_y1.head(1)
```

Out[69]:
```
1566    456.98989
Name: UTS (M Pa), dtype: float64
```

8. Predictions for new Data

In [70]:
```
# generating predictions for new Data
l=[(0.0,1.66,0.55,0.0,0.25,0.0,0.0,0.0,0.05,0.0,0.022,0.013,0.0,0.0,1323.0,1500.0,0,823)]
i=np.array(l)
y_pred = xgboost2.predict(i)
# creating table with test & predicted for test
print('predictions for new Data :',y_pred)
```

```
predictions for new Data : [401.32297]
```

9. Pickling the Model file for Deployement

In [71]:
```
import pickle
```

In [72]:
```
pickle.dump(xgboost,open("model.pkl","wb"))
```