Case Study 1:
Predicting the Critical Temperatures for Superconductors

Kebur Fantahun, Eli Kravez, Halle Purdom

January 17, 2022

Introduction

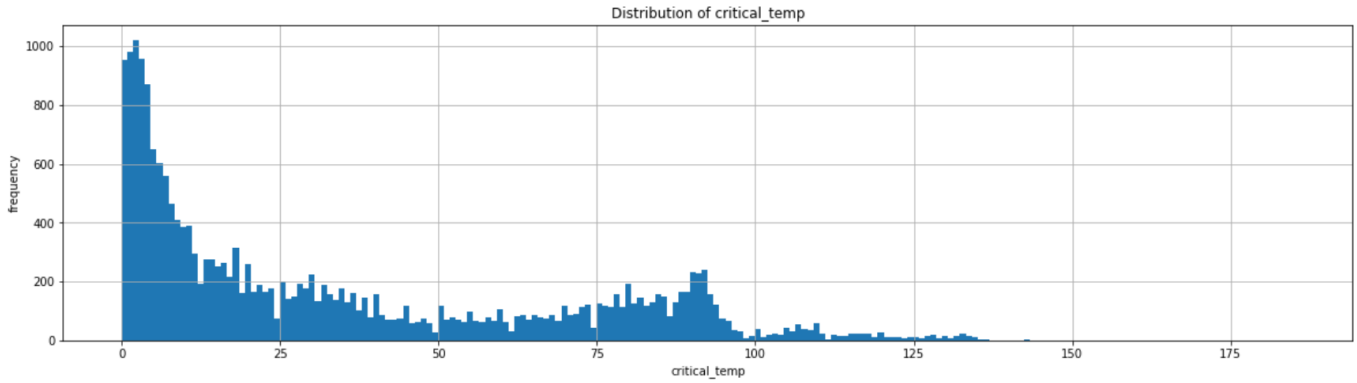
Superconductors are materials that have little to no resistance to electrical current, and are therefore very important to the scientific community for a variety of things like MRI machines, particle colliders, and magnetic levitation trains. Because data is available on known superconductors and their attributes, this data can be used to predict new superconductors with similar attributes. These attributes include the material composition, atomic mass, and critical operating temperature among other features. In this study data from known superconductors is used to develop different regression models that predict the critical temperature at which the superconductor operates.

Methods

Data Preparation

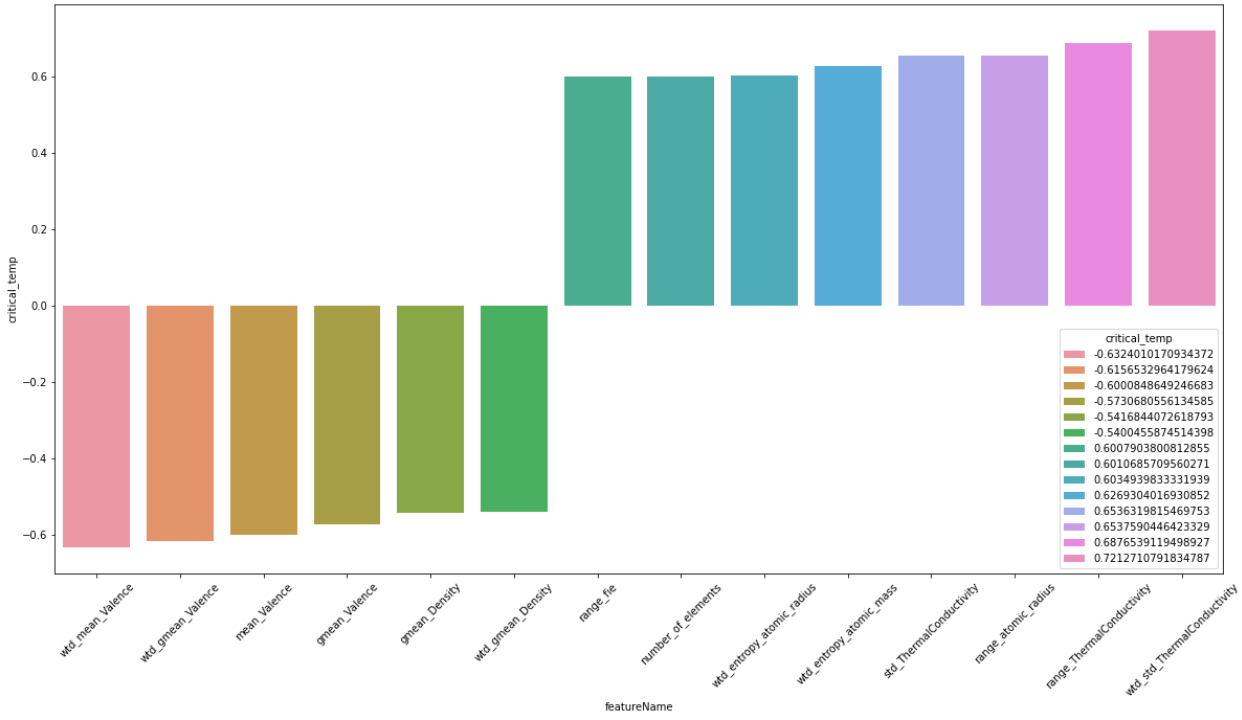
To prepare the data for model development, first the material composition breakdown file is combined with the rest of the superconductor data. There were 66 duplicates in train.csv but after joining with unique_m.csv no duplicates were detected. The duplicate critical temperature column was dropped and the element columns that had a constant value were dropped, as they would not be beneficial to the model. The material column was also dropped as the data from those chemical formulas is encompassed and usable in the individual element columns. The resulting dataset had 21,263 rows and 159 features. The data was also scaled as linear regression models are distance dependent.

The below histogram visualizes the distribution of the target column critical temperature.



In the histogram, the critical temperature appears to be right skewed, with a majority of the superconductors closer to a temperature of zero.

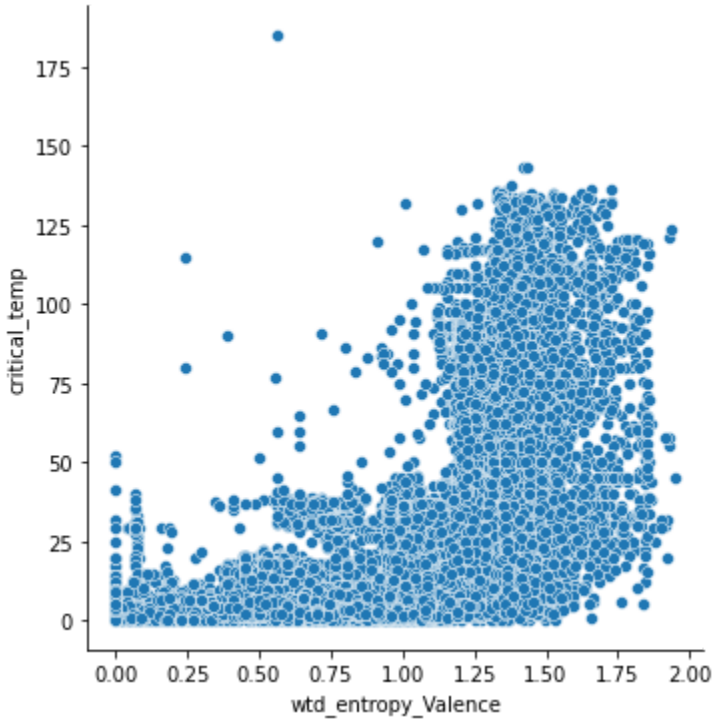
The highest correlations among the features were also considered since this study uses linear regression. Below are the highest correlations in magnitude.



The highest correlation with critical temperature was around 0.7 in magnitude. This indicates that some of these above features may be important in the models later on.

Multicollinearity was also considered as it is an assumption of linear models that the features are not strongly correlated. Ultimately no features were dropped as a result of multicollinearity based on professor feedback. This ensures that all data available is utilized in the model to create an effective model.

In looking further into the critical temperature and how it relates to the other features, many of the relationships appeared to be nonlinear. For example, this nonlinear relationship between the critical temperature and weighted entropy valence is visualized below.



As seen above the relationship between critical temperatures and the weighted entropy valence is not linear and has a curve. Because of this in the model development stage of the study, quadratic terms are introduced to the model.

Train Test Split, Parameter Tuning, and Model Development

The data was divided in an 80/20 split to define the train and test groups. The train group will be used for parameter tuning and the test group in evaluation of the final models.

Three types of models were created: linear regression, lasso regression, and ridge regression. In order to tune the parameters of the lasso and ridge models, 5-fold cross validation was performed on the train data. The RandomizedSearchCV package was used to test many values of alpha for the lasso and ridge models and to find the optimal alpha for each model.

In the 5-fold cross validation used for parameter tuning, some of the metrics for specific folds had much higher MSE values, showing the model was not performing equally on all folds of data. This was also seen after the data was shuffled. In addition, some of the folds were not converging likely because so many of the critical temperature data points are clustered near zero.

This is something to discuss with the dataset providers to get a clearer look into why this may be occurring and what this means for our model development.

The scoring metric used in parameter tuning was negative mean squared error. The final models are then tested using the train data and evaluated using the root mean squared error metric.

Results

Most Important Variables

The features with the highest magnitude coefficients represent the most important features as determined by each model. The most important variables from the lasso regression model are listed below.

Feature Name	Coef	Coef_abs
Ba	11.174418	11.174418
wtd_std_ThermalConductivity^2	10.211003	10.211003
Ca	9.407741	9.407741
range_atomic_mass^2	8.597100	8.597100
Ca^2	-5.778839	5.778839
wtd_std_atomic_mass^2	-4.063470	4.063470
Bi	3.793714	3.793714
wtd_std_Valence	-3.283155	3.283155
wtd_mean_ThermalConductivity	2.930476	2.930476
As^2	-2.758165	2.758165

The most important variables from the ridge regression model are listed below.

Feature Name	Coef	Coef_abs
Ba	7.768679	7.768679
Ca	6.084711	6.084711
wtd_std_ThermalConductivity^2	4.289562	4.289562
Bi	4.242535	4.242535
range_atomic_mass^2	4.166527	4.166527
wtd_mean_ThermalConductivity	4.027410	4.027410
Ca^2	-3.569985	3.569985
wtd_std_Valence	-3.505065	3.505065
wtd_std_atomic_mass^2	-3.104680	3.104680
wtd_std_ThermalConductivity	2.852972	2.852972

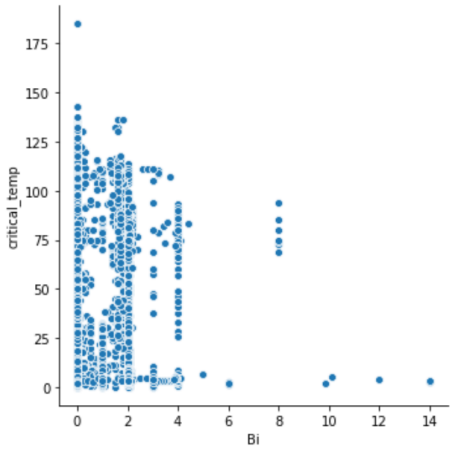
Many of the most important variables as determined by the models overlap, such as weighted standard thermal conductivity, ranged atomic mass, weighted standard valence, and many more. Many of the squared variables are also determined to be important, as seen in the initial nonlinear relationships between critical temperature and the other features.

The goal for this model is to help find new superconductors. These features identified as most important are key features to look at in potential superconductors. In the material composition of superconductors these features like weighted standard thermal conductivity are all indicative of a superconductor and its operating temperature.

More insight into the elemental features Ba, Ca, and Bi was conducted after seeing their high importance in the models. The non-scaled graphs between these features and critical temperature does not indicate that they should be important in linear regressions models. There were no indications of linear or quadratic relationships. In looking further into the StandardScaler package used to scale the data in data preparation, the scaling is not what would be expected. Research into the StandardScaler revealed that “in the presence of outliers, StandardScaler does not guarantee balanced feature scales, due to the influence of the outliers while computing the empirical mean and standard deviation. This leads to the shrinkage in the range of the feature values” (https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/). For example, the maximum Ba value is 24. Below is the StandardScaler formula,

$$z = (x - u) / s$$

Where u is the mean, s is the standard deviation, and x is the specific value. In this case, a Ba=24 value would be scaled to 23.42 due to the mean and standard deviation being less than 1. In looking more into another important element variable, Bi, the Pearson's correlation between the feature and critical temperature was 0.16. The relationship is further visualized below.



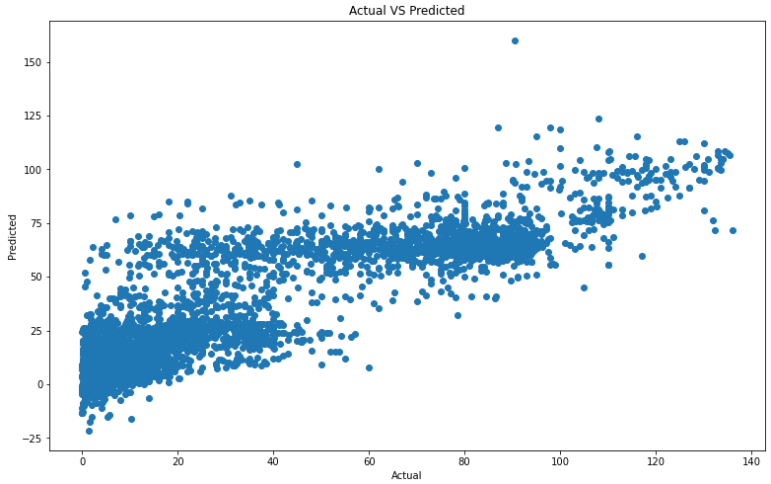
From the graph the relationship does not appear to be strongly linear and does not indicate this variable should be very important in modeling. In an attempt to find a better scaler, several other scaling methods were tried, however these introduced their own issues. There was a time limitation on this study, so future work for this study would include going deeper into these alternative scaling options.

Final Models

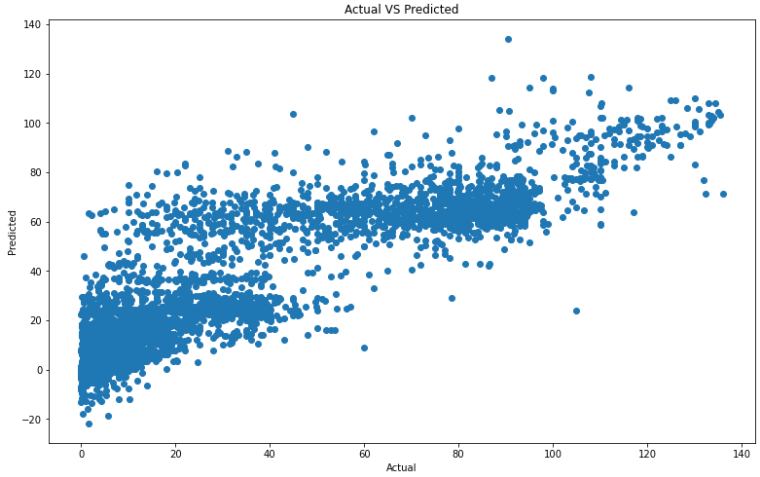
The performance of the final models was evaluated using the test data set. A linear regression model was produced using every feature in order to compare these metrics with the lasso and ridge models.

Model	Alpha	RMSE
Linear Regression Model	Not applicable	17.90
Lasso Regression Model	0.20	16.98
Ridge Regression Model	993	16.60

The resulting predictions on the test data set from the lasso regression model can be seen below.



The critical temperature predictions from the ridge model versus the actual critical temperatures are seen below.



In both graphs above from the lasso and ridge models, we can see that some of the critical temperatures are predicted to be negative, unlike the given data. In taking this work further, other nonlinear models could be explored to help prevent this like k-nearest neighbors. The residuals of each model were also analyzed for each model. While they were not perfectly random clouds as would be ideal for these types of models, they showed significantly more randomness when compared to the same models with no quadratic terms added.

Conclusion

To create a predictive model that has the ability to identify new superconductors based on the material's attributes, three model types were explored: linear regression, lasso regression, and ridge regression. These models helped to identify the most important features of the models,

which included weighted standard thermal conductivity, ranged atomic mass, weighted standard valence, and many more. After tuning the parameters of the lasso and ridge models, the best model of the three as determined by the lowest root mean squared error was the ridge regression model. This result is as expected, because the lasso model is used more for feature selection purposes and the linear regression model does not have any penalties for coefficient growth. In taking this work further, different ways to scale the data would be explored in addition to some nonlinear modeling techniques to see if a more effective model could be produced from the data.

Case Study 1: Predicting the Critical Temperatures for Superconductors

Your case study is to build a linear regression model using L1 or L2 regularization (or both) the task to predict the Critical Temperature as closely as possible. In addition, include in your write-up which variable carries the most importance.

In [2]:
```
# Importing relevant libraries
import pandas as pd

%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns;

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import Lasso  # L1
from sklearn.linear_model import Ridge # L2

from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold

from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import PolynomialFeatures

from sklearn import metrics

import random

import seaborn as sns
```

In [3]:
```
# Reading in the data files
def read_data():
    train_df = pd.read_csv('superconduct/train.csv')
    unique_m_df = pd.read_csv('superconduct/unique_m.csv')
    unique_m_df = unique_m_df.drop(['critical_temp'], axis=1)
    join_data_df = pd.concat([train_df,unique_m_df],axis=1)

    return join_data_df
```

In [4]:
```
reaserch_df = read_data()
```

In [5]:
```
reaserch_df.head()
```

Out[5]:

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_
0	4	88.944468	57.862692	66.361592	

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_
1	5	92.729214	58.518416	73.132787	
2	4	88.944468	57.885242	66.361592	
3	4	88.944468	57.873967	66.361592	
4	4	88.944468	57.840143	66.361592	

5 rows × 169 columns

In [6]:
```
# Data contains 168 features and total of 21,263 rows
reaserch_df.shape
```

Out[6]:
```
(21263, 169)
```

In [7]:
```
# All the data is float or int values and no missing values
print(reaserch_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21263 entries, 0 to 21262
Columns: 169 entries, number_of_elements to material
dtypes: float64(156), int64(12), object(1)
memory usage: 27.4+ MB
None
```

In [8]:
```
# There are no duplicate rows
print(reaserch_df.duplicated().sum())
```

```
0
```

In [9]:
```
# Drop columns which have constant values and therefore do not contribute for pr

columns_to_drop = reaserch_df.columns[reaserch_df.nunique() <= 1]
print(columns_to_drop)

reaserch_df = reaserch_df.drop(columns_to_drop, axis=1)
```

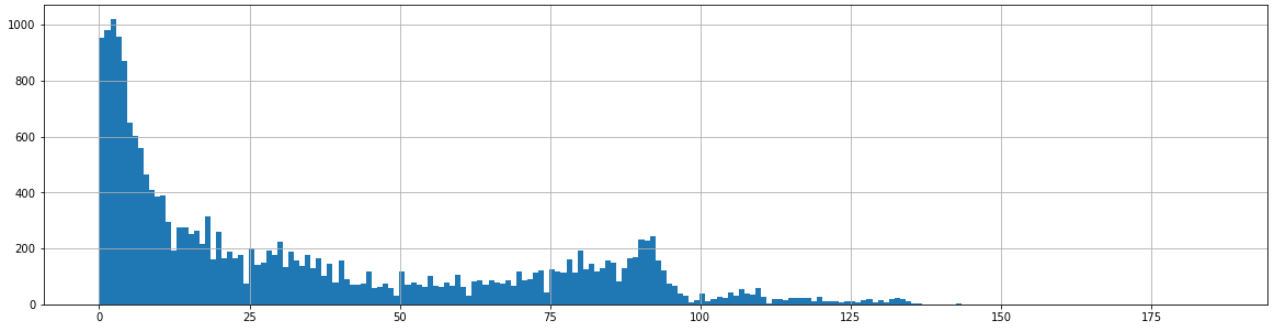
```
Index(['He', 'Ne', 'Ar', 'Kr', 'Xe', 'Pm', 'Po', 'At', 'Rn'], dtype='object')
```

In [10]:
```
# Looking into target feature: critical temperature
reaserch_df['critical_temp'].describe()
```

Out[10]:
```
count    21263.000000
mean        34.421219
std         34.254362
min          0.000210
25%          5.365000
50%         20.000000
75%         63.000000
max        185.000000
Name: critical_temp, dtype: float64
```

In [11]:
```
# Critical_temp is our target variable and it has a very right skewed distributi
reaserch_df['critical_temp'].hist(bins = 200, figsize = (20,5)) # bins = 50
```

```
plt.show()
```



In [12]:
```
# Looking at 'material' feature
print(reaserch_df['material'].nunique())
```

```
15542
```

In [13]:
```
# Decided to drop material as it has too many unique values and the info is enco
reaserch_df = reaserch_df.drop(['material'], axis=1)
```

In [14]:
```
# Looking at number of elements feature
print(reaserch_df['number_of_elements'].value_counts())
```

```
5    5792
4    4496
3    3895
2    3280
6    2666
7     774
1     285
8      61
9      14
Name: number_of_elements, dtype: int64
```

In [15]:
```
# Check correlation between target and features - for EDA
dfCorr = reaserch_df.corr()['critical_temp'][:]
filteredDf = dfCorr[((dfCorr >= .6) | (dfCorr <= -.5)) & (dfCorr !=1)]
```

In [16]:
```
corr_df = filteredDf.to_frame().sort_values('critical_temp')
corr_df.index.names = ['featureName']
```

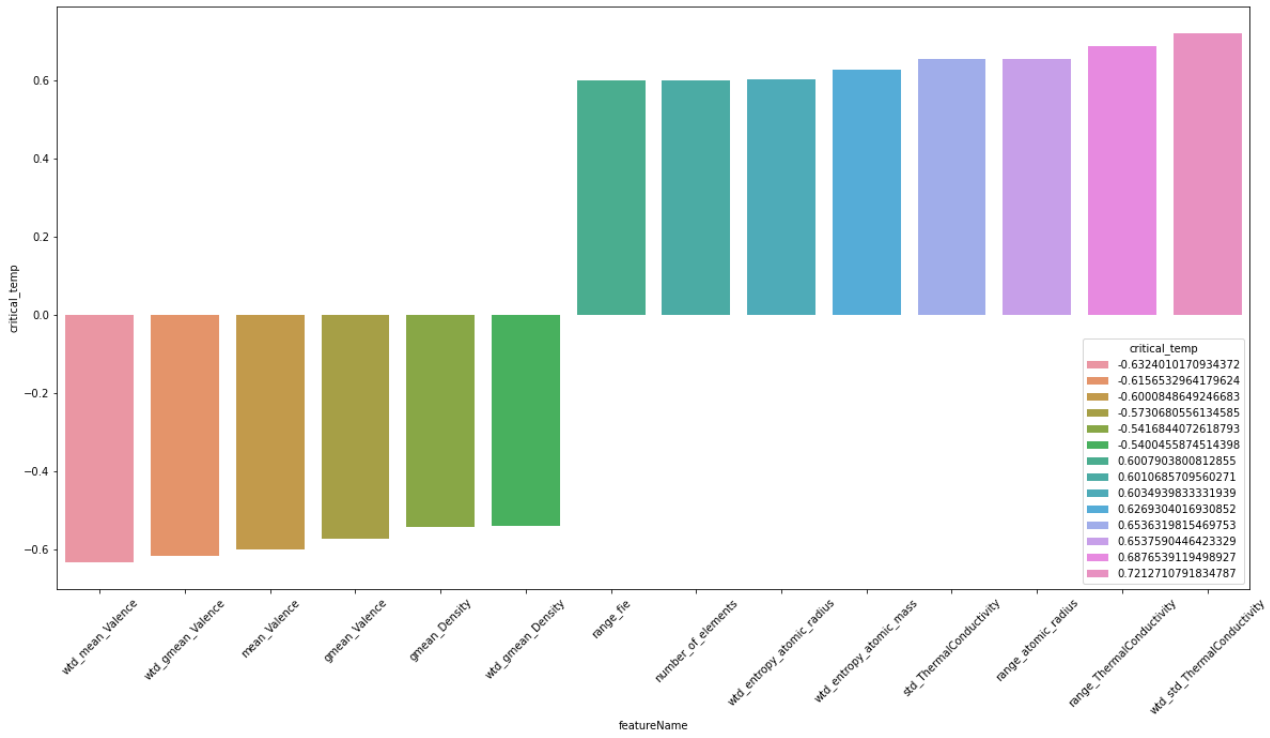
In [17]:
```
corr_df
```

Out[17]:

featureName	critical_temp
wtd_mean_Valence	-0.632401
wtd_gmean_Valence	-0.615653
mean_Valence	-0.600085
gmean_Valence	-0.573068

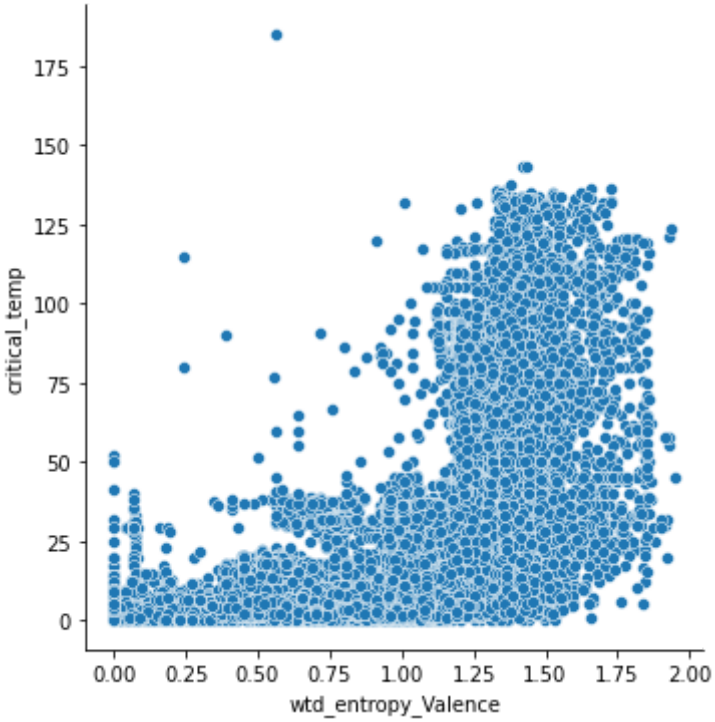
featureName	critical_temp
gmean_Density	-0.541684
wtd_gmean_Density	-0.540046
range_fie	0.600790
number_of_elements	0.601069
wtd_entropy_atomic_radius	0.603494
wtd_entropy_atomic_mass	0.626930
std_ThermalConductivity	0.653632
range_atomic_radius	0.653759
range_ThermalConductivity	0.687654
wtd_std_ThermalConductivity	0.721271

In [18]:
```
# Visualize correlations between target features and other variables
fig, ax = plt.subplots(figsize =(20, 10))
plt.xticks(rotation = 45)
ax = sns.barplot(x=corr_df.index, y="critical_temp", hue="critical_temp",
                 data=corr_df, dodge=False)
```



In [19]:
```
# We can see non linear relationships
sns.relplot(data=reaserch_df, x='wtd_entropy_Valence', y='critical_temp')
```

Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd8d36e1d30>
```



Train Test Split

In [20]:

```
# Define X dataframe containing features
X = reaserch_df.drop(['critical_temp'], axis=1 )

cols = X.columns

# Show examples of non linear relationsips: quadratic
for i in range(len(cols)):
    name = cols[i] + "^2"
    X[name] = X[cols[i]]*X[cols[i]]

y = reaserch_df["critical_temp"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Scaling the data
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(data = X_train_scaled, columns = X_train.columns)


X_test_scaled = scaler.fit_transform(X_test)
X_test_scaled = pd.DataFrame(data = X_test_scaled, columns = X_train.columns)
```

In [21]:

```
# Checking the scaled data
X_train_scaled.describe()
```

Out[21]:

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	w
count	1.701000e+04	1.701000e+04	1.701000e+04	1.701000e+04	

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	v
mean	1.799070e-16	5.532840e-17	5.153628e-16	-2.074440e-16	
std	1.000029e+00	1.000029e+00	1.000029e+00	1.000029e+00	
min	-2.171252e+00	-2.717471e+00	-1.992712e+00	-2.128630e+00	
25%	-7.746026e-01	-4.991540e-01	-6.225372e-01	-4.263353e-01	
50%	-7.627791e-02	-9.410265e-02	-3.647041e-01	-1.576794e-01	
75%	6.220467e-01	4.342271e-01	3.960063e-01	2.123406e-01	
max	3.415345e+00	4.097789e+00	4.071989e+00	4.447332e+00	

8 rows × 316 columns

L1 Regression

In [22]:

```
# L1 Lasso Regression to see error metrics, not for parameter tuning (we will no
alpha = 1
cv =  KFold(n_splits=5, shuffle=True, random_state=2022) # 5-fold cross validati

l1_model = Lasso() # Initialize model

df = pd.DataFrame(columns = ['lamda', 'neg_mean_squared_error'])

# Loop through different alphas to see resulting error metrics
for i in range(20):
    l1_model.alpha = alpha

    scores = cross_val_score(l1_model, X_train_scaled, y_train, cv=cv, scoring=
    print("aplha =", alpha, "CVScore =", scores)
    df = df.append({'lamda' : alpha, 'neg_mean_squared_error' : scores.mean()},
    print("----------")
    alpha = alpha / 1.2
```

```
aplha = 1 CVScore = [-330.64352601 -332.25598601 -341.397675   -345.23118195 -32
2.96643137]
----------
aplha = 0.8333333333333334 CVScore = [-323.48455889 -323.71799727 -335.23008594
-340.32486546 -315.05606447]
----------
aplha = 0.6944444444444445 CVScore = [-315.7470289  -315.21892235 -326.84404494
-336.88306407 -306.87468468]
----------
aplha = 0.5787037037037038 CVScore = [-308.55954094 -307.04958501 -316.95842859
-334.95188647 -299.54284167]
----------
aplha = 0.48225308641975323 CVScore = [-302.93391277 -299.86140428 -308.44558983
-334.95069698 -293.43213188]
----------
aplha = 0.401877572016461 CVScore = [-298.35889229 -294.14369987 -302.00175391 -
334.67767102 -288.49691357]
```

```
----------
aplha = 0.3348979766803842 CVScore = [-293.95822803 -288.83233163 -289.81041301
-332.99382831 -283.1703472 ]
----------
aplha = 0.2790816472336535 CVScore = [-289.53749764 -284.66907708 -296.83675994
-332.55160339 -278.65565626]
----------
aplha = 0.2325680393613779 CVScore = [-286.19112483 -280.96829683 -316.14765586
-332.58540056 -275.68866936]
----------
aplha = 0.19380669946781492 CVScore = [-283.07962051 -276.94529841 -342.70202278
-352.65781175 -273.28620241]
----------
aplha = 0.1615055828898458 CVScore = [-280.09825088 -273.23818195 -376.54797572
-418.85375274 -272.08044481]
----------
aplha = 0.13458798574153816 CVScore = [-277.48225123 -269.93233728 -408.25172793
-537.04328566 -271.94696327]
----------
aplha = 0.11215665478461513 CVScore = [-275.43111819 -267.37750944 -431.29993564
-669.55583706 -272.53527713]
----------
aplha = 0.09346387898717928 CVScore = [-273.57245193 -265.34550954 -457.00513328
-804.61797499 -273.01480231]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4568.97809033468
4, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 2409.08156778663
4, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 3861.38891810877
25, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 2026.43923660181
46, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
aplha = 0.07788656582264941 CVScore = [-270.57923783 -261.63949658 -469.39887859
-960.34834621 -273.24658192]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 23520.4523628670
72, tolerance: 1596.4480642417748
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 25009.4841559990
3, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 29800.2451483588
67, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
```

```
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 11804.5363515652
72, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 45499.8431312199
7, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
aplha = 0.06490547151887452 CVScore = [ -266.67920536  -257.91364593  -469.59057
606 -1133.41239858
  -273.11196698]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 57391.7228216151
7, tolerance: 1596.4480642417748
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 62688.876281297
3, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 78176.9586759316
7, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 65879.0699772648
5, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 69713.2491812473
2, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
aplha = 0.05408789293239544 CVScore = [ -263.15393202  -254.84361138  -461.94838
35  -1339.44458193
  -271.06593434]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 152217.553509692
2, tolerance: 1596.4480642417748
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 112085.509696892
93, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 126291.268111913
93, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 58807.0219750194
8, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
```

```
u might want to increase the number of iterations. Duality gap: 149476.221838064
96, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
aplha = 0.045073244110329536 CVScore = [ -260.22381264  -252.87179948  -462.6294
7452 -1537.61254579
  -269.53905165]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 304457.976148308
5, tolerance: 1596.4480642417748
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 243057.862589487
8, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 286135.293658701
94, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 239562.602589483
37, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 267913.904173745
8, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
aplha = 0.037561036758607946 CVScore = [ -257.57980099  -259.30159883  -465.2883
618  -1650.15720519
  -270.14653231]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 370946.74267204
9, tolerance: 1596.4480642417748
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 311089.48916050
7, tolerance: 1581.420437841415
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 306918.94362379
1, tolerance: 1587.4616116499983
  model = cd_fast.enet_coordinate_descent(
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 322326.288190980
9, tolerance: 1574.2352415718863
  model = cd_fast.enet_coordinate_descent(
aplha = 0.031300863965506624 CVScore = [ -255.56118641  -278.92939685  -465.3358
5783 -1733.66028736
  -271.7171086 ]
----------
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 331034.317950745
```

```
8, tolerance: 1586.953294930031
  model = cd_fast.enet_coordinate_descent(
```

In [23]:

```
# Sort by value
df.sort_values('neg_mean_squared_error', ascending=False)
```

Out[23]:

	lamda	neg_mean_squared_error
7	0.279082	-296.450119
6	0.334898	-297.753030
8	0.232568	-298.316229
5	0.401878	-303.535786
9	0.193807	-305.734191
4	0.482253	-307.924747
3	0.578704	-313.412457
2	0.694444	-320.313549
10	0.161506	-324.163721
1	0.833333	-327.562714
0	1.000000	-334.498960
11	0.134588	-352.931313
12	0.112157	-383.239935
13	0.093464	-414.711174
14	0.077887	-447.042508
15	0.064905	-480.141559
16	0.054088	-518.091289
17	0.045073	-556.575337
18	0.037561	-580.494700
19	0.031301	-601.040767

In [24]:

```
# L1 lasso parameter tuning and selection with RandomizedSearchCV

l1_alpha_estimators = [float(x) for x in np.linspace(start = 0.0, stop = 1.0, nu

l1_model = Lasso()
split = KFold(n_splits = 7, shuffle = True)

l1_param_grid = {
    "alpha": l1_alpha_estimators
}

L1_cv = RandomizedSearchCV(
    l1_model, l1_param_grid, n_iter=200, cv=split, n_jobs=-1, scoring='neg_mean_
)
```

In [25]:
```
L1_cv.fit(X_train_scaled, y_train) # Fit to train data
```

Out[25]:
```
RandomizedSearchCV(cv=KFold(n_splits=7, random_state=None, shuffle=True),
                   estimator=Lasso(), n_iter=200, n_jobs=-1,
                   param_distributions={'alpha': [0.0, 0.001001001001001001,
                                                  0.002002002002002002,
                                                  0.003003003003003003,
                                                  0.004004004004004004,
                                                  0.005005005005005005,
                                                  0.006006006006006006,
                                                  0.007007007007007007,
                                                  0.008008008008008008,
                                                  0.009009009009009009,
                                                  0.010010010010010...
                                                  0.015015015015015015,
                                                  0.016016016016016016,
                                                  0.017017017017017015,
                                                  0.018018018018018018,
                                                  0.01901901901901902,
                                                  0.02002002002002002,
                                                  0.02102102102102102,
                                                  0.022022022022022022,
                                                  0.023023023023023025,
                                                  0.024024024024024024,
                                                  0.025025025025025023,
                                                  0.026026026026026026,
                                                  0.02702702702702703,
                                                  0.028028028028028028,
                                                  0.029029029029029027, ...]},
                   scoring='neg_mean_squared_error')
```

In [26]:
```
# Best alpha
L1_cv.best_estimator_
```

Out[26]:
```
Lasso(alpha=0.2032032032032032)
```

In [27]:
```
# Create final L1 lasso model with best alpha
feature_cols = X_train.columns
l1_FinalModel = Lasso(alpha = .20) #1
l1_FinalModel.fit(X_train_scaled, y_train)
l1_features_list = list(zip(feature_cols, l1_FinalModel.coef_))
l1_features_df = pd.DataFrame(l1_features_list, columns = ['Feature Name', 'Coef
```

In [28]:
```
# Get train RMSE to see how well model performs on train data
l1_y_pred_train = l1_FinalModel.predict(X_train_scaled)
l1_RMSE_train = np.sqrt(metrics.mean_squared_error(y_train, l1_y_pred_train))
l1_RMSE_train
```

Out[28]:
```
16.6373624833527
```

In [29]:
```
# Highest magnitude coefficients from l1 lasso model
l1_features_df['Coef_abs'] = l1_features_df['Coef'].abs()
l1_features_df.sort_values('Coef_abs', ascending=False).head(10)
```

Out[29]:

	Feature Name	Coef	Coef_abs

	Feature Name	Coef	Coef_abs
131	Ba	11.174418	11.174418
228	wtd_std_ThermalConductivity^2	10.211003	10.211003
97	Ca	9.407741	9.407741
165	range_atomic_mass^2	8.597100	8.597100
255	Ca^2	-5.778839	5.778839
168	wtd_std_atomic_mass^2	-4.063470	4.063470
157	Bi	3.793714	3.793714
80	wtd_std_Valence	-3.283155	3.283155
62	wtd_mean_ThermalConductivity	2.930476	2.930476
268	As^2	-2.758165	2.758165

L2 Regression

In [30]:
```
# L2 ridge regression parameter tuning with RandomizedSearchCV
L2_model = Ridge()

l2_alpha_estimators = [int(x) for x in np.linspace(start = 0.0, stop = 1000.0, n

l2_param_grid = {
    "alpha": l2_alpha_estimators
}

L2_cv = RandomizedSearchCV(
    L2_model, l2_param_grid, n_iter=200, cv=split, n_jobs=-1, scoring='neg_mean_
)

L2_cv.fit(X_train_scaled, y_train)
```

Out[30]:
```
RandomizedSearchCV(cv=KFold(n_splits=7, random_state=None, shuffle=True),
                   estimator=Ridge(), n_iter=200, n_jobs=-1,
                   param_distributions={'alpha': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                  10, 11, 12, 13, 14, 15, 16,
                                                  17, 18, 19, 20, 21, 22, 23,
                                                  24, 25, 26, 27, 28, 29, ...]},
                   scoring='neg_mean_squared_error')
```

In [31]:
```
# Best alpha for ridge
L2_cv.best_estimator_
```

Out[31]:
```
Ridge(alpha=993)
```

In [32]:
```
# Create ridge model with best alpha
l2_FinalModel = Ridge(alpha = 993) #1
l2_FinalModel.fit(X_train_scaled, y_train)
l2_features_list = list(zip(feature_cols, l2_FinalModel.coef_))
l2_features_df = pd.DataFrame(l2_features_list, columns = ['Feature Name', 'Coef
```

In [33]:
```
# Most important features from ridge model
l2_features_df['Coef_abs'] = l2_features_df['Coef'].abs()
l2_features_df.sort_values('Coef_abs', ascending=False).head(10)
```

Out[33]:

	Feature Name	Coef	Coef_abs
131	Ba	7.768679	7.768679
97	Ca	6.084711	6.084711
228	wtd_std_ThermalConductivity^2	4.289562	4.289562
157	Bi	4.242535	4.242535
165	range_atomic_mass^2	4.166527	4.166527
62	wtd_mean_ThermalConductivity	4.027410	4.027410
255	Ca^2	-3.569985	3.569985
80	wtd_std_Valence	-3.505065	3.505065
168	wtd_std_atomic_mass^2	-3.104680	3.104680
70	wtd_std_ThermalConductivity	2.852972	2.852972

Test Model evaluation

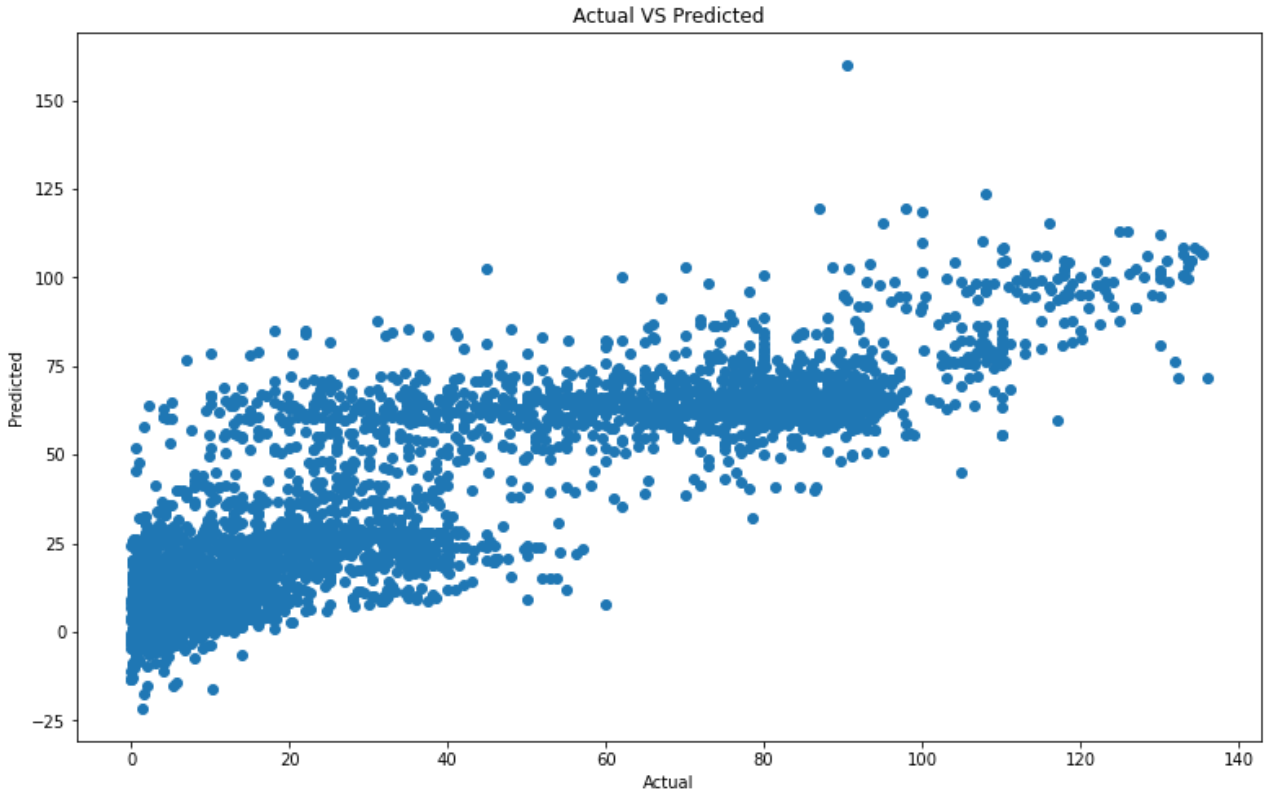
In [34]:
```
# Function to create actual vs predicted graph
def actual_vs_predicted(actual, predicted):
    fig, ax = plt.subplots(figsize=(13, 8))
    ax.scatter(actual, predicted)
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')
    plt.title('Actual VS Predicted')
    plt.show()

# Function to show residuals from model
def residuals(actual, predicted):
    fig, ax = plt.subplots(figsize=(13, 8))
    ax.scatter(actual, actual - predicted)
    ax.set_xlabel('Actual')
    ax.set_ylabel('Residuals')
    plt.title('Residuals')
    plt.show()
```

In [35]:
```
# Produce l1 lasso predictions from final model
l1_y_pred = l1_FinalModel.predict(X_test_scaled)
l1_RMSE = np.sqrt(metrics.mean_squared_error(y_test, l1_y_pred))
```

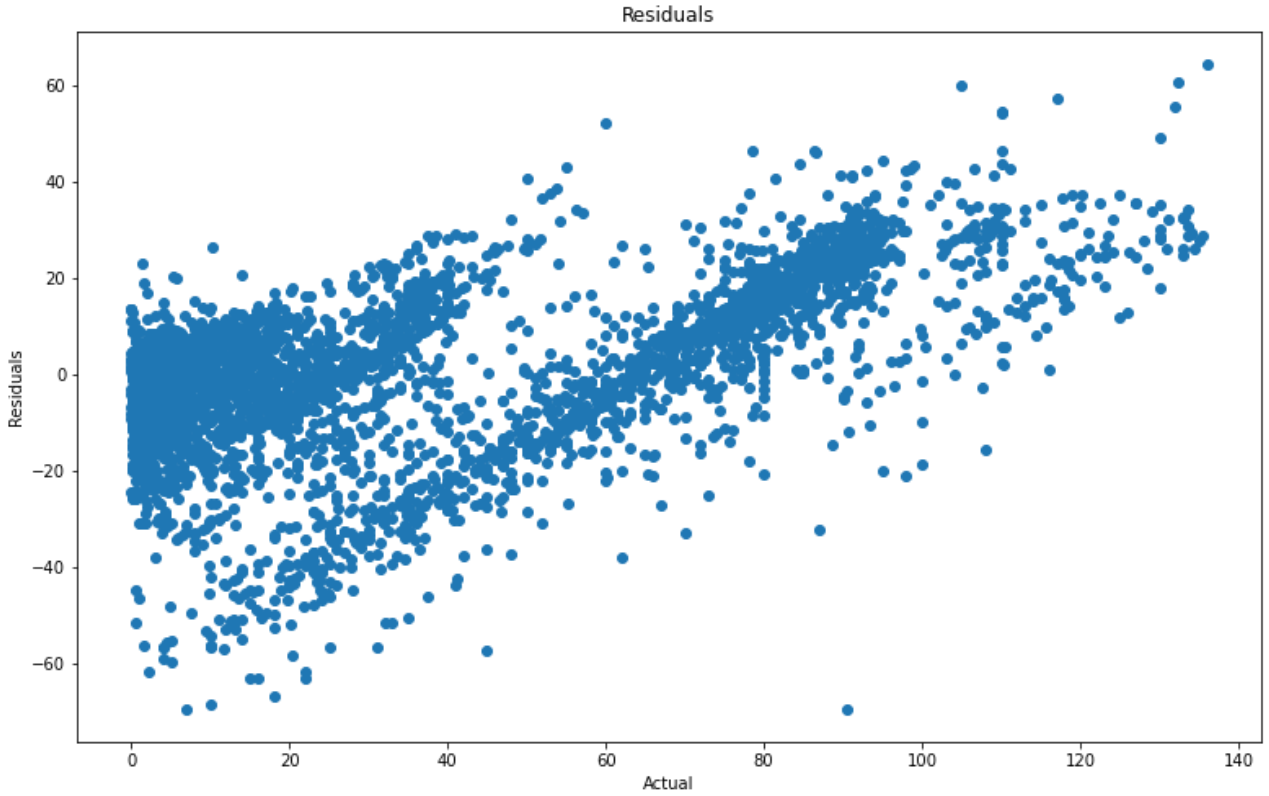
In [36]:
```
# Get l1 lasso RMSE
l1_RMSE
```

Out[36]:
```
16.980162793503002
```

In [37]:
```
# L1 lasso model actual vs predicted graph
actual_vs_predicted(y_test, l1_y_pred)
```



In [38]:
```
# L1 lasso residuals
residuals(y_test, l1_y_pred)
```

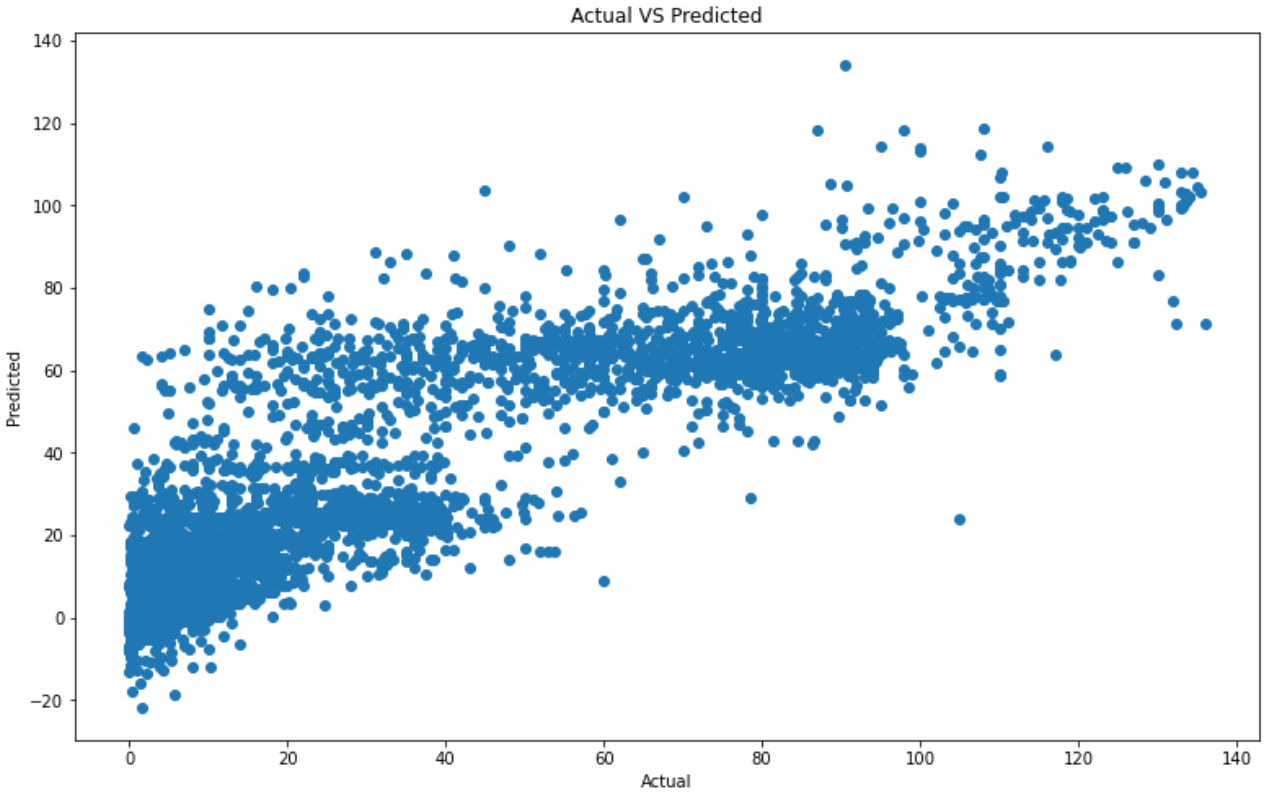


In [39]:

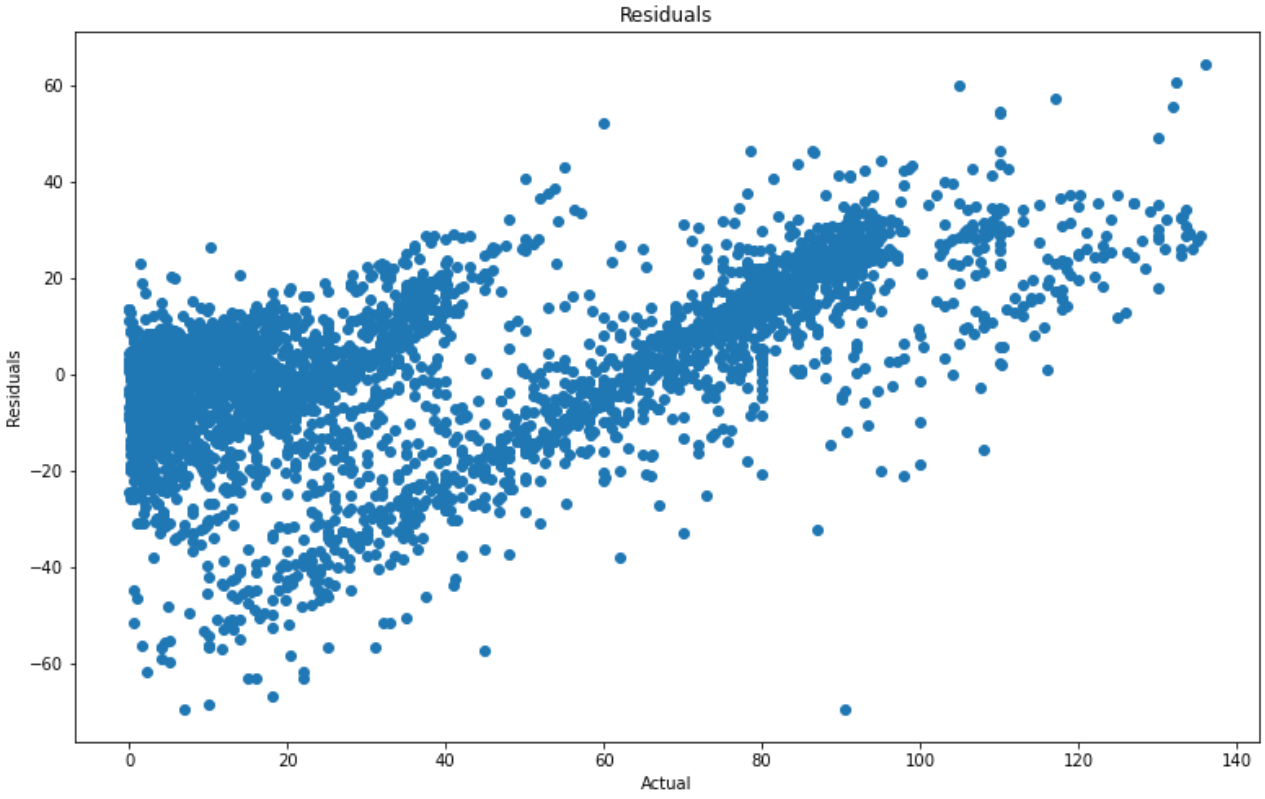
```
# Predictions for l2 ridge mdoel
l2_y_pred = l2_FinalModel.predict(X_test_scaled)
l2_RMSE = np.sqrt(metrics.mean_squared_error(y_test, l2_y_pred))
l2_RMSE # RMSE
```

Out[39]: 16.599845405633577

In [40]:
```
# Ridge actual vs predicted graph
actual_vs_predicted(y_test, l2_y_pred)
```



In [41]:
```
# Residual graph for l2 ridge model
residuals(y_test, l1_y_pred)
```



In [43]:

```
# Running linear regression to compare lasso and ridge results
LinearModel = LinearRegression()
LinearModel.fit(X_train_scaled, y_train)
linear_y_pred = LinearModel.predict(X_test_scaled)
linear_RMSE = np.sqrt(metrics.mean_squared_error(y_test, linear_y_pred))
linear_RMSE
```

Out[43]: 17.897873553168253