In [ ]:
```
from IPython.display import HTML
HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here t
```

Out[ ]: Click here to toggle on/off the raw code.

Case Study One

Group Members:

- Alva, Albert
- Gardner, Hollie
- Kumsa, Bethel

Business Understanding - 5 pts

You should always state the objective at the beginning of every case (a guideline you should follow in real life as well) and provide some initial "Business Understanding" statements (i.e., what is trying to be solved for and why might it be important)

The client has provided data on materials that have been identified as potential superconductors. The client would like to know what are the important features that impact a material's critical temperature, or the temperature at which the material acts as a superconductor (source) and more specifically, the critical temperature is the point at which the material "loses all electric resistance" (source).

The critical temperature is important because the expense of cooling can be quite costly. This project will seek to understand what factors are important for developing a material with higher critical temperatures. The commercial implications is that this allows for more plentiful and low-cost refrigerants which have a higher boiling point, like liquid nitrogen, to be used over other more expensive and rare materials with lower boiling points, such as liquid helium (source).

Part One: Data Evaluation/Engineering - 10 pts

Summarize the data being used in the case using appropriate mediums (charts, graphs, tables); address questions such as: Are there missing values? Which variables are needed (which ones

are not)? What assumptions or conclusions are you drawing that need to be relayed to your audience?

Importing Libraries and Reading in Data

In [ ]:

```
#importing libraries
import pandas as pd
import warnings
import sys
from sklearn.preprocessing import StandardScaler
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sb
from simple_colors import *

#modeling and regularization
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV

#evaluation metrics

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
#from ml_metrics import rmse
warnings.filterwarnings("ignore")
```

In [ ]:

```
print(black('Imported packages',['bold', 'underlined']))
print(blue('pandas, warnings, sys, StandardScaler, matplotlib.pyplot, matplotlib
```

Imported packages
pandas, warnings, sys, StandardScaler, matplotlib.pyplot, matplotlib inline,seaborn,simple_colors,
LinearRegression, Lasso Ridge, cross_val_score, mean_squared_error,r2_score, mean_absolute_error

The Data

(21263, 169)

The train dataset contains 21,263 observations with 82 variables. The train dataset contained the critical operating temperature plus 5 summary statistics for 8 variables (listed below)

***Summary Statistics

- Mean
- Geometric mean
- Entropy
- Range
- Standard Deviation

***Summarized Variables

- Atomic Mass
- Atomic Radius
- Density
- Electron Affinity
- First Ionization Energy
- Fusion Heat
- Thermal Conductivity
- Valence
- Number of Elements

***Target variable and non-numeric variable

- Material
- Critical Temperature

The unique_m dataset contains 21,263 observations for 88 elements and are float variables. It also contains the target variable 'critical_temp'

In [ ]:
```
#Reading in Data. We received the data in two separate csv files.
train = pd.read_csv('train.csv')
unique_m = pd.read_csv('unique_m.csv')
print(black('Train dataset\n',['bold']),black(train.dtypes,['bold']))
print(black('unique_m dataset\n',['bold']),black(unique_m.dtypes,['bold']))
```

```
Train dataset
 number_of_elements         int64
mean_atomic_mass           float64
wtd_mean_atomic_mass       float64
gmean_atomic_mass          float64
wtd_gmean_atomic_mass      float64
                             ...
range_Valence                int64
wtd_range_Valence          float64
std_Valence                float64
wtd_std_Valence            float64
critical_temp              float64
Length: 82, dtype: object
unique_m dataset
 H                  float64
He                   int64
Li                 float64
Be                 float64
B                  float64
                    ...
Po                   int64
At                   int64
Rn                   int64
critical_temp      float64
material            object
Length: 88, dtype: object
```

In [ ]:
```
#Dropping critical temp from train, since the column is present in both files.
train.drop('critical_temp', inplace=True, axis=1)
print (black("'Crititcal_temp' is in both datasets, dropped crititcal_temp from
```

```
'Crititcal_temp' is in both datasets, dropped crititcal_temp from train
```

In [ ]:
```
#quick look at data to verify same length
print(black('train shape\n',['bold']),train.shape)
print(black('unique_m shape\n',['bold']),unique_m.shape)
```

```
train shape
 (21263, 81)
unique_m shape
 (21263, 88)
```

In [ ]:
```
# Combining columns of the datasets
SuperConductors = pd.concat([train,unique_m],axis=1)
```

In [ ]:
```
# Creating dataframe named SuperConductors
SuperConductors = pd.DataFrame(SuperConductors)
# verifying df looks as expected
SuperConductors.shape
print(black('Created SuperConductors = train + unique_m\nshape',['bold']),SuperC
```

```
Created SuperConductors = train + unique_m
shape (21263, 169)
```

EDA

Data Summary

- No missing values

Dropped Features

- material - non-numeric data type
- 'He', 'Ne', 'Ar', 'Kr', 'Xe', 'Pm', 'Po', 'At', 'Rn' - these features had only 0's in the columns and would not add to the model

Analytical Dataset

The analytical dataset.SuperConductors contains 21,263 observations and 159 numeric features. The predictor and target variables are on the same dataset.

Impression

Predictors

There predictor summary statistics showed high variance among the variable means. This would indicate normalization is needed to equate the variable means and standard deviation.
Summary statistics for predictor means

- The smallest mean = 0.00229
- The largest mean = 8665.43882
- Average = 277.16908
- Standard Deviation = 1083.74335

Target Variable

The graph below shows the critical_temp is not normally distributed.

In [ ]:

```
# checking to see if all data types are numeric
pd.set_option('display.max_rows', None)
print(black('SuperConductors data types\n',['bold']),SuperConductors.dtypes)
pd.reset_option('display.max_rows')
```

SuperConductors data types

```
 number_of_elements                int64
mean_atomic_mass                 float64
wtd_mean_atomic_mass             float64
gmean_atomic_mass                float64
wtd_gmean_atomic_mass            float64
entropy_atomic_mass              float64
wtd_entropy_atomic_mass          float64
range_atomic_mass                float64
wtd_range_atomic_mass            float64
std_atomic_mass                  float64
wtd_std_atomic_mass              float64
mean_fie                         float64
wtd_mean_fie                     float64
gmean_fie                        float64
wtd_gmean_fie                    float64
entropy_fie                      float64
wtd_entropy_fie                  float64
range_fie                        float64
wtd_range_fie                    float64
std_fie                          float64
wtd_std_fie                      float64
mean_atomic_radius               float64
wtd_mean_atomic_radius           float64
gmean_atomic_radius              float64
wtd_gmean_atomic_radius          float64
entropy_atomic_radius            float64
wtd_entropy_atomic_radius        float64
range_atomic_radius                int64
wtd_range_atomic_radius          float64
std_atomic_radius                float64
wtd_std_atomic_radius            float64
mean_Density                     float64
wtd_mean_Density                 float64
gmean_Density                    float64
wtd_gmean_Density                float64
entropy_Density                  float64
wtd_entropy_Density              float64
range_Density                    float64
wtd_range_Density                float64
std_Density                      float64
wtd_std_Density                  float64
mean_ElectronAffinity            float64
wtd_mean_ElectronAffinity        float64
gmean_ElectronAffinity           float64
wtd_gmean_ElectronAffinity       float64
entropy_ElectronAffinity         float64
wtd_entropy_ElectronAffinity     float64
range_ElectronAffinity           float64
wtd_range_ElectronAffinity       float64
std_ElectronAffinity             float64
wtd_std_ElectronAffinity         float64
mean_FusionHeat                  float64
wtd_mean_FusionHeat              float64
gmean_FusionHeat                 float64
```

```
wtd_gmean_FusionHeat               float64
entropy_FusionHeat                 float64
wtd_entropy_FusionHeat             float64
range_FusionHeat                   float64
wtd_range_FusionHeat               float64
std_FusionHeat                     float64
wtd_std_FusionHeat                 float64
mean_ThermalConductivity           float64
wtd_mean_ThermalConductivity       float64
gmean_ThermalConductivity          float64
wtd_gmean_ThermalConductivity      float64
entropy_ThermalConductivity        float64
wtd_entropy_ThermalConductivity    float64
range_ThermalConductivity          float64
wtd_range_ThermalConductivity      float64
std_ThermalConductivity            float64
wtd_std_ThermalConductivity        float64
mean_Valence                       float64
wtd_mean_Valence                   float64
gmean_Valence                      float64
wtd_gmean_Valence                  float64
entropy_Valence                    float64
wtd_entropy_Valence                float64
range_Valence                        int64
wtd_range_Valence                  float64
std_Valence                        float64
wtd_std_Valence                    float64
H                                  float64
He                                   int64
Li                                 float64
Be                                 float64
B                                  float64
C                                  float64
N                                  float64
O                                  float64
F                                  float64
Ne                                   int64
Na                                 float64
Mg                                 float64
Al                                 float64
Si                                 float64
P                                  float64
S                                  float64
Cl                                 float64
Ar                                   int64
K                                  float64
Ca                                 float64
Sc                                 float64
Ti                                 float64
V                                  float64
Cr                                 float64
Mn                                 float64
Fe                                 float64
Co                                 float64
Ni                                 float64
Cu                                 float64
Zn                                 float64
Ga                                 float64
Ge                                 float64
As                                 float64
Se                                 float64
Br                                 float64
Kr                                   int64
Rb                                 float64
Sr                                 float64
```

```
Y                                 float64
Zr                                float64
Nb                                float64
Mo                                float64
Tc                                float64
Ru                                float64
Rh                                float64
Pd                                float64
Ag                                float64
Cd                                float64
In                                float64
Sn                                float64
Sb                                float64
Te                                float64
I                                 float64
Xe                                  int64
Cs                                float64
Ba                                float64
La                                float64
Ce                                float64
Pr                                float64
Nd                                float64
Pm                                  int64
Sm                                float64
Eu                                float64
Gd                                float64
Tb                                float64
Dy                                float64
Ho                                float64
Er                                float64
Tm                                float64
Yb                                float64
Lu                                float64
Hf                                float64
Ta                                float64
W                                 float64
Re                                float64
Os                                float64
Ir                                float64
Pt                                float64
Au                                float64
Hg                                float64
Tl                                float64
Pb                                float64
Bi                                float64
Po                                  int64
At                                  int64
Rn                                  int64
critical_temp                     float64
material                           object
dtype: object
```

In [ ]:

```
# Material is an object - deleting
SuperConductors = SuperConductors.drop(['material'], axis=1)
print(black("'material' is an object data type and was deleted\n",['bold']))
```

'material' is an object data type and was deleted

Checking for missing values

In [ ]:

```
# Checking for missing values
null_counts = SuperConductors.isnull().sum()
```

```
print(black("Checked for null values, none found\n",['bold']),null_counts[null_c
```

Checked for null values, none found
Series([], dtype: int64)

Checking for features with only one value

In [ ]:
```
# Finding features with only one value
Myunique = []
for col in SuperConductors.columns:
    if len (SuperConductors[col].unique ()) == 1:
        Myunique.append(col)
print(black("The following features have only one value:\n",['bold']),Myunique)
```

The following features have only one value:
['He', 'Ne', 'Ar', 'Kr', 'Xe', 'Pm', 'Po', 'At', 'Rn']

In [ ]:
```
# all nine features with single value have value of 0
pd.set_option('display.max_rows', None)
print(black("All nine features have only the value of zero\n",['bold']))
to_drop = SuperConductors[['He', 'Ne', 'Ar', 'Kr', 'Xe', 'Pm', 'Po', 'At', 'Rn']
#to_drop

print(black("Maximum Value\n",['bold']),black(to_drop.max(),['bold']))

#print(to_drop.max())

print(black("Minimum Value\n",['bold']),black(to_drop.min(),['bold']))
```

All nine features have only the value of zero

Maximum Value
He 0
Ne 0
Ar 0
Kr 0
Xe 0
Pm 0
Po 0
At 0
Rn 0
dtype: int64
Minimum Value
He 0
Ne 0
Ar 0
Kr 0
Xe 0
Pm 0
Po 0
At 0
Rn 0
dtype: int64

In [ ]:
```
# Dropping features wih only 0 values.
SuperConductors = SuperConductors.drop(
    ['He', 'Ne', 'Ar', 'Kr', 'Xe', 'Pm', 'Po', 'At', 'Rn'], axis=1)
print(black("None of the nine features add to the model and were dropped\n",['bo
```

None of the nine features add to the model and were dropped

Verifying all features are numeric

In [ ]:

```
# checking to see if all data types are numeric
pd.set_option('display.max_rows', None)

print(black("Verifying remaining Features are numeric\n",['bold']),black(SuperCo
pd.reset_option('display.max_rows')
```

```
Verifying remaining Features are numeric
 number_of_elements                  int64
mean_atomic_mass                   float64
wtd_mean_atomic_mass               float64
gmean_atomic_mass                  float64
wtd_gmean_atomic_mass              float64
entropy_atomic_mass                float64
wtd_entropy_atomic_mass            float64
range_atomic_mass                  float64
wtd_range_atomic_mass              float64
std_atomic_mass                    float64
wtd_std_atomic_mass                float64
mean_fie                           float64
wtd_mean_fie                       float64
gmean_fie                          float64
wtd_gmean_fie                      float64
entropy_fie                        float64
wtd_entropy_fie                    float64
range_fie                          float64
wtd_range_fie                      float64
std_fie                            float64
wtd_std_fie                        float64
mean_atomic_radius                 float64
wtd_mean_atomic_radius             float64
gmean_atomic_radius                float64
wtd_gmean_atomic_radius            float64
entropy_atomic_radius              float64
wtd_entropy_atomic_radius          float64
range_atomic_radius                  int64
wtd_range_atomic_radius            float64
std_atomic_radius                  float64
wtd_std_atomic_radius              float64
mean_Density                       float64
wtd_mean_Density                   float64
gmean_Density                      float64
wtd_gmean_Density                  float64
entropy_Density                    float64
wtd_entropy_Density                float64
range_Density                      float64
wtd_range_Density                  float64
std_Density                        float64
wtd_std_Density                    float64
mean_ElectronAffinity              float64
wtd_mean_ElectronAffinity          float64
gmean_ElectronAffinity             float64
wtd_gmean_ElectronAffinity         float64
entropy_ElectronAffinity           float64
wtd_entropy_ElectronAffinity       float64
range_ElectronAffinity             float64
wtd_range_ElectronAffinity         float64
std_ElectronAffinity               float64
wtd_std_ElectronAffinity           float64
mean_FusionHeat                    float64
```

wtd_mean_FusionHeat	float64
gmean_FusionHeat	float64
wtd_gmean_FusionHeat	float64
entropy_FusionHeat	float64
wtd_entropy_FusionHeat	float64
range_FusionHeat	float64
wtd_range_FusionHeat	float64
std_FusionHeat	float64
wtd_std_FusionHeat	float64
mean_ThermalConductivity	float64
wtd_mean_ThermalConductivity	float64
gmean_ThermalConductivity	float64
wtd_gmean_ThermalConductivity	float64
entropy_ThermalConductivity	float64
wtd_entropy_ThermalConductivity	float64
range_ThermalConductivity	float64
wtd_range_ThermalConductivity	float64
std_ThermalConductivity	float64
wtd_std_ThermalConductivity	float64
mean_Valence	float64
wtd_mean_Valence	float64
gmean_Valence	float64
wtd_gmean_Valence	float64
entropy_Valence	float64
wtd_entropy_Valence	float64
range_Valence	int64
wtd_range_Valence	float64
std_Valence	float64
wtd_std_Valence	float64
H	float64
Li	float64
Be	float64
B	float64
C	float64
N	float64
O	float64
F	float64
Na	float64
Mg	float64
Al	float64
Si	float64
P	float64
S	float64
Cl	float64
K	float64
Ca	float64
Sc	float64
Ti	float64
V	float64
Cr	float64
Mn	float64
Fe	float64
Co	float64
Ni	float64
Cu	float64
Zn	float64
Ga	float64
Ge	float64
As	float64
Se	float64
Br	float64
Rb	float64
Sr	float64
Y	float64
Zr	float64

```
Nb                                float64
Mo                                float64
Tc                                float64
Ru                                float64
Rh                                float64
Pd                                float64
Ag                                float64
Cd                                float64
In                                float64
Sn                                float64
Sb                                float64
Te                                float64
I                                 float64
Cs                                float64
Ba                                float64
La                                float64
Ce                                float64
Pr                                float64
Nd                                float64
Sm                                float64
Eu                                float64
Gd                                float64
Tb                                float64
Dy                                float64
Ho                                float64
Er                                float64
Tm                                float64
Yb                                float64
Lu                                float64
Hf                                float64
Ta                                float64
W                                 float64
Re                                float64
Os                                float64
Ir                                float64
Pt                                float64
Au                                float64
Hg                                float64
Tl                                float64
Pb                                float64
Bi                                float64
critical_temp                     float64
dtype: object
```

In [ ]:
```
print(black("Cleaned dataset shape\n",['bold']), SuperConductors.shape)
```

```
Cleaned dataset shape
 (21263, 159)
```

In [ ]:
```
pd.set_option('display.max_columns', None)
print(black("Cleaned dataset Summary Statistics\n",['bold']))
SuperConductors.describe()
```

Cleaned dataset Summary Statistics

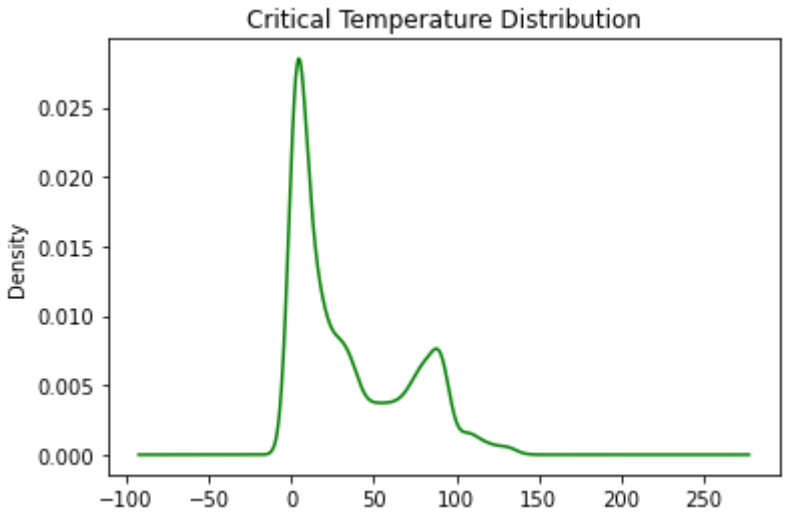
Out[ ]:

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd
count	21263.000000	21263.000000	21263.000000	21263.000000	
mean	4.115224	87.557631	72.988310	71.290627	
std	1.439295	29.676497	33.490406	31.030272	

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd
min	1.000000	6.941000	6.423452	5.320573	
25%	3.000000	72.458076	52.143839	58.041225	
50%	4.000000	84.922750	60.696571	66.361592	
75%	5.000000	100.404410	86.103540	78.116681	
max	9.000000	208.980400	208.980400	208.980400	

In [ ]:

```
#plt.plot(SuperConductors['critical_temp'])
SuperConductors.critical_temp.plot.density(color='green')
plt.title('Critical Temperature Distribution')

plt.show()
```



In [ ]:

```
x = SuperConductors.drop(['critical_temp'], axis=1)
y = SuperConductors['critical_temp']

print(black("Summary statistics for predictor means \n The smallest mean = ",['b
      ,blue(round(x.mean().min(),5),['bold'])
      , '\n'
      ,black("The largest mean =",['bold'])
      ,blue(round(x.mean().max(),5),['bold'])
      , '\n'
      ,black("Average =",['bold'])
      ,blue(round(x.mean().mean(),5),['bold'])
      , '\n'
      ,black("Standard Deviation =",['bold'])
      ,blue(round(x.mean().std(),5),['bold'])

     )
```

```
Summary statistics for predictor means
 The smallest mean =  0.00229
 The largest mean = 8665.43882
 Average = 277.16908
 Standard Deviation = 1083.74335
```

Correlation results

The majority of the features were correlated with other feature.
The heatmaps below show all features and the other shows just the features with correlations of at least .60.

Examples of correlated predictors:

- wtd_mean_Valence wtd_gmean_Valence 0.994939
- wtd_mean_fie wtd_gmean_fie 0.992331
- mean_Valence gmean_Valence 0.989911

Critical Temperature Correlations

Critical Temperature was correlated with 11 features at .6 or above.

- wtd_std_ThermalConductivity 0.7213
- range_ThermalConductivity 0.6877
- range_atomic_radius 0.6538
- std_ThermalConductivity 0.6536
- wtd_entropy_atomic_mass 0.6269
- wtd_entropy_atomic_radius 0.6035
- number_of_elements 0.6011
- range_fie 0.6008
- mean_Valence -0.6001
- wtd_gmean_Valence -0.6157
- wtd_mean_Valence -0.6324

Correlations

Correlations with Critical Temp

In [ ]:
```
scCorr = SuperConductors.corr()
sorted_mat = scCorr.unstack()
CTcorr = (sorted_mat.critical_temp).sort_values(ascending=False)
#Absolute correlations of at least .6
scCorr_Filtered = scCorr[((scCorr >= 0.6) | (scCorr <= -0.6)) & (scCorr !=1.000)]



print(black('Top 10 Correlations critical_temp sorted Decending.\n',['bold'])
      ,black(round(CTcorr,4),['bold'])
     )
```

```
Top 10 Correlations critical_temp sorted Decending.
 critical_temp                1.0000
wtd_std_ThermalConductivity    0.7213
range_ThermalConductivity      0.6877
range_atomic_radius            0.6538
std_ThermalConductivity        0.6536
                               ...
gmean_Density                 -0.5417
gmean_Valence                 -0.5731
mean_Valence                  -0.6001
```

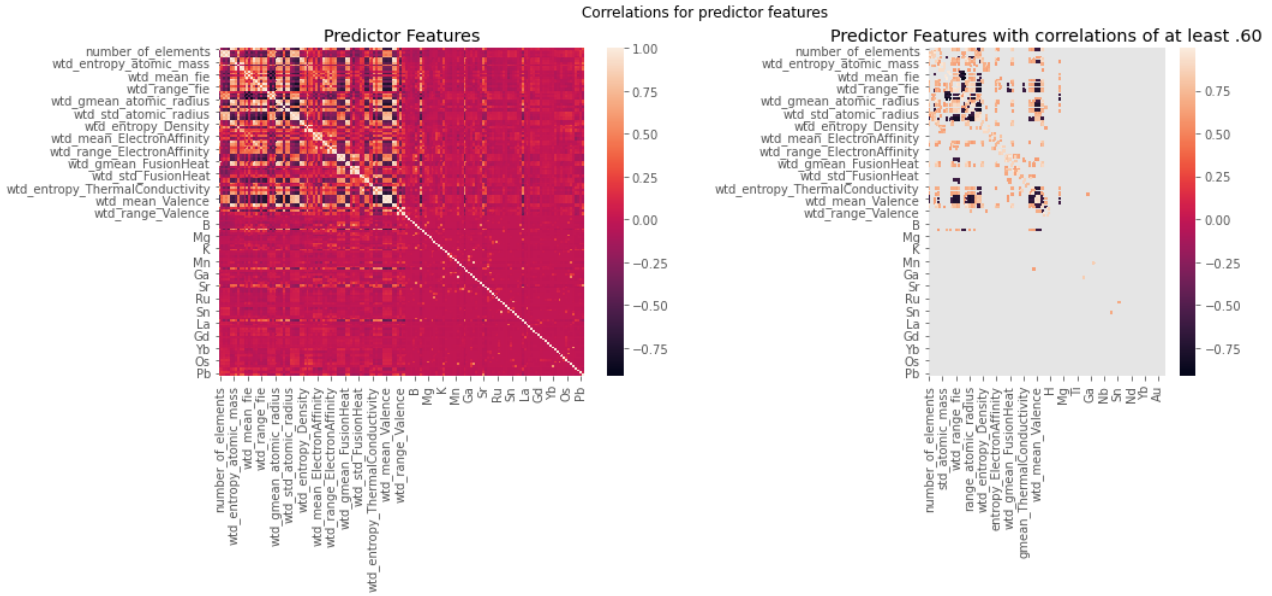
```
wtd_gmean_Valence              -0.6157
wtd_mean_Valence               -0.6324
Length: 159, dtype: float64
```

Predictor Correlations

Predictor feature correlation matrix showing high colinearity

In [ ]:

```
mycorr = x.corr()
plt.style.use('ggplot')
plt.figure(figsize = (15, 5))
plt.suptitle('Correlations for predictor features')

plt.subplot(1, 2, 1) # row 1, col 2 index 1
plt.title('Predictor Features')
sb.heatmap(mycorr, annot= False)

plt.subplot(1, 3, 3) # row 1, col 2 index 1
plt.title('Predictor Features with correlations of at least .60')
x_Filtered = mycorr[((mycorr >= 0.6) | (mycorr <= -0.6)) & (mycorr !=1.000)]

sb.heatmap(x_Filtered, annot= False)



plt.show()
```



Part Two: Modeling Preparations - 10 pts

Which methods are you proposing to utilize to solve the problem? Why is this method appropriate given the business objective?

The team has decided to utilize linear regression and Ridge regularization in order to predict the critical temperature of the superconductor materials based on the data provided to us. Since high interpretability is crucial to the business need, linear regression techniques can be used to perform quick, reliable predictions that are more interpretable over other types of machine

learning techniques. The RIDGE regularization was selected due to the data being highly colinear. The predictor variables are normalized to insure homogeneity of variance.

How will you determine if your approach is useful (or how will you differentiate which approach is more useful than another)? More specifically, what evaluation metrics are most useful given that the problem is a regression one (ex., RMSE, logloss, MAE, etc.)?

The team will use RMSE to serve as the metric for our model evaluation. Root mean square error gives us an idea of how far off our predictions are and we want to identify the parameters, such as the regularization loss penalty (alpha), that will produce accurate predictions with the least amount of error without also overfitting our model to the training data. In addition, the team will use MSE, MAE, and R^2 to further evaluate the effectiveness and accuracy of the model.

In [ ]:
```
# Making X and Y
x = SuperConductors.drop(['critical_temp'], axis=1)
y = SuperConductors['critical_temp']
```

Normalizing the Data

The means of the predictor features have large variability.

The feature means range from .002 to 8665.439 with a standard deviation of 1080.480.

In [ ]:
```
scaler = StandardScaler()
```

In [ ]:
```
x_scaled = scaler.fit_transform(x)
```

In [ ]:
```
#x_scaled
```

In [ ]:
```
# adding names back to new x_scaled
feature_names = list(x)
x_scaled = pd.DataFrame(x_scaled, columns = feature_names)
```

Results of normalization

In [ ]:
```
pd.set_option('display.float_format', lambda x: '%.5f' % x)
print(black( 'Scaled Predictor Data Summary',['bold']))
x_scaled.describe()
```

Out[ ]:

Scaled Predictor Data Summary

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd
count	21263.00000	21263.00000	21263.00000	21263.00000	
mean	-0.00000	-0.00000	-0.00000	-0.00000	
std	1.00002	1.00002	1.00002	1.00002	

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd
min	-2.16446	-2.71658	-1.98763	-2.12604	
25%	-0.77486	-0.50882	-0.62242	-0.42699	
50%	-0.08006	-0.08879	-0.36703	-0.15885	
75%	0.61474	0.43290	0.39162	0.21999	
max	3.39395	4.09164	4.06072	4.43738	

Part Three: Model Building and Evaluation - 40 pts

In this case, your primary task is to build a linear regression model using L1 or L2 regularization (or both) to predict the critical temperature and will involve the following steps:

Specify your sampling methodology

Our sampling methodology is to create a 70/30 split of training and test data. While in this iteration the data is standardized prior to the split of the training and test data, future iterations of this analysis will involve standarization practices that do not create data leakage.

Setup your model(s) - specifying the regularization type chosen and including the parameters utilized by the model

In addition to a simple linear regression, the team has opted to perform an L2, or Ridge regularlarization, regression model. The alpha, or loss, found to be the optimal amount was 1.0 which can be seen below.

Analyze your model's performance - referencing your chosen evaluation metric (including supplemental visuals and analysis where appropriate)

R2:0.750, MSE:286.21, RMSE:16.92

Linear Regression

Creating x_train x_test y_train y_test

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.3,

print('x_train shape', x_train.shape)
print('y_train shape', y_train.shape)
print('x_test shape', x_test.shape)
print('y_test shape', y_test.shape)
```

```
x_train shape (14884, 158)
y_train shape (14884,)
x_test shape (6379, 158)
y_test shape (6379,)
```

Ridge Regularization

In [ ]:
```
print(black('Innitial Training RIDGE model',['bold']))
l2_mod = Ridge(alpha=0, normalize=False).fit(x_train, y_train)
print(l2_mod.get_params())
```

```
Innitial Training RIDGE model
{'alpha': 0, 'copy_X': True, 'fit_intercept': True, 'max_iter': None, 'normaliz
e': False, 'random_state': None, 'solver': 'auto', 'tol': 0.001}
```

In [ ]:
```
# Innitial model Coefficients
l2_mod_coef = pd.DataFrame(l2_mod.coef_)
feature_name = pd.DataFrame(feature_names)
l2_mod_coefs = pd.concat([feature_name,l2_mod_coef],axis=1)
l2_mod_coefs.columns = ['Features', 'Coefficients']
pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.9f' % x)
print(black('Linear Regression Coefficients',['bold']))
#order - greatest to least to support business need of creating higher crit temp
l2_mod_coefs.sort_values(by='Coefficients',ascending=False ,key=abs)
```

Linear Regression Coefficients

Out[ ]:

	Features	Coefficients
24	wtd_gmean_atomic_radius	-44.899742900
22	wtd_mean_atomic_radius	44.533539433
2	wtd_mean_atomic_mass	-40.809621882
52	wtd_mean_FusionHeat	-29.335837662
49	std_ElectronAffinity	26.416083523
1	mean_atomic_mass	26.225582661
76	wtd_entropy_Valence	-26.057044499
4	wtd_gmean_atomic_mass	25.668638208
12	wtd_mean_fie	25.485147184
62	wtd_mean_ThermalConductivity	24.810334739
75	entropy_Valence	23.089944547
25	entropy_atomic_radius	-21.787900772
47	range_ElectronAffinity	-20.632183897
11	mean_fie	-20.591312849
69	std_ThermalConductivity	19.728483824
54	wtd_gmean_FusionHeat	18.901346338
67	range_ThermalConductivity	-17.798618266
32	wtd_mean_Density	17.115315534
16	wtd_entropy_fie	16.870734220
31	mean_Density	-16.703720835
51	mean_FusionHeat	16.004714277
44	wtd_gmean_ElectronAffinity	-15.909271650

	Features	Coefficients
13	gmean_fie	15.813892506
14	wtd_gmean_fie	-15.730517642
72	wtd_mean_Valence	-15.700706724
64	wtd_gmean_ThermalConductivity	-15.460867038
17	range_fie	14.971135536
3	gmean_atomic_mass	-13.649390979
42	wtd_mean_ElectronAffinity	13.457942614
74	wtd_gmean_Valence	12.744570946
50	wtd_std_ElectronAffinity	-11.104678414
15	entropy_fie	10.840718806
29	std_atomic_radius	-10.758005081
59	std_FusionHeat	-10.352584903
7	range_atomic_mass	10.088856866
9	std_atomic_mass	-9.933473596
19	std_fie	-9.307790651
56	wtd_entropy_FusionHeat	9.293269020
55	entropy_FusionHeat	-9.082098026
5	entropy_atomic_mass	-9.076435040
53	gmean_FusionHeat	-8.990733308
21	mean_atomic_radius	-8.307677914
68	wtd_range_ThermalConductivity	-8.053270483
77	range_Valence	7.980407498
58	wtd_range_FusionHeat	7.869926168
131	Ba	7.407185450
26	wtd_entropy_atomic_radius	7.085672313
60	wtd_std_FusionHeat	7.013508291
80	wtd_std_Valence	-7.012208228
39	std_Density	6.915531428
20	wtd_std_fie	-6.487081258
27	range_atomic_radius	5.859659112
46	wtd_entropy_ElectronAffinity	-5.415700724
43	gmean_ElectronAffinity	4.479215652
157	Bi	4.373627284
71	mean_Valence	4.126779408
33	gmean_Density	4.092484157

	Features	Coefficients
37	range_Density	-4.089764631
8	wtd_range_atomic_mass	3.543842218
45	entropy_ElectronAffinity	3.537682336
18	wtd_range_fie	3.466260484
65	entropy_ThermalConductivity	3.364747168
6	wtd_entropy_atomic_mass	3.318903378
23	gmean_atomic_radius	3.310495738
92	Si	-2.923526002
34	wtd_gmean_Density	-2.817448584
28	wtd_range_atomic_radius	-2.735923413
57	range_FusionHeat	-2.584580805
30	wtd_std_atomic_radius	2.564986537
38	wtd_range_Density	-2.348549782
40	wtd_std_Density	-2.301276963
41	mean_ElectronAffinity	-2.075845115
61	mean_ThermalConductivity	-2.040583971
36	wtd_entropy_Density	-2.023821483
63	gmean_ThermalConductivity	-2.014400603
48	wtd_range_ElectronAffinity	-1.986792145
78	wtd_range_Valence	-1.939665211
123	Ag	-1.879734812
110	As	-1.684799222
79	std_Valence	-1.677373347
155	Tl	1.581966062
106	Cu	-1.562778574
94	S	-1.515067485
87	O	-1.448522167
152	Pt	1.347941204
154	Hg	1.312879729
96	K	1.270719839
70	wtd_std_ThermalConductivity	-1.232854075
113	Rb	1.195812472
88	F	1.109016908
103	Fe	0.972300947
97	Ca	0.960886842

	Features	Coefficients
109	Ge	-0.944874906
73	gmean_Valence	-0.935490892
145	Lu	0.873579360
111	Se	-0.739988717
95	Cl	-0.738041269
125	In	0.732493903
93	P	-0.706285211
66	wtd_entropy_ThermalConductivity	0.667891515
89	Na	0.655886845
144	Yb	0.641363509
156	Pb	0.626103277
129	I	0.595853063
0	number_of_elements	0.528185155
153	Au	-0.484757234
130	Cs	0.484262548
82	Li	0.466590581
136	Sm	-0.461219952
133	Ce	-0.460263353
135	Nd	-0.444359909
108	Ga	0.444168569
140	Dy	0.428488265
141	Ho	0.424452876
91	Al	-0.401909888
105	Ni	-0.383076258
117	Nb	0.377998149
10	wtd_std_atomic_mass	0.351513307
118	Mo	0.305957942
114	Sr	-0.304123929
104	Co	-0.290816416
84	B	-0.286614878
150	Os	0.278247669
142	Er	0.254510529
83	Be	-0.241888976
121	Rh	-0.207827786
81	H	-0.175539894

	Features	Coefficients
143	Tm	0.167439282
126	Sn	-0.150519026
120	Ru	0.145727322
90	Mg	0.138913915
99	Ti	-0.136406034
35	entropy_Density	0.132725908
146	Hf	-0.131171003
115	Y	-0.129026758
116	Zr	0.114915309
148	W	0.105039752
124	Cd	-0.094863417
128	Te	0.090182208
119	Tc	0.078878586
149	Re	-0.075354995
122	Pd	-0.064695757
107	Zn	-0.063103875
134	Pr	-0.052693386
85	C	-0.047992553
101	Cr	-0.047606525
132	La	0.040080931
102	Mn	-0.039684944
112	Br	-0.033489410
147	Ta	0.033158815
138	Gd	-0.030154144
137	Eu	0.021921922
139	Tb	0.020987313
98	Sc	-0.016269207
86	N	-0.011984692
127	Sb	-0.010511922
151	Ir	0.004425107
100	V	0.001638383

In [ ]:

```
# Evaluating innitial RIDGE model
l2_y_pred_train = l2_mod.predict(x_train)
l2_mae_train = mean_absolute_error(y_train,l2_y_pred_train)
l2_mse_train = mean_squared_error(y_train,l2_y_pred_train)
```

```
l2_rmse_train = mean_squared_error(y_train,l2_y_pred_train, squared= False)
l2_r2_train = r2_score(y_train,l2_y_pred_train)
alpha = l2_mod.alpha
l2_pred_y_diff_train = pd.DataFrame({'Actual Value': y_train, 'Predicted Value':
                                     y_train-l2_y_pred_train})
print("Alpha:{0:.3f}, R2:{1:.6f}, MAE:{2:6F}, MSE:{3:.6f}, RMSE:{4:.6f}"
      .format(alpha,l2_r2_train, l2_mae_train, l2_mse_train, l2_rmse_train))
```

```
Alpha:0.000, R2:0.767867, MAE:12.360196, MSE:273.612070, RMSE:16.541223
```

Tuning RIDGE model alpha

In [ ]:

```
alphas = [0, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1,0.5, 1,5,10,20,50]
for a in alphas:
    model = Ridge(alpha=a, normalize=False).fit(x_train,y_train)
    score = model.score(x_train, y_train)
    pred_y = model.predict(x_train)
    mse = mean_squared_error(y_train, pred_y)
    mae = mean_absolute_error(y_train, pred_y)
    print("Alpha:{0:.6f}, R2:{1:.3f}, MAE:{2:.5f}, MSE:{3:.5f}, RMSE:{4:.5f}"
    .format(a, score, mae, mse, np.sqrt(mse)))
    print('------------------')
```

```
Alpha:0.000000, R2:0.768, MAE:12.36020, MSE:273.61207, RMSE:16.54122
------------------
Alpha:0.000001, R2:0.768, MAE:12.36020, MSE:273.61207, RMSE:16.54122
------------------
Alpha:0.000010, R2:0.768, MAE:12.36020, MSE:273.61207, RMSE:16.54122
------------------
Alpha:0.000100, R2:0.768, MAE:12.36020, MSE:273.61207, RMSE:16.54122
------------------
Alpha:0.001000, R2:0.768, MAE:12.36020, MSE:273.61207, RMSE:16.54122
------------------
Alpha:0.010000, R2:0.768, MAE:12.36023, MSE:273.61210, RMSE:16.54122
------------------
Alpha:0.100000, R2:0.768, MAE:12.36053, MSE:273.61511, RMSE:16.54132
------------------
Alpha:0.500000, R2:0.768, MAE:12.36195, MSE:273.66517, RMSE:16.54283
------------------
Alpha:1.000000, R2:0.768, MAE:12.36383, MSE:273.76411, RMSE:16.54582
------------------
Alpha:5.000000, R2:0.767, MAE:12.37306, MSE:274.64732, RMSE:16.57249
------------------
Alpha:10.000000, R2:0.766, MAE:12.38354, MSE:275.60095, RMSE:16.60123
------------------
Alpha:20.000000, R2:0.765, MAE:12.40365, MSE:277.19380, RMSE:16.64914
------------------
Alpha:50.000000, R2:0.762, MAE:12.45879, MSE:280.74715, RMSE:16.75551
------------------
```

Training RIDGE model with optimal alpha level alpha=1

In [ ]:

```
ridge_mod=Ridge(alpha=1, normalize=False).fit(x_train,y_train)
ypred = ridge_mod.predict(x_test)
score = model.score(x_test,y_test)
mse = mean_squared_error(y_test,ypred)
mae = mean_absolute_error(y_test,ypred)
ridge_pred_y_diff_test = pd.DataFrame({'Actual Value': y_test, 'Predicted Value'
                                     y_test-ypred})
```

```
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
   .format(score, mse,np.sqrt(mse)))
```

```
R2:0.750, MSE:286.21, RMSE:16.92
```

In [ ]:

```
# Optimized model Coefficients
ridge_mod_coef = pd.DataFrame(ridge_mod.coef_)
feature_name = pd.DataFrame(feature_names)
ridge_mod_coefs = pd.concat([feature_name,ridge_mod_coef],axis=1)
ridge_mod_coefs.columns = ['Features', 'Coefficients']
pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.9f' % x)

print(black('Ridge Regression Coefficients',['bold']))
#order - greatest to least to support business need of creating higher crit temp
ridge_mod_coefs.sort_values(by='Coefficients',ascending=False ,key=abs)
```

Out[ ]:

Ridge Regression Coefficients

	Features	Coefficients
2	wtd_mean_atomic_mass	-34.830064942
22	wtd_mean_atomic_radius	32.252686393
24	wtd_gmean_atomic_radius	-30.205166288
49	std_ElectronAffinity	26.144060981
52	wtd_mean_FusionHeat	-25.541731343
62	wtd_mean_ThermalConductivity	23.754868581
76	wtd_entropy_Valence	-23.384496458
1	mean_atomic_mass	22.589498481
75	entropy_Valence	21.514470002
47	range_ElectronAffinity	-20.273268643
4	wtd_gmean_atomic_mass	19.686713305
69	std_ThermalConductivity	18.343083341
25	entropy_atomic_radius	-17.926562964
67	range_ThermalConductivity	-17.359678700
44	wtd_gmean_ElectronAffinity	-16.063715537
54	wtd_gmean_FusionHeat	15.415231682
16	wtd_entropy_fie	15.403202956
32	wtd_mean_Density	15.392051753
31	mean_Density	-15.332116832
17	range_fie	14.599202460
64	wtd_gmean_ThermalConductivity	-14.477835231
11	mean_fie	-14.264642354
51	mean_FusionHeat	13.725466190
42	wtd_mean_ElectronAffinity	13.665567289

	Features	Coefficients
12	wtd_mean_fie	13.603430445
29	std_atomic_radius	-13.043719810
19	std_fie	-11.524528016
50	wtd_std_ElectronAffinity	-11.090396139
13	gmean_fie	10.481432227
3	gmean_atomic_mass	-10.266421953
7	range_atomic_mass	9.925761498
72	wtd_mean_Valence	-9.463105002
5	entropy_atomic_mass	-9.453522280
56	wtd_entropy_FusionHeat	9.041976591
15	entropy_fie	9.039865293
9	std_atomic_mass	-8.701134777
55	entropy_FusionHeat	-8.475715944
59	std_FusionHeat	-8.365489936
77	range_Valence	7.921653895
68	wtd_range_ThermalConductivity	-7.837346419
58	wtd_range_FusionHeat	7.657012961
80	wtd_std_Valence	-7.515927926
131	Ba	7.383899482
74	wtd_gmean_Valence	7.330349245
53	gmean_FusionHeat	-6.944928051
39	std_Density	6.794323321
27	range_atomic_radius	6.322498628
14	wtd_gmean_fie	-6.247110202
30	wtd_std_atomic_radius	5.804623510
60	wtd_std_FusionHeat	5.517724291
46	wtd_entropy_ElectronAffinity	-5.343021113
26	wtd_entropy_atomic_radius	4.952532590
43	gmean_ElectronAffinity	4.530810285
157	Bi	4.360185042
37	range_Density	-4.356159623
6	wtd_entropy_atomic_mass	4.280892843
8	wtd_range_atomic_mass	3.527843891
45	entropy_ElectronAffinity	3.510522659
18	wtd_range_fie	3.321546716

	Features	Coefficients
20	wtd_std_fie	-3.100868768
65	entropy_ThermalConductivity	2.995673968
92	Si	-2.912653122
57	range_FusionHeat	-2.887202773
28	wtd_range_atomic_radius	-2.810476805
23	gmean_atomic_radius	-2.646251606
63	gmean_ThermalConductivity	-2.570555740
21	mean_atomic_radius	-2.564808330
33	gmean_Density	2.532150700
38	wtd_range_Density	-2.400537473
41	mean_ElectronAffinity	-2.229091850
36	wtd_entropy_Density	-2.179099092
73	gmean_Valence	2.160131096
48	wtd_range_ElectronAffinity	-2.106397635
40	wtd_std_Density	-2.038813626
123	Ag	-1.852838329
110	As	-1.744894021
78	wtd_range_Valence	-1.605430804
106	Cu	-1.579564539
155	Tl	1.563639462
94	S	-1.555966291
61	mean_ThermalConductivity	-1.468369361
152	Pt	1.380789729
79	std_Valence	-1.368485040
87	O	-1.341242488
154	Hg	1.284765713
96	K	1.202410160
113	Rb	1.139784410
88	F	1.086177039
10	wtd_std_atomic_mass	-1.071444113
103	Fe	1.017109748
66	wtd_entropy_ThermalConductivity	0.947665613
109	Ge	-0.931289826
97	Ca	0.931266926
34	wtd_gmean_Density	-0.904550409

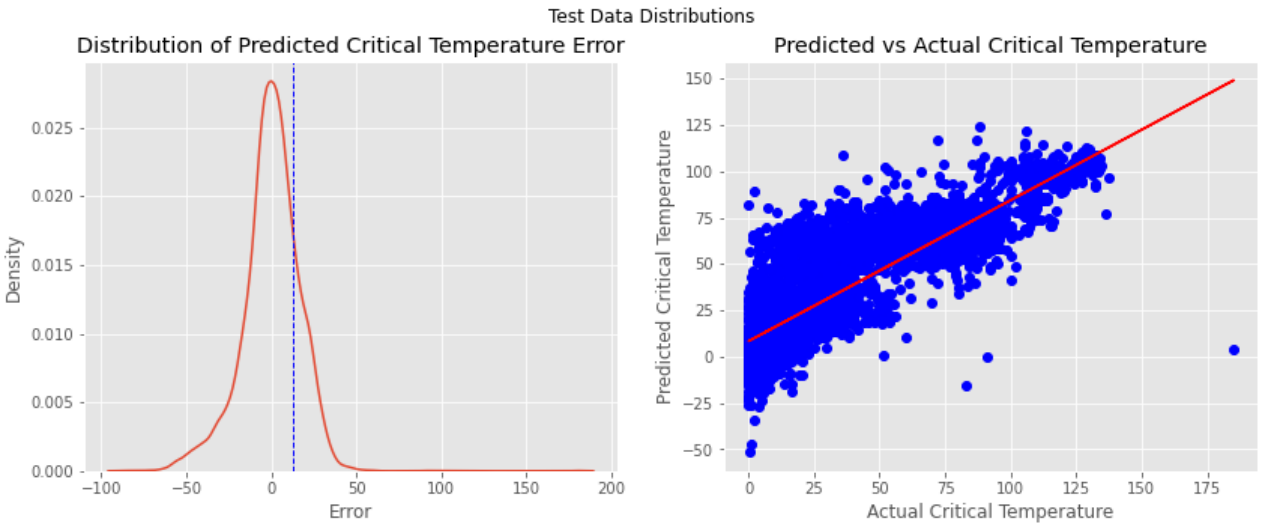
	Features	Coefficients
145	Lu	0.859714745
95	Cl	-0.810106776
111	Se	-0.764552552
125	In	0.747510971
93	P	-0.708380876
144	Yb	0.641815873
156	Pb	0.617596943
89	Na	0.607766432
0	number_of_elements	0.598222678
129	I	0.571254332
153	Au	-0.481680401
133	Ce	-0.471396565
135	Nd	-0.459254051
108	Ga	0.453978824
82	Li	0.444087310
136	Sm	-0.442790230
91	Al	-0.437818210
130	Cs	0.437647585
140	Dy	0.423537156
141	Ho	0.420256515
117	Nb	0.380071156
114	Sr	-0.375403326
105	Ni	-0.372004393
71	mean_Valence	0.360133932
104	Co	-0.298448636
150	Os	0.296001002
118	Mo	0.277142470
84	B	-0.272250609
81	H	-0.249979620
142	Er	0.245965671
83	Be	-0.212521112
121	Rh	-0.187362483
35	entropy_Density	-0.185021494
70	wtd_std_ThermalConductivity	-0.168242162
143	Tm	0.159886191

	Features	Coefficients
120	Ru	0.152732472
99	Ti	-0.142001288
90	Mg	0.129395617
115	Y	-0.128782964
146	Hf	-0.126472574
126	Sn	-0.123927810
116	Zr	0.097575614
124	Cd	-0.095918339
128	Te	0.090742786
149	Re	-0.085126816
119	Tc	0.080493583
148	W	0.071172320
122	Pd	-0.068498613
134	Pr	-0.061908478
107	Zn	-0.054532125
138	Gd	-0.050619717
101	Cr	-0.048576060
85	C	-0.048109796
112	Br	-0.042931910
102	Mn	-0.039653181
132	La	0.036108054
86	N	-0.035540727
147	Ta	0.032972307
139	Tb	0.026395660
127	Sb	-0.020434568
151	Ir	0.011189135
100	V	-0.010799878
137	Eu	0.010752678
98	Sc	-0.006142099

In [ ]:

```
plt.figure(figsize=(14,5))
plt.suptitle('Test Data Distributions')

plt.subplot(1, 2, 1) # row 1, col 2 index 1
sb.distplot(a=ridge_pred_y_diff_test.Difference, hist=False)
plt.axvline(mae, linestyle='dashed', linewidth=1, color='blue')
plt.xlabel('Error')
plt.title('Distribution of Predicted Critical Temperature Error')
```

```
plt.subplot(1, 2, 2) # index 2
plt.plot(y_test, ypred, 'o', color='blue')
m, b = np.polyfit(y_test, ypred, 1)
plt.plot(y_test, m*y_test+b, color='red')
plt.xlabel('Actual Critical Temperature')
plt.ylabel('Predicted Critical Temperature')
plt.title('Predicted vs Actual Critical Temperature')
plt.show()
```



In [ ]:
```
new = x_scaled.loc[5:5]
new
```

Out[ ]:

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gm
5	-0.080057504	0.046732916	-0.453670942	-0.158849738	

In [ ]:
```
y[5]
```

Out[ ]:
```
23.0
```

In [ ]:
```
print("Linear Prediction:",l2_mod.predict(new))
print("RIDGE Prediction:",ridge_mod.predict(new),'\n' )

print("Linear alpha:",l2_mod.alpha)
print("RIDGE alpha:",ridge_mod.alpha,'\n')

print("Linear mae:",mean_absolute_error(y_train,l2_y_pred_train) )
print("RIDGE mae:",mean_absolute_error(y_test,ypred),'\n')

print("Linear rmse:",mean_squared_error(y_train,l2_y_pred_train,squared=False))
print("RIDGE rmse:",mean_squared_error(y_test,ypred,squared=False),'\n')

print("Linear r2:",r2_score(y_train,l2_y_pred_train))
print("RIDGE r2:",r2_score(y_test,ypred),'\n')
```

```
print("linear Regression:",cross_val_score(l2_mod,x_scaled,y).mean())
print("RIDGE:",cross_val_score(ridge_mod,x_scaled,y).mean())
```

```
Linear Prediction: [35.21735867]
RIDGE Prediction: [35.16923057]

Linear alpha: 0
RIDGE alpha: 1

Linear mae: 12.3601958460669
RIDGE mae: 12.610302084821411

Linear rmse: 16.541223357152035
RIDGE rmse: 16.917601852583726

Linear r2: 0.7678670132794634
RIDGE r2: 0.7534192022353778

linear Regression: -0.7944407411247927
RIDGE: -0.7832389184906937
```

Part Four: Model Interpretability & Explanation - 20 pts

Using at least one of your models above (if multiple were trained):

Which variable(s) was (were) "most important" and why?

The following three features are the most important in the prediction model as they had the highest absolute coefficients.

- wtd_mean_atomic_mass -34.830064942
- wtd_mean_atomic_radius 32.252686393
- wtd_gmean_atomic_radius -30.205166288

As the wtd_mean_atomic_mass and the wtd_gmean_atomic_radius increase, the critical temperature falls.
As the wtd_mean_atomic_radius increase, the critical temperature increases.

One concern is that the three variable identified above are highly correlated with each other. The client did not provide a data dictionary, therefore the team did not feel comfortable with removing variables without understanding the full rationale for the inclusion of the variables. It is recommended that the team meets with the client or representatives of the client to discuss which variables can be removed to address the overall issues with the multicollinearity taking place.

How did you come to the conclusion and how should your audience interpret this?

For every unit increase of wtd_mean_atomic_mass there is a 34 unit decrease in the critical temperature. The wtd_mean_atomic_radius and the wtd_gmean_atomic_radius almost negate each other as there is a one unit increase in the former increases critical temperature by 32 units and the latter decreases the critical temperature by 30 units.

Part Five: Case Conclusions - 10 pts

After all of your technical analysis and modeling; what are you proposing to your audience and why? How should they view your results and what should they consider when moving forward? Are there other approaches you'd recommend exploring? This is where you "bring it all home" in language they understand.

The data suffers from collinearity amoung the predictor variables. Before moving forward with modeling, the variables recording the same information (i.e. different calculations of mean, standard deviation, range) should be removed to reduce colllinearity. The above recommended meeting would be necessary to identify which of the variables can be removed based on domain knowledge.

As the conclusions now stand without further meetings, if the client is looking to find a superconductor that doesn't need as many resources to cool, then our recommendation would be to look for materials with less wtd_mean_atomic_mass and less wtd_gmean_atomic_radius and a greater wtd_mean_atomic_radius.

In [ ]: