A model predicting the Critical Temperature of Super Conducting Materials.

By Rashmi Patel

Business Understanding:

Superconductivity is the ability of some materials that offers zero resistance when electric current is passed through it. This property yields interesting and potentially useful effects., Low temperatures are required for a material to behave as a superconductor. All electrically conductive materials have different resistances with different temperatures that the material is subjected to.
The case study provides us data of different materials which is a mixture of elements from periodic table and the critical temperature at which these materials exhibit super conductive properties. The goal of the case study is to perform modelling on the data provided which can predict the critical temperature for any new material.

Data Understanding:

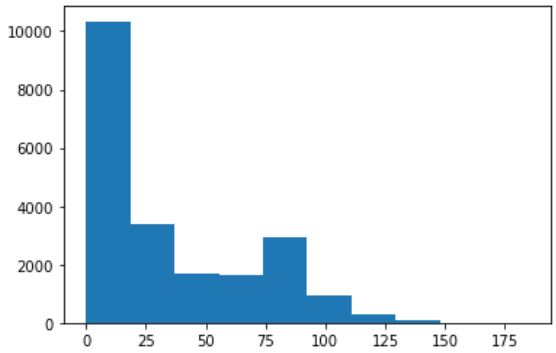
The data provided contains 2 files.

1. train.csv and
2. unique_m.csv

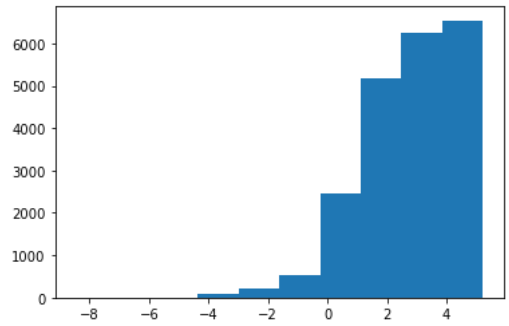
The “train.csv” file consists of 82 features and “unique_m.csv” file consists of 88 features. The data is related one-to-one on based on rows of the csv files provided.

The attribute ‘critical_temp’ is present in both the files, so we have dropped one when combining the 2 csv files. The attribute “materials” describes the material in a string format and does not contribute to model building. After eliminating these 2 columns, the data contains 21,263 rows and 168 columns. The 88 attributes that came from “unique_m.csv” are nothing but the elements in the periodic table. Because one-hot encoding is done on the data, a zero (0) in this attribute indicates that element is not present the mixture being created and one (1) indicates the presence of that element in the mixture. So, we can say that the 88 attributes in unique_m.csv contains categorical data and there is no need to conversion since the data is provided in proper format.

Looking at the plot below we can say that the critical temperature is skewed heaving on the right side. So, to normalize this right skewed distribution, we will apply log transform on the target variable.



Even after the transformation, the data is still skewed but this time towards left. So considering that the dataset is large enough, we can violate the assumption of normality.



Missing and duplicate values:

There is no missing and duplicate data found in the dataset provided.

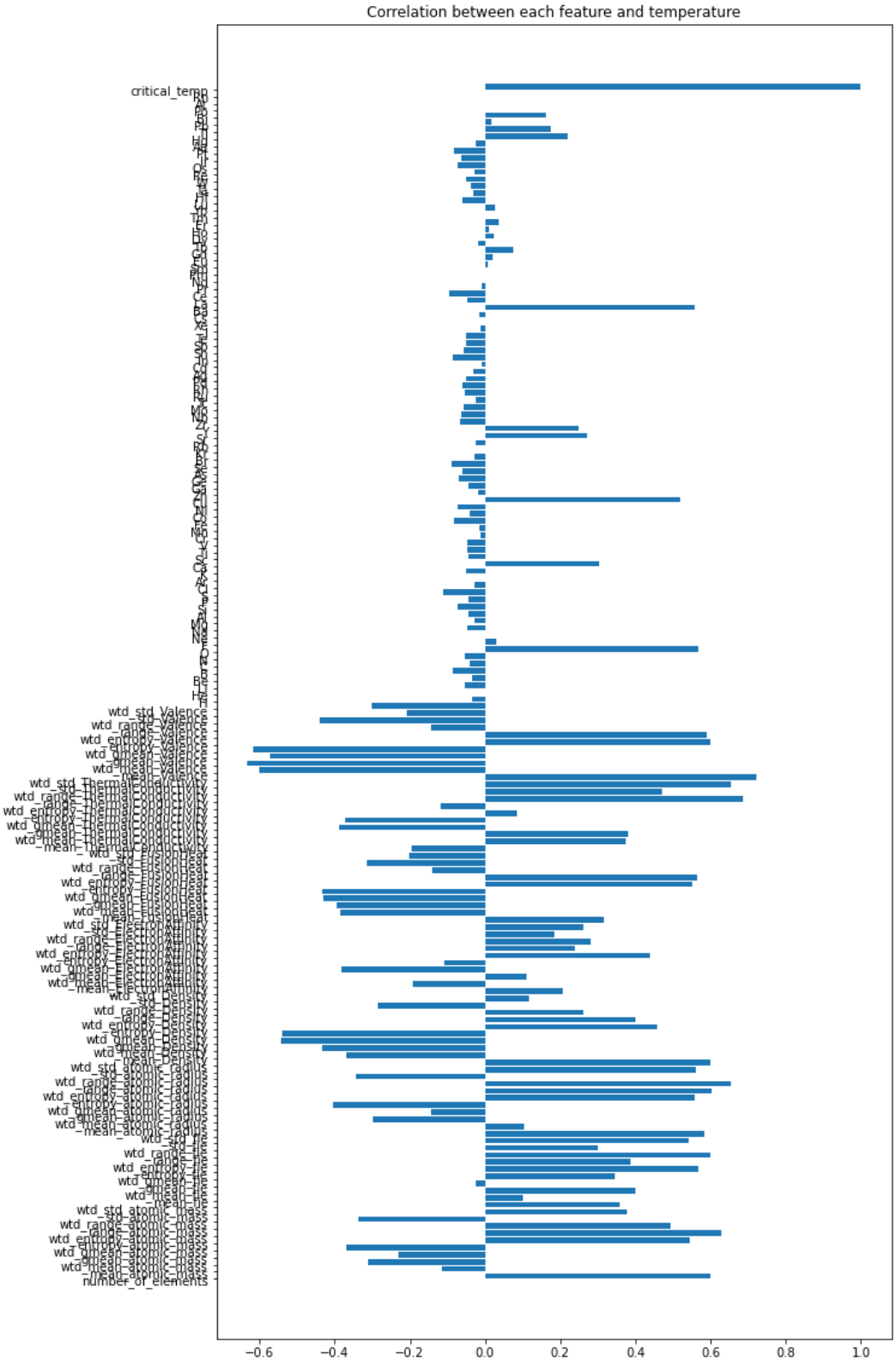
Standardization:

The attributes from train.csv is numeric. Since, each feature varies on scale we need to standardize it through which we can prevent over-emphasizing of other attributes.

Correlation of each attribute with temperature:

We found that some attributes are strongly correlated with critical temperature either positively or negatively. Some of them are: wtd_entropy_atomic_mass, wtd_range_atomic_mass, etc. are positively

correlated and wtd_entropy_valence, wtd_gmean_valence, etc. are negatively corelated. This can be seen in the correlation plot provided below.



Model preparation:

We are going to use L1-LASSO and L2-Ridge methods to perform regression, but prior performing these methods, we will build the regular regression model.

The first step is to split the data into 2 parts, using 75/25 split. One will be training and other is test set. The test will not contain the target variable.

```
# Dividing the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=0)
X_train.shape, X_test.shape
```

Now we, will build a regular regression model which will help in understanding baseline coefficients and performance.

```
# Building and fitting the Linear Regression model
reg_mod = LinearRegression().fit(X_train, y_train)

# Evaluating the Linear Regression model
print(reg_mod.score(X_train, y_train))
print(reg_mod.coef_)
print(reg_mod.intercept_)
print(reg_mod.get_params())
```

Looking at the output, generated by the regular linear regression:

```
0.7683816092272885
[ 7.70288831e-01  8.67373343e-01 -1.36735794e+00 -4.29140856e-01
  8.63761026e-01 -3.02940213e+01  2.15347179e+00  1.59748903e-01
  1.16594287e-01 -4.45808921e-01  6.24269247e-02 -2.29476241e-01
  5.78889874e-02  2.06349104e-01 -1.75156087e-02 -1.70660882e+01
  5.33296285e+01  4.60565032e-02  2.05842406e-02 -6.45141039e-02
 -7.36768174e-02 -7.77139053e-01  2.58768422e+00  5.53794659e-01
 -2.34152440e+00 -1.33985319e+01  1.87133207e+01  6.95744735e-02
 -7.18466697e-02 -2.20875553e-01 -1.04529738e-01 -5.51945479e-03
  5.28681331e-03  8.91298727e-04 -9.56095968e-04 -8.86510884e-01
 -2.08371953e+00 -9.11148109e-04 -6.18719647e-04  3.55693207e-03
 -1.49123640e-03 -5.33209478e-02  3.13496555e-01  1.35096327e-01
 -4.03995250e-01  8.66660300e+00 -1.75223900e+01 -3.41652959e-01
 -5.78284910e-02  1.15585108e+00 -4.72661799e-01  1.01925769e+00
 -1.58772077e+00 -7.12765086e-01  1.14455656e+00 -1.70073697e+01
  2.10986518e+01 -7.47371697e-02  5.95978466e-01 -8.46012687e-01
  4.96765498e-01 -9.61779402e-02  5.15461894e-01 -4.41516723e-02
 -3.34192631e-01  1.36456858e+01  1.24516951e+00 -1.02452808e-01
 -2.13479424e-01  3.32916853e-01  1.38506125e-02  3.77359176e+00
 -6.44140751e+00 -1.96860438e-01  4.12972051e+00  5.37782877e+01
 -6.39190968e+01  4.95955407e+00 -2.38269924e+00 -1.23861489e+00
 -1.44814700e+01  5.86218398e-02 -1.56623159e-10  3.53730888e+00
 -3.98490905e-01 -1.08093785e+00 -4.48965256e-02  3.02876516e-01
 -1.51679872e-01  9.42325675e+00 -3.67167186e-10  5.64226628e+00
  1.41058364e+00 -1.18689213e-01 -1.41468982e+00 -1.32051711e+00
 -1.63875594e+00 -2.77812959e+00  1.69827707e-10  8.23522894e+00
  9.41838234e-01 -2.44819277e-01 -7.02078326e-02  3.82927282e-02
  3.90052480e-01 -3.21509185e-01  1.02085607e+00 -4.54091937e-01
 -3.49454656e-01 -6.46706074e-01 -1.04873654e-01  3.60480523e-01
 -9.87714911e-01 -1.33226789e+00 -9.80105477e-01  8.81575998e-01
 -3.06883408e-11  8.23939271e+00 -2.73092052e-01 -1.17883453e+00
  1.83474112e-02  6.90034125e-02  1.07376222e-01  1.92359243e+00
  1.40144706e-01 -1.97337402e-01 -7.65816738e-02 -1.20122228e+01
 -1.63926141e-01  1.16633725e+00 -7.16475827e-02  2.96024671e-02
  2.98015358e-01  7.93947272e+00  5.05906428e-12  4.90699279e+00
  7.41320078e+00 -9.91962709e-03 -1.63042599e+00 -1.17761063e+01
 -3.49904854e+00 -1.23954180e-11 -2.00949296e+00 -2.66460465e+00
 -1.64684298e+00  2.55395703e-01  2.87020790e+00  1.52052570e+00
  1.66329306e+00  4.13615114e-01  2.07515165e+00  3.38737384e+00
 -3.37622788e-01  4.70699458e-02  6.61754592e-01 -6.08207250e-02
  1.53926775e+00  4.33078893e-02  4.72327341e+00 -1.26766413e+00
  7.25687121e+00  6.55072359e+00  1.21280847e+00  6.16721955e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00]
-8.596236270014316
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': False, 'positive': False}
```

L1-LASSO model

```
# does the same thing above except for lasso
alphas = [0.01, 0.1, 1, 5, 8]
print('different alpha values:', alphas)

lasso_weight = []
for alpha in alphas:
    lasso = Lasso(alpha = alpha, fit_intercept = True)
    lasso.fit(X_train_std, y_train)
    lasso_weight.append(lasso.coef_)

lasso_fig = weight_versus_alpha_plot(lasso_weight, alphas, features)
```

```
# n_alphas: number of alpha values (automatically chosen) to try
# select the best alpha with LassoCV
lasso_cv = LassoCV(n_alphas = 10, fit_intercept = True)
lasso_cv.fit(X_train_std, y_train)


# examine the coefficients and the errors of the predictions
# using the best alpha value
y_pred = lasso_cv.predict(X_test_std)
print('coefficients:\n', lasso_cv.coef_)
print('best alpha:\n', lasso_cv.alpha_)
print('\nRSS:', np.sum(( y_test - y_pred ) ** 2))
```

```
coefficients:
 [ 8.19850490e-01  3.29058574e+00 -4.87094696e+00  0.00000000e+00
-0.00000000e+00 -2.00482909e+00  4.19783383e+00  8.57308828e+00
-0.00000000e+00  0.00000000e+00 -6.82342345e+00  0.00000000e+00
 0.00000000e+00  7.27614345e-01  1.11605477e+00  0.00000000e+00
 3.21666915e-01  2.05665339e+00  0.00000000e+00 -3.45353201e+00
-0.00000000e+00 -0.00000000e+00  1.96790338e+00 -1.73719532e+00
-0.00000000e+00  0.00000000e+00  0.00000000e+00  2.07041146e+00
-1.24776147e+00 -3.05288110e+00  5.96184283e+00 -2.77271767e+00
 3.87973630e+00 -2.18481293e+00  0.00000000e+00 -2.70830851e+00
-1.17722928e+00 -1.28641930e+00  5.88330980e-01  0.00000000e+00
-6.53918143e-01  8.53263962e-01  0.00000000e+00  1.14681709e+00
-3.86213952e+00  0.00000000e+00 -3.46230278e+00 -7.75311013e+00
-7.54058492e-01  1.04214838e+01 -3.05398740e+00 -0.00000000e+00
-0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 3.71710431e+00 -4.09222840e-01  1.81066059e+00 -0.00000000e+00
-2.39302650e+00 -0.00000000e+00  1.66760972e+01 -3.09843686e+00
-9.95273626e+00  1.76382497e+00  2.57948092e+00 -0.00000000e+00
-5.10595777e+00 -0.00000000e+00  4.30604093e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  7.64426206e-02  0.00000000e+00
-0.00000000e+00  2.20201958e+00  1.83202691e+00 -0.00000000e+00
-5.61524746e+00 -5.34435662e-02  0.00000000e+00  4.93768520e-01
-3.23614291e-01 -7.22415631e-01 -0.00000000e+00 -3.14935488e-01
-0.00000000e+00  6.34905073e-01  0.00000000e+00  5.62490126e-02
 5.28999552e-01 -3.68176539e-01 -3.70067010e+00 -4.13415617e-01
-1.32184595e+00 -1.20864919e+00  0.00000000e+00  2.97506749e-01
 1.39739254e+00  4.61138229e-02 -2.72535111e-01  3.00580254e-01
 0.00000000e+00 -0.00000000e+00  4.83240864e-01 -3.60380054e-01
-3.12174326e-01 -1.18525995e+00 -1.07516784e-01  1.37064959e-01
-1.10262032e+00 -1.47575002e+00 -6.90265703e-01 -1.13536746e-01
 0.00000000e+00  3.59609112e-01 -0.00000000e+00 -8.00106257e-02
 2.80319889e-02  4.29212710e-01 -1.53857422e-02  1.76665133e-01
 0.00000000e+00 -6.61023670e-02 -1.98663856e-01 -1.58941266e+00
-6.29106995e-02  2.58108674e-01  5.09163636e-02 -0.00000000e+00
 0.00000000e+00  2.37782200e-01  0.00000000e+00  7.33613183e-02
 8.45703433e+00 -2.44667144e-02 -4.61781889e-01 -1.72051671e+00
-8.09518894e-01  0.00000000e+00 -4.08632548e-02 -5.47817369e-01
-3.78522126e-01  0.00000000e+00  1.38261026e-01  1.45477275e-02
 8.34522696e-02 -0.00000000e+00  3.14553170e-01  6.00957605e-01
-6.33171489e-02  0.00000000e+00 -0.00000000e+00 -3.58182542e-02
 5.26178165e-01  0.00000000e+00  8.38663826e-01 -2.73947129e-01
 1.80506014e+00  1.29310240e+00  1.95113133e-01  4.13606409e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00]
best alpha:
 0.053353446993498053

RSS: 4372455.525331622
```

We found that the best alpha selected by LASSO method is 0.533 and RSS (Residuals sum of squares) is 4372455.525331622

L2-Ridge Model:

```
# alpha: array of alpha values to try; must be positive, increase for more regularization
# create an array of alpha values and select the best one with RidgeCV
alpha_range = 10. ** np.arange(-2, 3)
ridge_cv = RidgeCV(alphas = alpha_range, fit_intercept = True)
ridge_cv.fit(X_train_std, y_train)

# examine the coefficients and the errors of the predictions
# using the best alpha value
y_pred = ridge_cv.predict(X_test_std)


print('coefficients:\n', ridge_cv.coef_)
print('best alpha:\n' , ridge_cv.alpha_)
print('\nRSS:', np.sum((y_test - y_pred) ** 2))
```

```
coefficients:
 [ 1.19326769e+00  4.28518037e+00 -7.04259864e+00  2.70909686e+00
 -1.46259678e+00 -5.97008428e+00  3.57968283e+00  6.69167150e+00
  5.00832634e-01  6.37851805e-01 -6.86624827e+00 -1.43017458e+00
  1.17262979e+00  1.51437575e+00  1.49615677e+00  1.92606685e+00
  3.89279140e+00  4.82540465e+00  1.43236360e+00 -5.15344268e+00
 -1.95863056e+00 -2.62951574e-01  5.90825312e+00 -2.59942152e+00
 -1.98342555e+00 -7.95780161e-02  1.25137334e+00  4.46387959e+00
 -2.09338029e+00 -6.08228792e+00  7.19205964e+00 -4.07318303e+00
  4.72192544e+00 -3.96748852e+00  1.55668594e+00 -2.41354201e+00
 -2.20470300e+00 -2.69386956e+00  3.73863033e-01  1.98496085e+00
 -1.45939844e+00  9.77446615e-01  3.44003234e+00  1.20373183e+00
 -6.60761063e+00  5.34202732e-01 -4.18080086e+00 -8.45919727e+00
 -2.03866189e+00  1.10708971e+01 -3.97181321e+00  7.37658148e-01
 -1.64646644e+00  2.55642616e-01 -3.07557683e-01 -1.62436264e+00
  4.96198297e+00 -1.56277638e+00  3.30594009e+00  1.26591488e-01
 -2.15330791e+00  2.67364113e-01  1.36720347e+01 -3.79658181e+00
 -7.11644729e+00  2.12906699e+00  2.50589920e+00 -3.26850668e+00
 -4.75472334e+00  2.00040010e+00  8.01639882e+00 -5.03575663e-01
 -1.45555350e-01  1.17509281e+00  5.66487923e-01  2.54908625e+00
 -3.24726070e+00  3.81964207e+00  1.44416827e+00 -9.23942182e-01
 -6.06702303e+00 -3.33152828e-01  0.00000000e+00  4.61656669e-01
 -3.77556238e-01 -7.46041192e-01 -1.08130982e-01 -3.81937356e-01
 -2.69775826e-01  6.08028114e-01  0.00000000e+00  2.12852763e-01
  4.43540960e-01 -3.83412445e-01 -3.70497808e+00 -5.39936907e-01
 -1.28776505e+00 -1.35699165e+00  0.00000000e+00  5.07405026e-01
  1.46075157e+00  1.13623780e-01 -3.09713689e-01  2.38955867e-01
  4.03029082e-02 -7.93872696e-02  7.84079281e-01 -4.25369114e-01
 -3.65684642e-01 -1.34897391e+00 -1.34124440e-01  3.35199352e-01
 -1.13953932e+00 -1.94715723e+00 -7.01566988e-01 -1.92039240e-01
  0.00000000e+00  5.28900255e-01 -1.24673129e-01 -8.81263374e-02
  1.76517722e-01  3.82130068e-01 -2.31316427e-02  1.66472796e-01
  9.69899188e-02 -1.89980265e-01 -2.28852982e-01 -1.66043529e+00
 -1.15349795e-01  5.34275523e-01  9.63128796e-02 -5.14970607e-03
  1.36433462e-01  2.78145926e-01  0.00000000e+00  1.60943103e-01
  8.33189022e+00 -3.89631742e-02 -4.83529210e-01 -1.76845949e+00
 -8.66692013e-01  0.00000000e+00 -1.22588329e-01 -5.34155767e-01
 -3.58130282e-01  7.74704695e-02  2.38746595e-01  1.24791313e-01
  1.77206051e-01  1.01646180e-02  3.89634736e-01  8.06014473e-01
 -1.23034938e-01  8.92771597e-02 -4.37142494e-02 -1.37703433e-01
  6.11274318e-01 -7.78518596e-03  1.17554542e+00 -4.08702747e-01
  1.59319362e+00  1.39494338e+00  2.49606788e-01  3.95426119e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00]
best alpha:
 100.0

RSS: 4513122.382153014
```

We found that the best alpha selected by Ridge method is 100.0 and RSS (Residual sum of squares) is 4513122.382153014.

Comparing the accuracy score for all models:

```
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import GridSearchCV

from sklearn.pipeline import make_pipeline
lasso_pipe_svc = make_pipeline(RobustScaler(), Lasso(random_state=1))
ridge_pipe_svc = make_pipeline(RobustScaler(), Ridge(random_state=1))

param_range = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 10, 100, 1000, 10000]
param_l1_ratio = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

param_grid_lasso = [{'lasso__alpha': param_range}]
param_grid_ridge = [{'ridge__alpha': param_range}]

gs_lasso = GridSearchCV(estimator=lasso_pipe_svc, param_grid=param_grid_lasso, scoring='r2', cv=5, n_jobs=-1)
gs_lasso.fit(X_train, y_train)

gs_ridge = GridSearchCV(estimator=ridge_pipe_svc, param_grid=param_grid_ridge, scoring='r2', cv=5, n_jobs=-1)
gs_ridge.fit(X_train, y_train)
```

```
print("Lasso")
print(gs_lasso.best_score_)
print(gs_lasso.best_params_)
print("")

print("Ridge")
print(gs_ridge.best_score_)
print(gs_ridge.best_params_)
print("")

print("Regular")
print(reg.score(X_train, y_train))
```

```
Lasso
0.7379441963982758
{'lasso__alpha': 0.1}

Ridge
0.7381836497384814
{'ridge__alpha': 100}

Regular
0.7683816092272885
```

We found that each model performs almost the same on the training data. Now we will look at the prediction obtained using L1-LASSO and L2-Ridge model techniques:

```
from sklearn import metrics

# Note the X_test gets run through the pipeline above! Very important, it means that the scaler is also run on the test
y_lasso_pred = gs_lasso.predict(X_test)
y_ridge_pred = gs_ridge.predict(X_test)
y_pred = reg.predict(X_train)

print("Lasso")
print("R2 ->", metrics.r2_score(y_test, y_lasso_pred))
print("MAE ->", metrics.mean_absolute_error(y_test, y_lasso_pred))

print("Ridge")
print("R2 ->", metrics.r2_score(y_test, y_ridge_pred))
print("MAE ->", metrics.mean_absolute_error(y_test, y_ridge_pred))
print("")
```

```
Lasso
R2 -> 0.5670044899346585
MAE -> 13.337904510832372
Ridge
R2 -> 0.4425319380341406
MAE -> 12.995829837668456
```

It seems that Ridge is doing better although the R2 is less than LASSO because lower the MAE better is the prediction.

Conclusions:

In short, we recommend using Ridge regression when attempting to predict critical temperatures of superconducting materials.

Based on the GridsearchCV technique used for model comparison, we can conclude that Ridge regression produces better prediction than LASSO because lower is the MAE (mean absolute error) better is the prediction done by model.

Based on RSS (Residual sum of squares) we can conclude the same that, ridge method is doing better than LASSO in predicting the critical temperature of superconductor materials because lower the RSS, better is the model prediction.

For comparing more precisely, we can also work on looking at the MSE (Mean Squared Error), RMSE (Root Mean Squared error), R2 (R-squared) etc. factors.