Modelling Critical Temperature of Super Conducting Materials.

By Ben Goodwin, Pradeep Kumar, Sid Swarupananda, Liju Mathews

Business Understanding:

All electrically conductive materials have varying resistances with varying temperatures that the material is subjected to. Super conductivity is a state of these conductors when it offers least resistance to the flow of electricity. The case study at hand offers us data of different materials which have mix of elements from periodic table and the critical temperature at which these materials exhibit super conductive properties. The goal here is to build a model which can predict the critical temperature for any new material when all its features are provided.

Data Evaluation/Engineering:

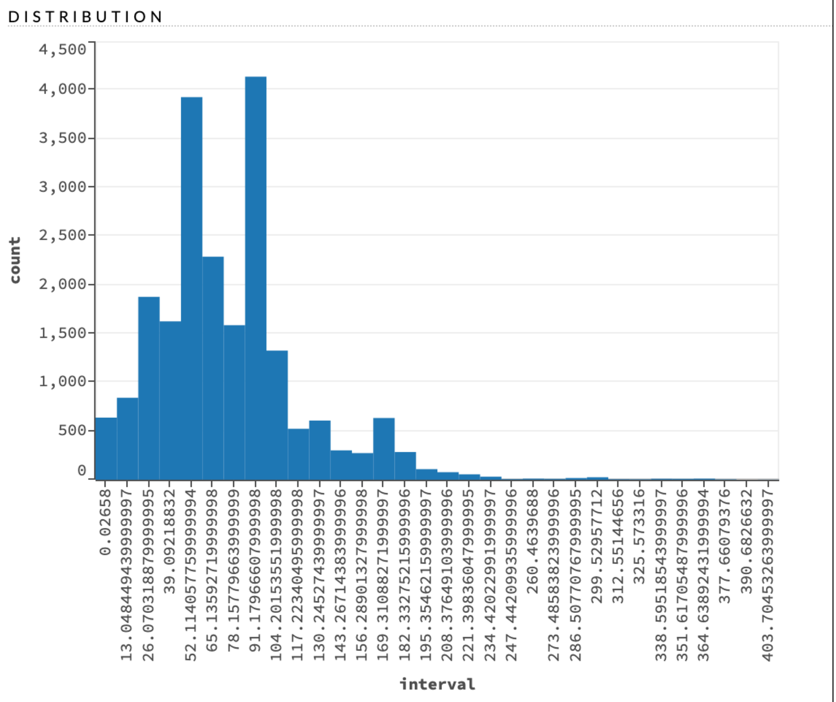
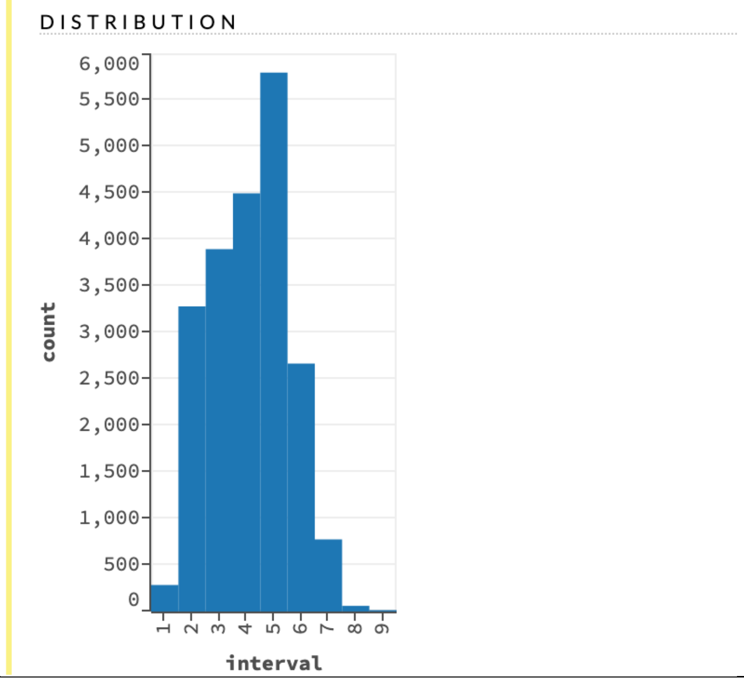
The data is derived from 2 files, "train.csv" and "unique_m.csv." "Train.csv" consists of 82 features and "unique_m.csv" consists of 88 features. The rows from these 2 files have a 1-to-1 relation, as a result they were concatenated in the column axis.

The feature 'critical_temp' is repeated in both the files, so one was eliminated.

The feature "materials" describes the material in a string format, and is not useful for model building. After eliminating these 2 columns, the data contains 21,263 rows and 168 columns. The 88 features that came from "unique_m.csv" represents elements in the periodic table. A zero in this feature represents absence of this element and one or other number represents presence of the element. So, these 88 features can be considered as categorical data. And there is no categorical conversion need since they are already in required format.

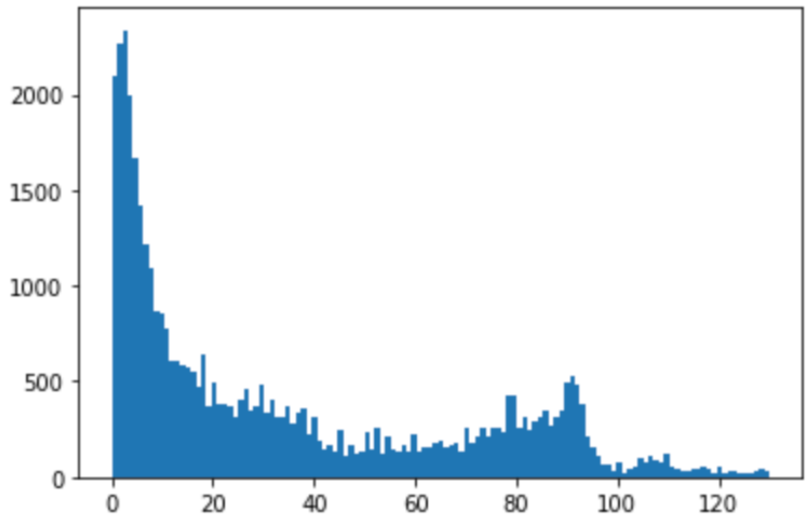
Some of these independent variables have normal distribution and some do not.

Example:

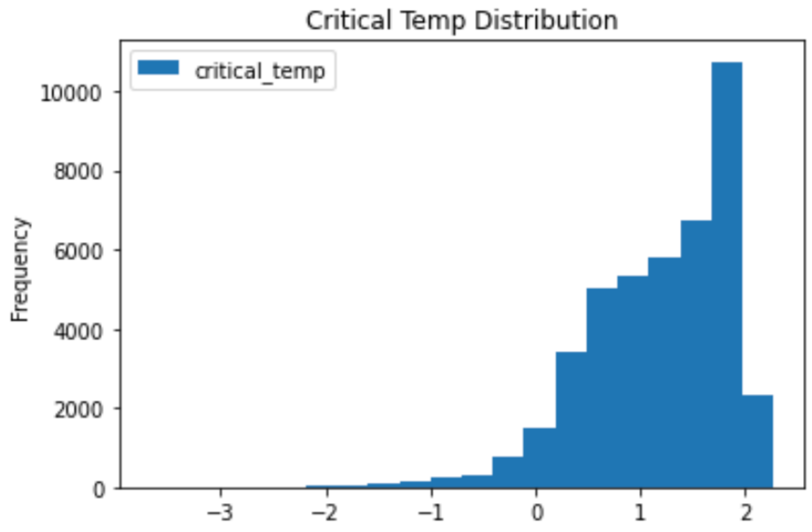
Feature	Distribution
wtd_mean_thermalconductivity	
Number_of_elements	

After inspecting the distributions of all the numerical features, we will assume that they are suitable for modeling with linear regression algorithms.

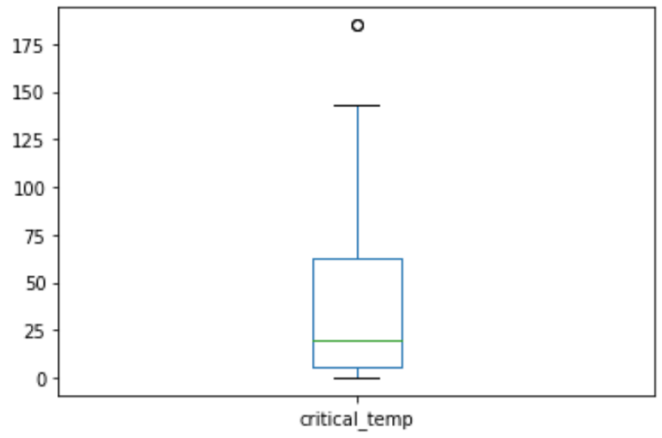
Some exploratory data analyses:



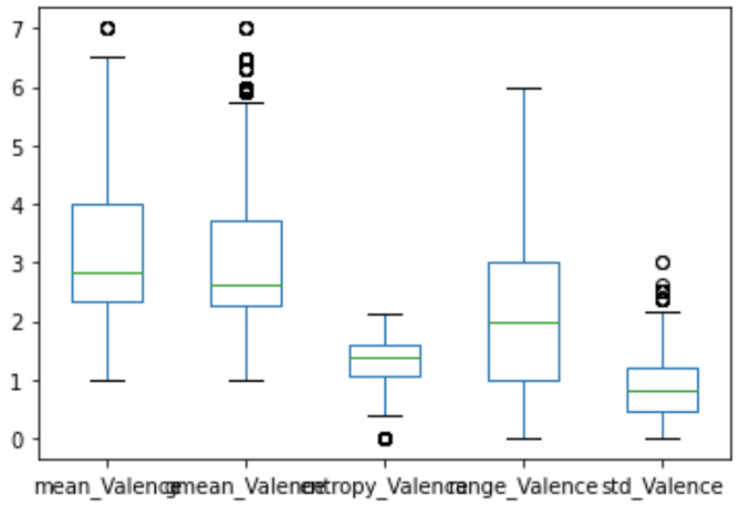
The purpose of the plot above was to investigate the distribution of values of our outcome variable. Based on this histogram we can see that the data is heavily right skewed. This may be worth log-transforming to fit our normality assumption.

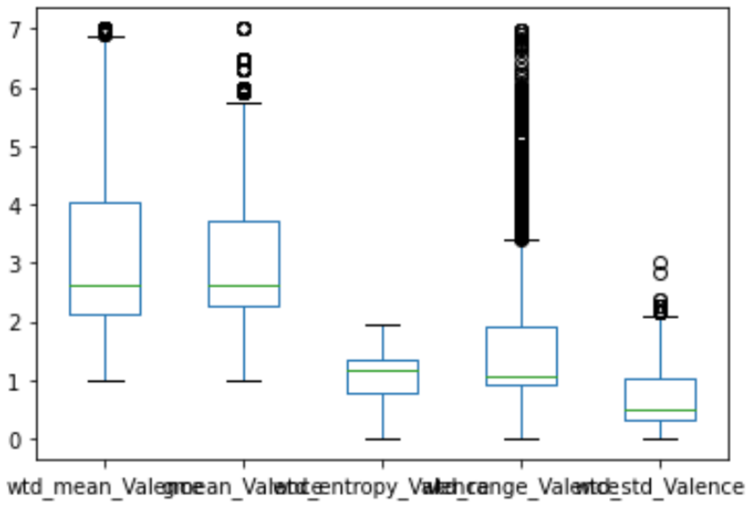


Above is the log transformed version of critical temp- the distribution is ever more so slightly normal. Due to the large sample size, this assumption may be violated.



Again, another perspective on the outcome variable. This is a simple box plot of the outcome variable and shows us that most of the values are <75.



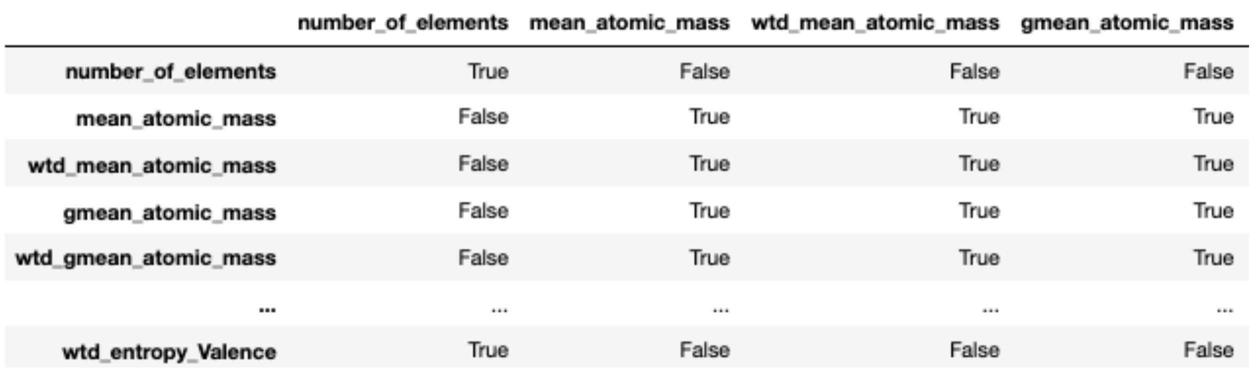


In the plots above we observe the differences between the weighted valences and the non-weighted valences. From these plots we can see that there is not much of a difference in the shapes of the distributions between the weighted and non-weighted. Additionally we can see that certain variables are like entropy valance are quite “tight” distributions.

Checking for multicollinearity in data:

There are independent features which have high correlation with other features which violates the independence assumption for linear regression. To eliminate multicollinearity, correlation matrix was generated.
Calculating the pair wise correlation between numeric features and listing out the features which have correlation greater than 0.8, we can get 144 pairs which have correlations equal to greater than 0.8.
To fix the multicollinearity, some of the highly correlated features were removed from modeling. We are assuming that removing the highly correlated features will not have any unintended consequences for our audience. We are also assuming that predicting the critical temperature is the number one priority for our audience and that we as data scientists can freely make assumptions about variables and fit a model as we see fit.

Sample correlation matrix with correlation > 0.8



	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass
number_of_elements	True	False	False	False
mean_atomic_mass	False	True	True	True
wtd_mean_atomic_mass	False	True	True	True
gmean_atomic_mass	False	True	True	True
wtd_gmean_atomic_mass	False	True	True	True
...	...	...	...	...
wtd_entropy_Valence	True	False	False	False

Missing values:

There are no missing values in any of these features. So, no imputation exercise is required.

Duplicate data:

Duplicate data was identified and handled appropriately.

Standardizing data:

The 82 features from train.csv consist of numeric data.
The scale of each feature varies so standardizing the data is required, as to prevent the over-emphasize of some variables relative to others. The standardization is done by subtracting each value by mean and dividing by standard deviation, it is important this standardization is done after splitting the data into train and test split to avoid data snooping.

Modeling Preparations:

The dependent variable to predict is 'critical_temp'. This dependent variable is a real number, and it ranges from 0 to 185 and mean at 34.4. A tree-based model or a linear regression model is suitable for this problem. For this case study, the linear model will be chosen.

The following evaluation metrics will be used for the regression task. -Mean Absolute Error (MAE) -Root Mean Squared Error (RMSE) -Mean Absolute Percentage Error (MAPE) and R^2

Mean absolute error is being used because it is an intuitive metric that simply looks at the absolute difference between the data and the models' predictions. This metric is a good first pass at evaluating a model's performance, its simple and quick. The mean absolute error, however, does not indicate under/over performance in the model. The residuals all contribute proportionally, so larger errors will contribute significantly to the model. A small MAE is a good indicator of good prediction, and a large MAE indicates the model may need more work. MAE of 0 is ideal, but not possible. The MAE is robust to outliers since it uses the absolute value. For this model a lower MAE is "better." This metric is appropriate since we are concerned with the model's ability to predict critical temperatures.

Root mean squared error is the standard deviation of the residuals (commonly known as prediction errors). We use these residuals to see how far from the line the points are. RMSE is a good metric to tell us how concentrated the data is around the line of best fit. The nice and differentiating part of RMSE is that they can be positive or negative as the predicted value under or over the estimates of the actual value unlike the MAE and MAPE which use the absolute value. Additionally, we can use the RMSE as a good measure of the spread of the y values about the predicted y values.

Mean absolute percentage error is being used as the percentage equivalent of MAE. Similarly, to the MAE, MAPE measures how far our model's predictions are off from their corresponding outputs on average. MAPE is easy to interpret because of its use of percentages. As a downside, MAPE is undefined for data where we have 0 values. The MAPE can also grow very large if the values themselves are small, and as a final note MAPE is biased towards predictions that are systematically less than actual values. We will use MAPE as a more interpretable MAE. We will interpret the MAPE in terms of percentages, as an example the MAPE will state, "our model's predictions are on average xx% off from the actual value." This metric is appropriate since we are concerned with the model's ability to predict critical temperatures, furthermore the addition of percentage will further our ability to interpret the model's performance.

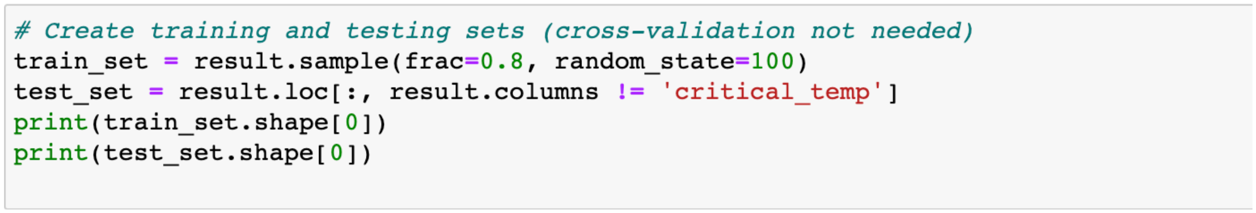
Finally, R2 or the coefficient of determination will be used and is the proportion of the variation in the dependent variable that is predictable from the independent variable. This isn't the best metric, but it is quite easy to interpret as it lives in a 0-100% scale.

Model Building and Evaluation:

Baseline Modeling:

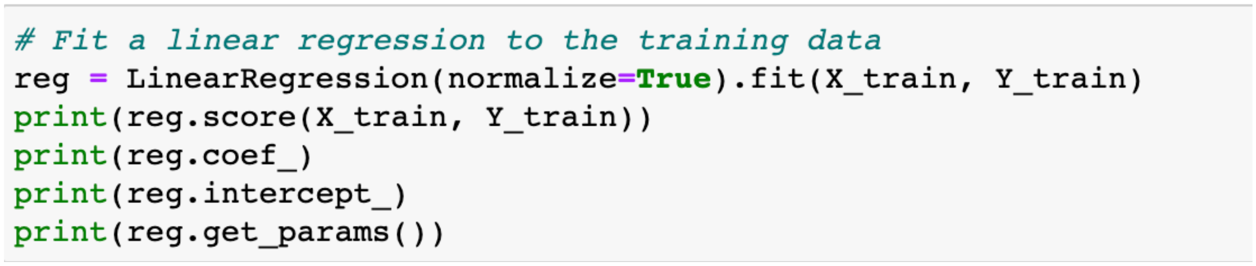
Prior to running L1 and L2 regularization, we ran a baseline Linear Regression Model to assess performance and to use as a baseline.

We first began by splitting the data into training and testing datasets, using a 80% training/testing split. Note the outcome variable has been removed from the testing set.



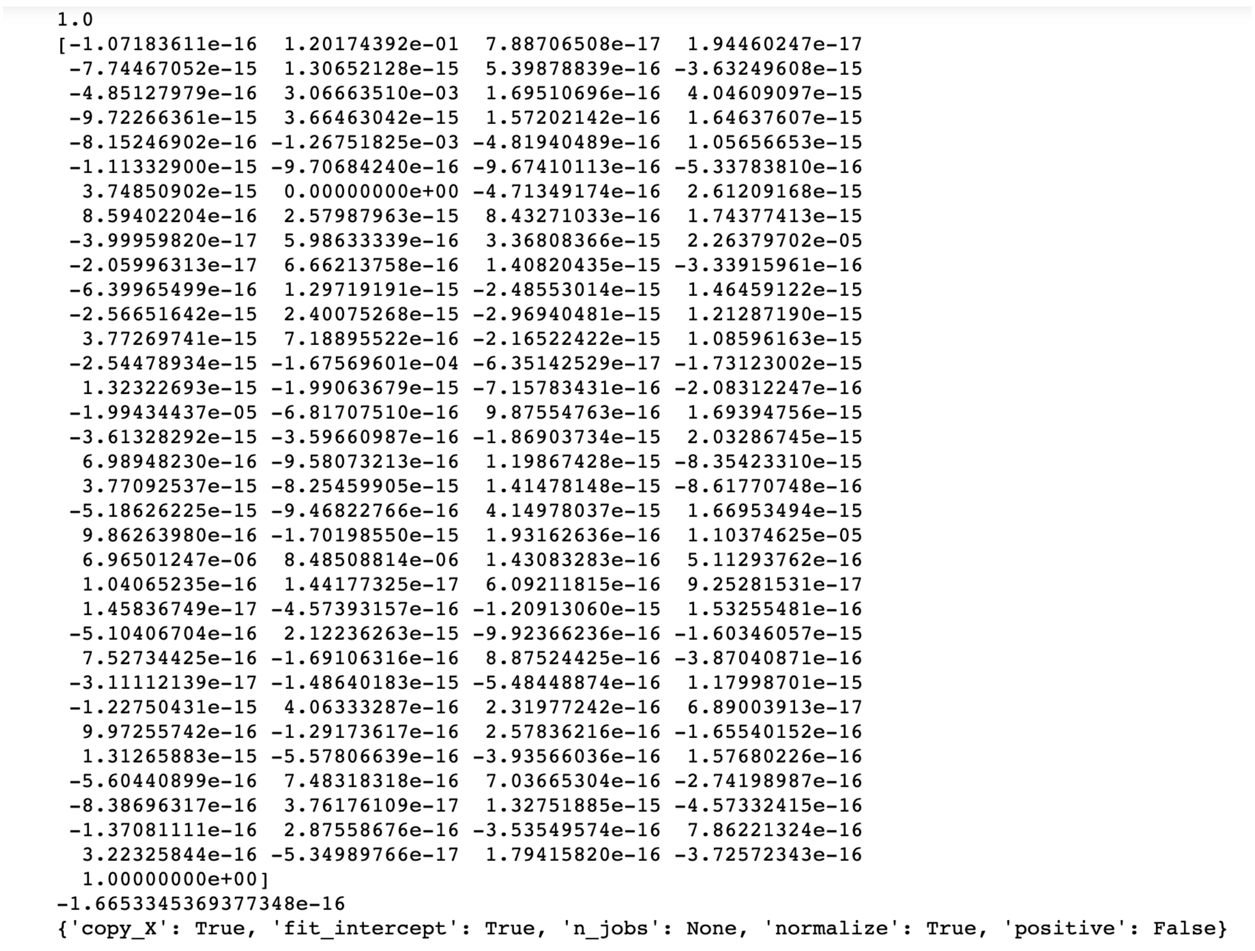
```
# Create training and testing sets (cross-validation not needed)
train_set = result.sample(frac=0.8, random_state=100)
test_set = result.loc[:, result.columns != 'critical_temp']
print(train_set.shape[0])
print(test_set.shape[0])
```

We then fit a linear regression model to the data to assess baseline model performance



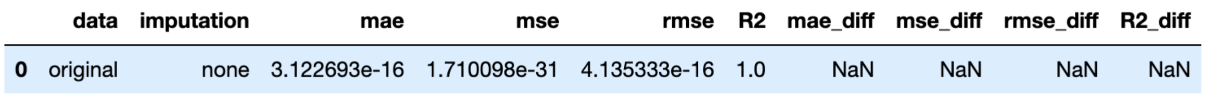
```
# Fit a linear regression to the training data
reg = LinearRegression(normalize=True).fit(X_train, Y_train)
print(reg.score(X_train, Y_train))
print(reg.coef_)
print(reg.intercept_)
print(reg.get_params())
```

The baseline linear regression model produced the following coefficients (parameters are also listed below)



```
1.0
[-1.07183611e-16  1.20174392e-01  7.88706508e-17  1.94460247e-17
 -7.74467052e-15  1.30652128e-15  5.39878839e-16 -3.63249608e-15
 -4.85127979e-16  3.06663510e-03  1.69510696e-16  4.04609097e-15
 -9.72266361e-15  3.66463042e-15  1.57202142e-16  1.64637607e-15
 -8.15246902e-16 -1.26751825e-03 -4.81940489e-16  1.05656653e-15
 -1.11332900e-15 -9.70684240e-16 -9.67410113e-16 -5.33783810e-16
  3.74850902e-15  0.00000000e+00 -4.71349174e-16  2.61209168e-15
  8.59402204e-16  2.57987963e-15  8.43271033e-16  1.74377413e-15
 -3.99959820e-17  5.98633339e-16  3.36808366e-15  2.26379702e-05
 -2.05996313e-17  6.66213758e-16  1.40820435e-15 -3.33915961e-16
 -6.39965499e-16  1.29719191e-15 -2.48553014e-15  1.46459122e-15
 -2.56651642e-15  2.40075268e-15 -2.96940481e-15  1.21287190e-15
  3.77269741e-15  7.18895522e-16 -2.16522422e-15  1.08596163e-15
 -2.54478934e-15 -1.67569601e-04 -6.35142529e-17 -1.73123002e-15
  1.32322693e-15 -1.99063679e-15 -7.15783431e-16 -2.08312247e-16
 -1.99434437e-05 -6.81707510e-16  9.87554763e-16  1.69394756e-15
 -3.61328292e-15 -3.59660987e-16 -1.86903734e-15  2.03286745e-15
  6.98948230e-16 -9.58073213e-16  1.19867428e-15 -8.35423310e-15
  3.77092537e-15 -8.25459905e-15  1.41478148e-15 -8.61770748e-16
 -5.18626225e-15 -9.46822766e-16  4.14978037e-15  1.66953494e-15
  9.86263980e-16 -1.70198550e-15  1.93162636e-16  1.10374625e-05
  6.96501247e-06  8.48508814e-06  1.43083283e-16  5.11293762e-16
  1.04065235e-16  1.44177325e-17  6.09211815e-16  9.25281531e-17
  1.45836749e-17 -4.57393157e-16 -1.20913060e-15  1.53255481e-16
 -5.10406704e-16  2.12236263e-15 -9.92366236e-16 -1.60346057e-15
  7.52734425e-16 -1.69106316e-16  8.87524425e-16 -3.87040871e-16
 -3.11112139e-17 -1.48640183e-15 -5.48448874e-16  1.17998701e-15
 -1.22750431e-15  4.06333287e-16  2.31977242e-16  6.89003913e-17
  9.97255742e-16 -1.29173617e-16  2.57836216e-16 -1.65540152e-16
  1.31265883e-15 -5.57806639e-16 -3.93566036e-16  1.57680226e-16
 -5.60440899e-16  7.48318318e-16  7.03665304e-16 -2.74198987e-16
 -8.38696317e-16  3.76176109e-17  1.32751885e-15 -4.57332415e-16
 -1.37081111e-16  2.87558676e-16 -3.53549574e-16  7.86221324e-16
  3.22325844e-16 -5.34989766e-17  1.79415820e-16 -3.72572343e-16
  1.00000000e+00]
-1.6653345369377348e-16
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': True, 'positive': False}
```

Parameters: Fit Intercept: True, n_jobs: None, Normalize: True, positive: False
The baseline linear regression model produced the following metrics below:



	data	imputation	mae	mse	rmse	R2	mae_diff	mse_diff	rmse_diff	R2_diff
0	original	none	3.122693e-16	1.710098e-31	4.135333e-16	1.0	NaN	NaN	NaN	NaN

From the above results we can see that the baseline linear regression model scores decently well. MSE and RMSE are low, but we cannot adequately compare until we run our regularization algorithms.

L1 LASSO Modeling:

This modeling approach utilizes LASSO regression and utilizes the same training and testing datasets, using a 80% training/testing split. Note the outcome variable has been removed from the testing set.

LASSO Model setup:

```
l1_mod = linear_model.Lasso(alpha=0.001, normalize=True).fit(X_train, Y_train)
print(l1_mod.score(X_train, Y_train))
print(l1_mod.coef_)
print(l1_mod.intercept_)
print(l1_mod.get_params())
```

The LASSO model produced the following coefficients (parameters are also listed below)

```
0.0
[-0.  0. -0. -0. -0. -0. -0. -0. -0.  0. -0. -0. -0. -0. -0. -0. -0.  0.
 -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.  0.
 -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.  0.
 -0. -0. -0. -0. -0. -0.  0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.
 -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.  0.  0.  0. -0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
0.11261187177580305
{'alpha': 0.001, 'copy_X': True, 'fit_intercept': True, 'max_iter': 1000, 'normalize': True, 'positive': False, 'prec
ompute': False, 'random_state': None, 'selection': 'cyclic', 'tol': 0.0001, 'warm_start': False}
```

(The coefficients are quite small and are not immediately visible in this view)

Parameters: Alpha: 0.001, copy X:True, fit intercept: True, max iterations: 1000, normalize: true. Positive, false, precompute: false, random state: none, selection: cyclic; tol:0.0001, warm_start: False

The LASSO model produced the following metrics below (and is compared to baseline):

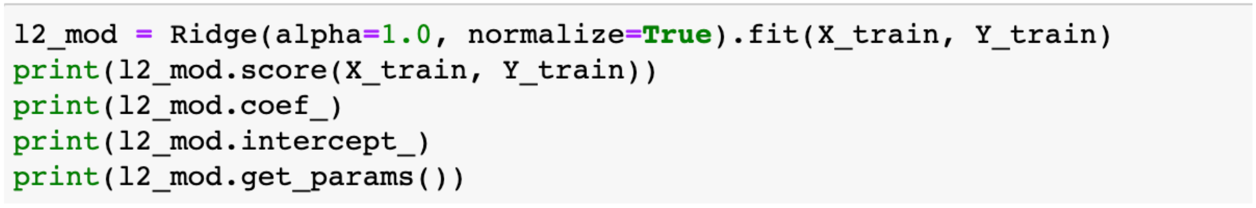
	data	imputation	mae	mse	rmse	R2	mae_diff	mse_diff	rmse_diff	R2_diff
0	original	none	3.122693e-16	1.710098e-31	4.135333e-16	1.000000	NaN	NaN	NaN	NaN
0	lasso	none	1.248918e-01	2.417295e-02	1.554765e-01	-0.000013	0.124892	0.024173	0.155477	-1.000013

We can observe that LASSO has a higher MAE, MSE, and RMSE value than the baseline linear regression model. This could indicate that LASSO may not make the most sense for this dataset.

L2 (Ridge) Modeling:

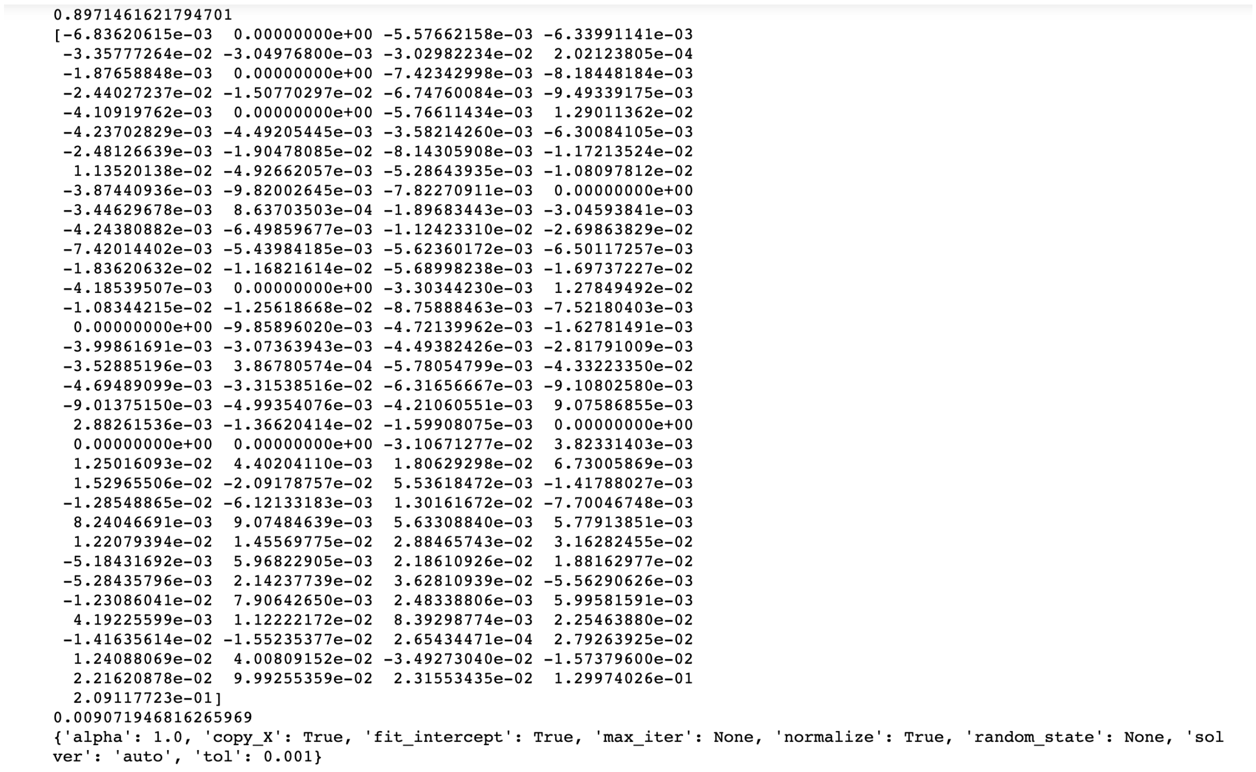
This modeling approach utilizes Ridge regression and utilizes the same training and testing datasets, using a 80% training/testing split. Note the outcome variable has been removed from the testing set.

Ridge Model Setup:



```
l2_mod = Ridge(alpha=1.0, normalize=True).fit(X_train, Y_train)
print(l2_mod.score(X_train, Y_train))
print(l2_mod.coef_)
print(l2_mod.intercept_)
print(l2_mod.get_params())
```

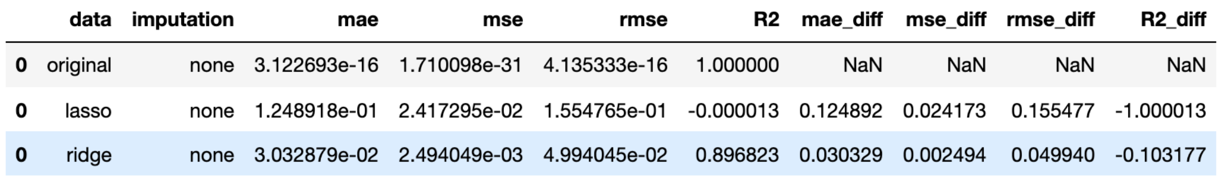
The Ridge model produced the following coefficients (parameters are also listed below)



```
0.8971461621794701
[-6.83620615e-03  0.00000000e+00 -5.57662158e-03 -6.33991141e-03
 -3.35777264e-02 -3.04976800e-03 -3.02982234e-02  2.02123805e-04
 -1.87658848e-03  0.00000000e+00 -7.42342998e-03 -8.18448184e-03
 -2.44027237e-02 -1.50770297e-02 -6.74760084e-03 -9.49339175e-03
 -4.10919762e-03  0.00000000e+00 -5.76611434e-03  1.29011362e-02
 -4.23702829e-03 -4.49205445e-03 -3.58214260e-03 -6.30084105e-03
 -2.48126639e-03 -1.90478085e-02 -8.14305908e-03 -1.17213524e-02
  1.13520138e-02 -4.92662057e-03 -5.28643935e-03 -1.08097812e-02
 -3.87440936e-03 -9.82002645e-03 -7.82270911e-03  0.00000000e+00
 -3.44629678e-03  8.63703503e-04 -1.89683443e-03 -3.04593841e-03
 -4.24380882e-03 -6.49859677e-03 -1.12423310e-02 -2.69863829e-02
 -7.42014402e-03 -5.43984185e-03 -5.62360172e-03 -6.50117257e-03
 -1.83620632e-02 -1.16821614e-02 -5.68998238e-03 -1.69737227e-02
 -4.18539507e-03  0.00000000e+00 -3.30344230e-03  1.27849492e-02
 -1.08344215e-02 -1.25618668e-02 -8.75888463e-03 -7.52180403e-03
  0.00000000e+00 -9.85896020e-03 -4.72139962e-03 -1.62781491e-03
 -3.99861691e-03 -3.07363943e-03 -4.49382426e-03 -2.81791009e-03
 -3.52885196e-03  3.86780574e-04 -5.78054799e-03 -4.33223350e-02
 -4.69489099e-03 -3.31538516e-02 -6.31656667e-03 -9.10802580e-03
 -9.01375150e-03 -4.99354076e-03 -4.21060551e-03  9.07586855e-03
  2.88261536e-03 -1.36620414e-02 -1.59908075e-03  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -3.10671277e-02  3.82331403e-03
  1.25016093e-02  4.40204110e-03  1.80629298e-02  6.73005869e-03
  1.52965506e-02 -2.09178757e-02  5.53618472e-03 -1.41788027e-03
 -1.28548865e-02 -6.12133183e-03  1.30161672e-02 -7.70046748e-03
  8.24046691e-03  9.07484639e-03  5.63308840e-03  5.77913851e-03
  1.22079394e-02  1.45569775e-02  2.88465743e-02  3.16282455e-02
 -5.18431692e-03  5.96822905e-03  2.18610926e-02  1.88162977e-02
 -5.28435796e-03  2.14237739e-02  3.62810939e-02 -5.56290626e-03
 -1.23086041e-02  7.90642650e-03  2.48338806e-03  5.99581591e-03
  4.19225599e-03  1.12222172e-02  8.39298774e-03  2.25463880e-02
 -1.41635614e-02 -1.55235377e-02  2.65434471e-04  2.79263925e-02
  1.24088069e-02  4.00809152e-02 -3.49273040e-02 -1.57379600e-02
  2.21620878e-02  9.99255359e-02  2.31553435e-02  1.29974026e-01
  2.09117723e-01]
0.009071946816265969
{'alpha': 1.0, 'copy_X': True, 'fit_intercept': True, 'max_iter': None, 'normalize': True, 'random_state': None, 'sol
ver': 'auto', 'tol': 0.001}
```

Parameters: Alpha: 1.0, copy x, true. Fit intercept: True, max_iter: none, normalize: true, random state: none, solver: auto, tol: 0.001

The Ridge model produced the following metrics below (and is compared to baseline):



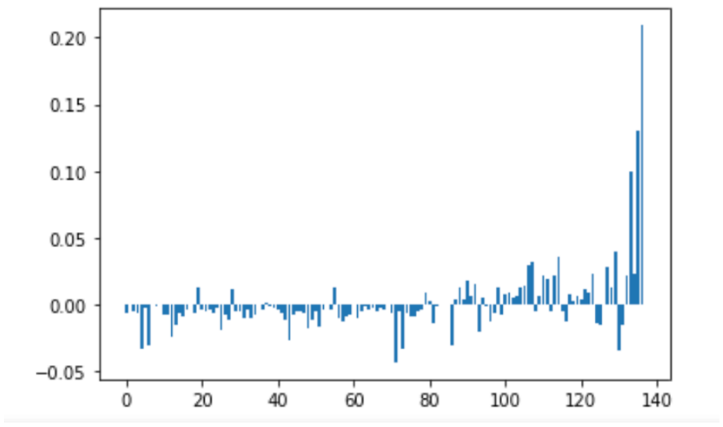
	data	imputation	mae	mse	rmse	R2	mae_diff	mse_diff	rmse_diff	R2_diff
0	original	none	3.122693e-16	1.710098e-31	4.135333e-16	1.000000	NaN	NaN	NaN	NaN
0	lasso	none	1.248918e-01	2.417295e-02	1.554765e-01	-0.000013	0.124892	0.024173	0.155477	-1.000013
0	ridge	none	3.032879e-02	2.494049e-03	4.994045e-02	0.896823	0.030329	0.002494	0.049940	-0.103177

We can observe that Ridge has a lower MAE, MSE, and RMSE value than the LASSO regression model. This could indicate that Ridge is a better fit for this data.

Model Interpretability and Explain-ability

Which variables are most important and why?

Using our Ridge model, the feature importance's plotted as following:



As noted above, all our coefficients were negative, and that is reflected here. Feature 136 in our ridge regression model has the greatest absolute magnitude of 0.2091.

Why discuss feature importance?

Feature importance scores are useful for interpretation of the data, and help to select and rank features of a predictive model. We can use the features that were selected as only use the "most" important ones as inputs to a predictive model. A key idea behind the baseline regression model is to evaluate the model with all the predictors and then evaluate other models with selectively chosen variables. We can also explore results of models with different features.

Most important feature from our Ridge model: wtd_std_Valence

Interpretation of feature: We can measure the importance of this variable (wtd_std_Valence) by calculating the increase in the model's prediction error after manipulating the feature. In our model this feature increased the model's prediction error, because the model relied on this variable for the prediction.

Conclusions:

In short, we recommend using Ridge regression when attempting to predict critical temperatures of superconducting materials. Our conclusion of Ridge regression is based on comparing the modeling results of MSE, MAE, and RMSE against different modeling techniques. When compared to LASSO ridge consistently produces lower validation metrics in our selected three.

We firmly believe that Ridge regression is the route to go down to predict critical temperatures of superconducting materials relative precisely and accurately to the other methods proposed.

We suggest viewing the results linearly and noting the performance metrics of each of the proceeding models (baseline regression and LASSO). Pay close attention to the performance metrics of MSE, MAE, and RMSE (and note that lower is better).

Going forward, we feel that the results using Ridge are very reasonable and given the quick compute times of Ridge relative to the somewhat sizeable dataset present is quite reasonable and provides an accuracy that may not be beaten by a slower converging algorithm.