**DS 7333; Case Study #1: Linear Regression – Super Conductivity**

Allen Miller, Vijayasrikanth Kaniti, Venkata ‘MG’ Vanga.

September 6, 2021

Professor: Dr. R. D. Slater

# Introduction

The case study focuses using the linear regression principals to predict the new superconductors and the temperature at which the predicted superconductors operate from the input data. The input data consists of two data sets with 21264 rows of data. There are 82 columns (variables) in the first data set ‘train.csv’ which include features like the number of elements contained in the superconductor alloy, the mean atomic mass, etc. The second data set ‘unique\_m.csv’ contains 88 columns (variables) which represent the pure element distribution for each superconductor alloy rows. The material column in the ‘unique\_m.csv’ represent the alloy name.

This case study combines both the data sets to evaluate the best features that help in predicting superconductor critical temperature. The combined data set is sleuthed for missing values, duplicated rows and any other statistical incongruities and found to be not having any of these. The linear regression in predicting the critical temperatures can be represented as shown in Eq. 1

Where,

– represents the predicted critical temperature of the super conductor alloy.

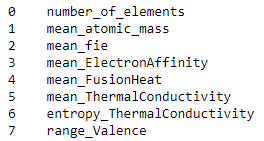
– represents the intercept of the linear regression.

– represent the coefficients of different features used in prediction using linear regression.

– represent the corresponding features used in prediction.

# Methods

As described in the introduction the combined data set is prepared for the analysis. First step that was used was to eliminate the rows of elements that had no contribution to the superconductor alloys. There are 9 elements that have a zero in all the rows, they are listed as:   
[He; Ne; Ar; Kr; Xe; Pm; Po;At; Rn] – The first five are inert gases and the later four radioactive elements, research into the chemical properties indicate that there is very less chance of these elements contributing to the superconductivity.   
The second step was to use a standard scalar to scale all the remaining variables. In the third step the correlation between the elemental attributes is determined and the elemental attributes which have an absolute correlation greater than 0.6 are removed from the features. Limiting the absolute correlation to greater than 0.6 reduced the elemental attributes columns from 81 attributes to 8 attributes. The heat map for correlation is presented in Fig. A1 (last page of report for reference only). The only elemental attributes that were finally used are:



The fourth and final step consists of running three linear regression models:

i. Baseline Linear Regression (LR).  
ii. L1 (Least Absolute Shrinkage and Selection Operator - LASSO) model.  
iii. L2 (Ridge) model.

The later models provide a way to eliminate the overfitting from the baseline linear regression model using the following equations for the loss function (:

Where, The first term in both Eq.2 and Eq.3 represents the Mean Squared Error (MSE).

– represents the strength of the penalty – depends on the slope function.

– represents the slope function.

Note: Eq. 3, depends on the square of the slope function, further discussion regarding regularization is provided in detail below.

Regularization is the most used technique to penalize complex models in machine learning, it is deployed for reducing overfitting. Also, it enhances the performance of models for new inputs.

Both L1 and L2 can add a penalty to the cost depending upon the model complexity, so at the place of computing the cost by using a loss function, there will be an auxiliary component, known as regularization terms, added in order to panelizing complex models.

By adding regularization term, the value of weights matrices reduces by assuming that a neural network having less weights makes simpler models. And hence, it reduces the overfitting to a certain level.

**L1 regularization:** Adds an L1 penalty that is equal to the absolute value of the magnitude of coefficient, or simply restricting the size of coefficients.

**L2 Regularization:** Adds an L2 penalty which is equal to the square of the magnitude of coefficients.

Different values were iterated to get the lowest MSE and the and are used as the final values for L1 and L2 models respectively. The MSE with these optimal values are shown in Table. 1

All the analysis is carried out in JUPYTER notebook using Python – *‘sklearn’*, ‘*pandas’* and *‘numpy’* as major modules.

# Results

The MSE for the three models is presented in Table 1 below. Fig. 1 shows the top 20 coefficients for variables (all three models) sorted by absolute value of the coefficient. Fig. 2 the predicted vs. actual values for L1 and L2 models. Fig. 3 shows the scatter plot for Critical temperature vs. the first two important variables (‘Barium’ and ‘number of elements’).

Table : MSE Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Description | LR | L1 Model | L2 Model |
|  | 1 | 0.25 | 100 |
| MSE | 369.785 | 376.717 | 373.166 |

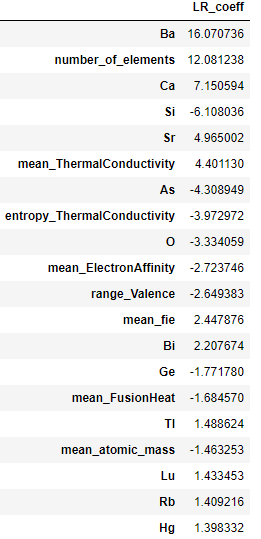
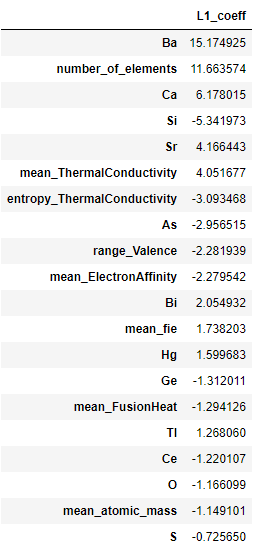
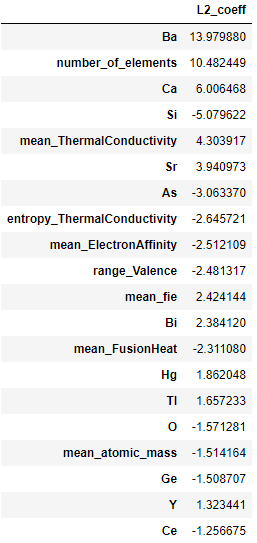
  

Figure : Comparing LR, L1 & L2 model - 20 coefficients[sorted descending by absolute value].

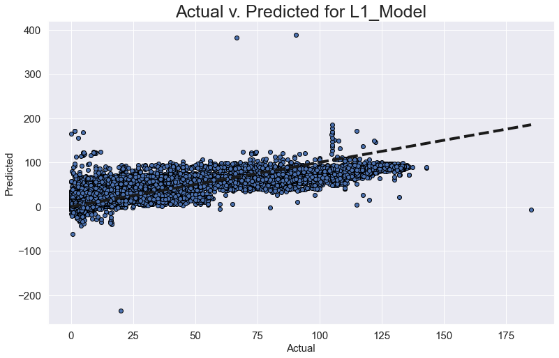
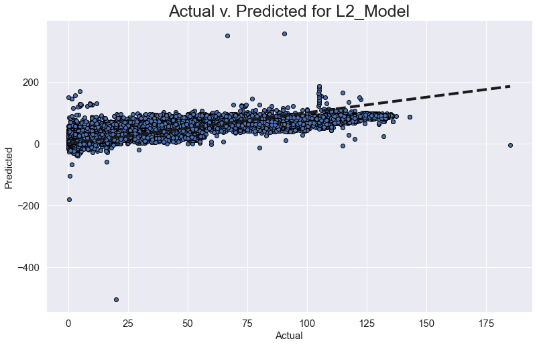
 

Figure 2: Actual v. Predicted plots for L1 and L2 models.

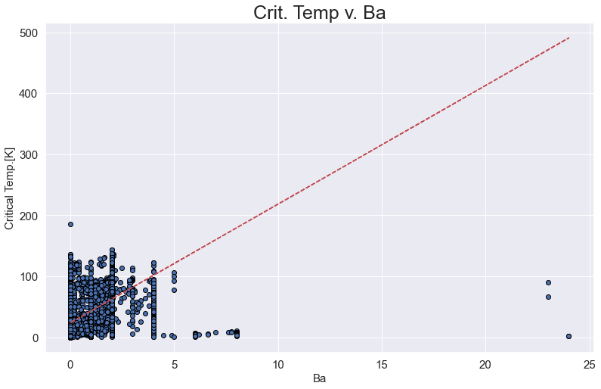
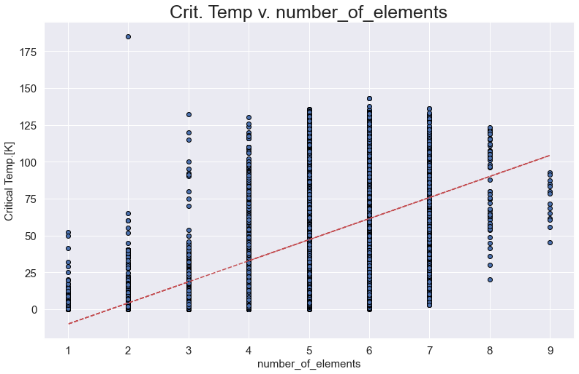
 

Figure 3: Scatter plot for Crit. Temp v. first two variables with highest coefficients.

# Conclusion

Evaluating the results from the variable coefficients (Fig. 1) it can be concluded that the element Barium (Ba) is the most contributing factor to critical temperature of the superconductors followed by Calcium (Ca), Silicon (Si) and Strontium (Sr). The number of elements and mean thermal conductivity of the superconductor alloy also play significant role in the critical temperature determination. Although the number of elements in the alloy combination would be determined by the stability of alloy which would require thorough domain knowledge. The conclusion presented here is only from data science perspective.

# Code

# SuperConductor (Case Study #1: DS7331)

SMU Student collaboration:

1. Allen Miller

2. Vijay Kaniti

3. Venkata 'MG' Vanga

---------------------------------------------

Links:

- https://datacarpentry.org/python-ecology-lesson/05-merging-data/

- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html

- https://scikit-learn.org/stable/modules/linear\_model.html

- https://stackabuse.com/applying-filter-methods-in-python-for-feature-selection/

```python

import pandas as pd

import numpy as np

#reading in the CSV files

unique\_m = pd.read\_csv('https://raw.githubusercontent.com/VenkataVanga/Quantifying-the-World-DS7333-/'\

'main/CS%231\_SuperConductor/unique\_m.csv')

train\_data = pd.read\_csv('https://raw.githubusercontent.com/VenkataVanga/Quantifying-the-World-DS7333-/'\

'main/CS%231\_SuperConductor/train.csv')

```

```python

#print(unique\_m)

```

```python

print(train\_data)

```

```python

#unique\_m['critical\_temp'].unique

```

```python

#train\_data['critical\_temp'].unique

```

```python

Sup\_condt = pd.concat([train\_data, unique\_m], axis=1)

Sup\_condt = Sup\_condt.loc[:, ~Sup\_condt.columns.duplicated()]

Sup\_condt.describe()

```

```python

#Sup\_condt.to\_csv('out.csv', index=False)

Sup\_condt.info(1)

```

```python

# Normalize all the continuous variables. (Material is only the object variable all element names are ordinal)

from sklearn.preprocessing import StandardScaler

sclr = StandardScaler()

# material and critical\_temp columns removed to get scalar tranformation.

Sup\_condt\_norm = Sup\_condt.drop(columns = ['critical\_temp','material'])

Sup\_condt\_norm = pd.DataFrame(sclr.fit\_transform(Sup\_condt\_norm), columns=Sup\_condt\_norm.columns)

Sup\_condt\_norm

```

```python

# creating a response variable.

Y = Sup\_condt['critical\_temp'].copy()

Y = Y.values

```

```python

# Creating Cross Validation.

from sklearn.model\_selection import KFold

cv = KFold(n\_splits=10, shuffle=True, random\_state=5)

print(cv)

```

```python

# Removed element columns with all zero's

X = Sup\_condt\_norm.loc[:,(Sup\_condt\_norm!=0).any(axis=0)]

X

```

```python

cor\_col = ['number\_of\_elements', 'mean\_atomic\_mass', 'wtd\_mean\_atomic\_mass',

'gmean\_atomic\_mass', 'wtd\_gmean\_atomic\_mass', 'entropy\_atomic\_mass',

'wtd\_entropy\_atomic\_mass', 'range\_atomic\_mass', 'wtd\_range\_atomic\_mass',

'std\_atomic\_mass', 'wtd\_std\_atomic\_mass', 'mean\_fie', 'wtd\_mean\_fie',

'gmean\_fie', 'wtd\_gmean\_fie', 'entropy\_fie', 'wtd\_entropy\_fie',

'range\_fie', 'wtd\_range\_fie', 'std\_fie', 'wtd\_std\_fie',

'mean\_atomic\_radius', 'wtd\_mean\_atomic\_radius', 'gmean\_atomic\_radius',

'wtd\_gmean\_atomic\_radius', 'entropy\_atomic\_radius',

'wtd\_entropy\_atomic\_radius', 'range\_atomic\_radius',

'wtd\_range\_atomic\_radius', 'std\_atomic\_radius', 'wtd\_std\_atomic\_radius',

'mean\_Density', 'wtd\_mean\_Density', 'gmean\_Density',

'wtd\_gmean\_Density', 'entropy\_Density', 'wtd\_entropy\_Density',

'range\_Density', 'wtd\_range\_Density', 'std\_Density', 'wtd\_std\_Density',

'mean\_ElectronAffinity', 'wtd\_mean\_ElectronAffinity',

'gmean\_ElectronAffinity', 'wtd\_gmean\_ElectronAffinity',

'entropy\_ElectronAffinity', 'wtd\_entropy\_ElectronAffinity',

'range\_ElectronAffinity', 'wtd\_range\_ElectronAffinity',

'std\_ElectronAffinity', 'wtd\_std\_ElectronAffinity', 'mean\_FusionHeat',

'wtd\_mean\_FusionHeat', 'gmean\_FusionHeat', 'wtd\_gmean\_FusionHeat',

'entropy\_FusionHeat', 'wtd\_entropy\_FusionHeat', 'range\_FusionHeat',

'wtd\_range\_FusionHeat', 'std\_FusionHeat', 'wtd\_std\_FusionHeat',

'mean\_ThermalConductivity', 'wtd\_mean\_ThermalConductivity',

'gmean\_ThermalConductivity', 'wtd\_gmean\_ThermalConductivity',

'entropy\_ThermalConductivity', 'wtd\_entropy\_ThermalConductivity',

'range\_ThermalConductivity', 'wtd\_range\_ThermalConductivity',

'std\_ThermalConductivity', 'wtd\_std\_ThermalConductivity',

'mean\_Valence', 'wtd\_mean\_Valence', 'gmean\_Valence',

'wtd\_gmean\_Valence', 'entropy\_Valence', 'wtd\_entropy\_Valence',

'range\_Valence', 'wtd\_range\_Valence', 'std\_Valence', 'wtd\_std\_Valence']

```

```python

# Checking for correlation

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.simplefilter('ignore', DeprecationWarning)

%matplotlib inline

sns.set()

sns.set(style='darkgrid')

cmap = sns.diverging\_palette(220, 10, as\_cmap=True)

f, ax= plt.subplots(figsize=(18,18))

sns.heatmap(X[cor\_col].corr(),cmap=cmap,fmt='.3f',linewidth=.5,ax=ax)

f.tight\_layout()

```

```python

# Removing all correlated features.

cor\_features = set()

cor\_matrix = X[cor\_col].corr()

for i in range(len(cor\_matrix.columns)):

for j in range(i):

if abs(cor\_matrix.iloc[i, j]) > 0.6:

colname = cor\_matrix.columns[i]

cor\_features.add(colname)

```

```python

# There are 59 correlated features.

len(cor\_features)

```

```python

X\_uncor = X.drop(labels=cor\_features, axis = 1)

X\_uncor

```

### Initial LR

```python

# Baseline linear regression.

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(X\_uncor, Y)

```

```python

mse = round(mean\_squared\_error(Y, lr.predict(X\_uncor)), ndigits = 3)

print('MSE\_Model\_LR=',mse)

```

```python

for i in range(len(X\_uncor.columns)):

print(i," ", X\_uncor.columns[i]," "\*(31-len(X\_uncor.columns[i])),":\t", round(lr.coef\_[i], 5))

```

```python

lr\_important = lr.coef\_

lr\_important = pd.DataFrame(lr\_important, index = X\_uncor.columns, columns = ['LR\_coeff'])

lr\_important = lr\_important.iloc[(-np.abs(lr\_important['LR\_coeff'].values)).argsort()]

lr\_important.head(20)

```

```python

lr.intercept\_

```

### Finding Important Features

```python

#Lasso (L1)

from sklearn.linear\_model import Lasso

from sklearn.model\_selection import cross\_val\_score

l1\_model = Lasso(alpha=1)

l1\_model.fit(X\_uncor,Y)

```

```python

alpha = 1

for i in range(10):

l1\_model.alpha = alpha

print("aplha =", alpha, "CVScore =", cross\_val\_score(l1\_model, X\_uncor, Y, cv=cv, scoring= 'neg\_mean\_squared\_error').mean())

print("----------")

alpha = alpha / 2

```

```python

l1\_model = Lasso(alpha=0.25)

l1\_model.fit(X\_uncor,Y)

for i in range(len(X\_uncor.columns)):

print(i, X\_uncor.columns[i]," "\*(30-len(X\_uncor.columns[i])),":\t", round(l1\_model.coef\_[i], 3))

```

```python

l1\_important = l1\_model.coef\_

l1\_important = pd.DataFrame(l1\_important, index = X\_uncor.columns, columns = ['L1\_coeff'])

l1\_important = l1\_important.iloc[(-np.abs(l1\_important['L1\_coeff'].values)).argsort()]

l1\_important.head(20)

```

```python

l1\_model.intercept\_

```

```python

from sklearn.metrics import mean\_squared\_error

mse = round(mean\_squared\_error(Y, l1\_model.predict(X\_uncor)), ndigits = 3)

print('MSE\_Model\_L1 =',mse)

```

```python

from sklearn.model\_selection import cross\_val\_predict

import matplotlib.pyplot as plt

%matplotlib inline

predicted = cross\_val\_predict(l1\_model, X\_uncor, Y, cv=cv)

fig, ax = plt.subplots(figsize=(13, 8))

ax.scatter(Y, predicted, edgecolors=(0,0,0))

ax.plot([Y.min(), Y.max()], [Y.min(), Y.max()], 'k--', lw=4)

ax.set\_xlabel('Actual')

ax.set\_ylabel('Predicted')

plt.title('Actual v. Predicted for L1\_Model')

plt.rc('axes', titlesize=25)

plt.rc('axes', labelsize=15)

plt.rc('xtick', labelsize=15)

plt.rc('ytick', labelsize=15)

plt.show()

```

```python

from sklearn.linear\_model import Ridge

l2\_model = Ridge(alpha=1)

l2\_model.fit(X\_uncor,Y)

```

```python

l2\_model.score(X\_uncor,Y)

```

```python

alpha = 1E-5

for i in range(10):

l2\_model.alpha = alpha

print("aplha =", alpha, "CVScore =", cross\_val\_score(l2\_model, X\_uncor, Y, cv=cv, scoring= 'neg\_mean\_squared\_error').mean())

print("----------")

alpha = alpha \*10

```

```python

l2\_model.alpha = 1000

l2\_model.fit(X\_uncor,Y)

l2\_model.score(X\_uncor,Y)

```

```python

l2\_important = l2\_model.coef\_

l2\_important = pd.DataFrame(l2\_important, index = X\_uncor.columns, columns = ['L2\_coeff'])

l2\_important = l2\_important.iloc[(-np.abs(l2\_important['L2\_coeff'].values)).argsort()]

l2\_important.head(20)

```

```python

l2\_model.intercept\_

```

```python

mse = round(mean\_squared\_error(Y, l2\_model.predict(X\_uncor)), ndigits = 3)

print('MSE\_Model\_L2=',mse)

```

```python

from sklearn.model\_selection import cross\_val\_predict

import matplotlib.pyplot as plt

%matplotlib inline

predicted = cross\_val\_predict(l2\_model, X\_uncor, Y, cv=cv)

fig, ax = plt.subplots(figsize=(13, 8))

ax.scatter(Y, predicted, edgecolors=(0,0,0))

ax.plot([Y.min(), Y.max()], [Y.min(), Y.max()], 'k--', lw=4)

ax.set\_xlabel('Actual')

ax.set\_ylabel('Predicted')

plt.title('Actual v. Predicted for L2\_Model')

plt.rc('axes', titlesize=25)

plt.rc('axes', labelsize=15)

plt.rc('xtick', labelsize=15)

plt.rc('ytick', labelsize=15)

plt.show()

```

#### Elements that were removed as all rows were zero.

```python

X1 = Sup\_condt\_norm.loc[:,(Sup\_condt\_norm == 0).any(axis=0)]

X1

```

```python

fig, ax = plt.subplots(figsize=(13, 8))

plt.scatter(Sup\_condt['Ba'],Y, edgecolors=(0,0,0))

z = np.polyfit(Sup\_condt['Ba'],Y, 1)

p = np.poly1d(z)

plt.plot(Sup\_condt['Ba'],p(Sup\_condt['Ba']),"r--")

ax.set\_xlabel('Ba')

ax.set\_ylabel('Critical Temp.[K]')

plt.title('Crit. Temp v. Ba')

plt.show()

```

```python

fig, ax = plt.subplots(figsize=(13, 8))

plt.scatter(Sup\_condt['number\_of\_elements'],Y, edgecolors=(0,0,0))

z = np.polyfit(Sup\_condt['number\_of\_elements'],Y, 1)

p = np.poly1d(z)

plt.plot(Sup\_condt['number\_of\_elements'],p(Sup\_condt['number\_of\_elements']),"r--")

#plt.plot([Sup\_condt['number\_of\_elements'].min(), Sup\_condt['number\_of\_elements'].max()], [Y.min(), Y.max()], 'k--', lw=4)

ax.set\_xlabel('number\_of\_elements')

ax.set\_ylabel('Critical Temp.[K]')

plt.title('Crit. Temp v. number\_of\_elements')

plt.show()

```

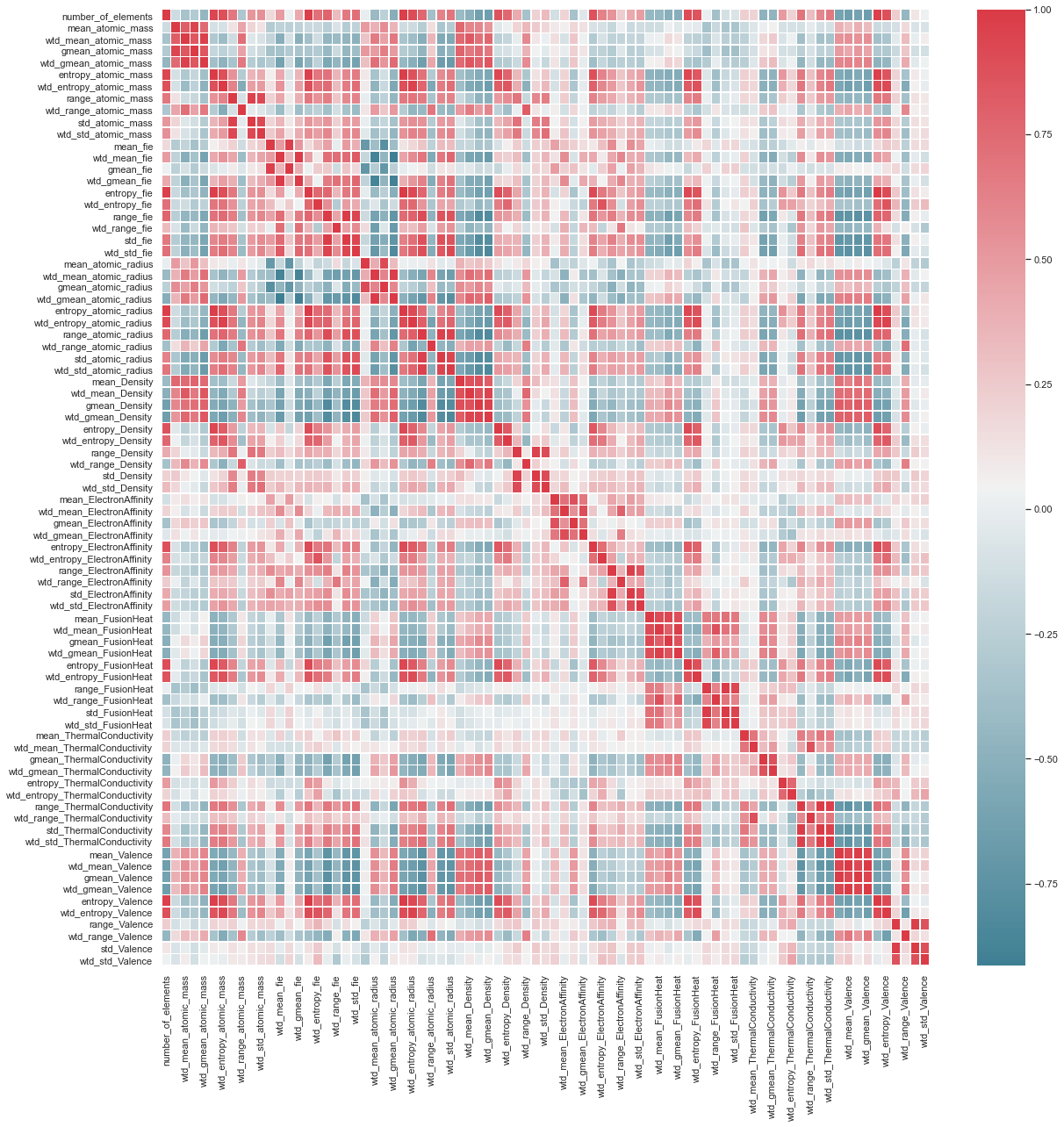


Figure A1: Heatmap for Correlation among the element attributes