Predicting Critical Temperature of Superconductors using Principal Component Regression

By:

Joshua Fontes

Abstract

The focus of this paper is to explain and demonstrate a regression technique called principal component regression (PCR). This regression procedure is based on the dimensionality reduction technique called principal component analysis (PCA). PCR is a moderately advanced but commonly used regression technique for high multicollinearity datasets. This paper first briefly outlines the background, purpose, and mathematical assumptions needed for PCA and PCR. Then a demonstration of the PCR technique is shown using the superconductivity dataset from the UCI machine learning repository. The demonstration extends through many first steps of a regression analysis to give appropriate context of the PCR method, including exploratory data analysis, the model building process, interpretations in the context of the dataset, as well as a residual analysis. The results of the analysis are then discussed to examine the pros and cons of PCR and to discuss future considerations for prospective PCR model builders. At least a moderate understanding of linear algebra and multiple linear regression is needed to get the most out of reading this paper. All of the data cleaning and statistical analyses done for this paper was completed in Python using several packages including *Pandas*, *NumPy*, *Matplotlib*, *Seaborn*, *Scikit-Learn*, and *SciPy*. Some of the data visualizations were made in the statistical software R, using the packages *ggplot2* and *plot_ly*.

Introduction

Background

Regression is statistical technique used for modeling and describing relationships between variables. For example, one main goal of regression is to make predictions on a particular response variable by building a regression model using predictor variables. Regression is one of the most commonly used statistical techniques because it can be effectively applied in many fields of study and because of its versatility. There are many different types of regression models that are suited for different types of problems. One major problem that can arise in a regression analysis is multicollinearity. One major regression technique that is used to combat multicollinearity is called principal component regression (PCR).

The focus of this paper is to introduce and detail the PCR method. This regression procedure is based on principal component analysis (PCA), which is a dimensionality reduction technique. First, this paper briefly explains the context of the problem where PCR can be useful. Then, the mathematical assumptions needed for PCA and PCR are outlined. Next, this paper demonstrates a concise analysis using the PCR technique on the superconductivity dataset from the UCI machine learning repository. Then, the results of the analysis will be discussed to highlight the advantages and disadvantages of using PCR. Lastly, any remaining questions and considerations are addressed to help any other potential users of PCR. All of the data cleaning and statistical analyses done for this paper were completed in Python using several packages including *Pandas*, *NumPy*, *Matplotlib*, *Seaborn*, *Scikit-Learn*, and *SciPy*. Some of the data visualizations were made in the statistical software R, using the packages *ggplot2* and *plot_ly*.

Purpose and Mathematical Assumptions

In general, PCR is a regression technique and so it follows all the usual assumptions for ordinary least squares regression analysis, including independent and identically distributed observations and residuals with constant variance that are also normally distributed. Furthermore, an important assumption of regression is that the regressor variables are independent of each other so that the effects of individual predictor variables can be estimated without any unclear influence from other predictors. However, high multicollinearity between predictor variables is a common problem in real-world applications of regression. This directly causes problems for model building in regression because high multicollinearity inflates the variance of the regression coefficients. Consider the ordinary least squares regression model, $y = \beta X + \epsilon$, where X is the data matrix of predictor variables (design matrix) and the coefficients in our model are found by $\beta = (X'X)^{-1}X'y$. When the predictors are nearly linearly dependent, $X'X$ becomes ill-conditioned which results in very large numerical error. So, in theory, if a researcher collected more data to build a similar model in a similar study, there could be wildly different results and conclusions made between the two models. One method that can be used in this situation to mediate high multicollinearity is PCR.

PCR Outline and the Mathematical Process

The general outline of the PCR procedure is to do PCA on the centered design matrix of the dataset and then use only a subset of the principal components as the new predictor variables for the regression model. PCA is a statistical technique used to transform a set of variables into a set of linearly uncorrelated variables called principal components. The principal components represent the directions of maximum variance of the dataset. The goal with PCA is to subset the principal components to preserve as much variance as possible while reducing dimensionality of the dataset. In the context of regression, this reduces multicollinearity.

The PCR procedure begins with conducting PCA on the design matrix for your regression model. This transformation is applied to the centered design matrix of the dataset which is found by subtracting each column of the design matrix by their respective means, $\bar{X} = X - \bar{x}$. The PCA transformation is done by using the singular value decomposition (SVD) of the centered design matrix. The centered design matrix is decomposed into three matrices defined by the eigenvectors of $\bar{X}\bar{X}'$ (columns of U), the singular values of $\bar{X}$ (diagonal entries of Σ), and the eigenvectors of $\bar{X}'\bar{X}$ (columns of V) to form the SVD of the centered design matrix, $\bar{X} = U\Sigma V'$. The principal components, denoted by Z, are then found by multiplying the centered design matrix with the right singular vectors, $Z_{k\times r} = \bar{X}_{k\times p}V_{p\times r}$ where *k* denotes the number of observations in the dataset, *p* denotes the number of predictor variables, and *r* denotes the number of first principal components used in the subset. The chosen subset of the principal components is used to fit an ordinary least squares regression model, $y = Z\beta_z + \epsilon$ where the coefficients in the model are found by $\beta_z = (Z'Z)^{-1}Z'y$. The coefficients in this regression model can be transformed back into the units of the original predictors with $\beta_x = V\beta_z$. The whole mathematical process and model formulation is outlined in the following graphic (figure 1).

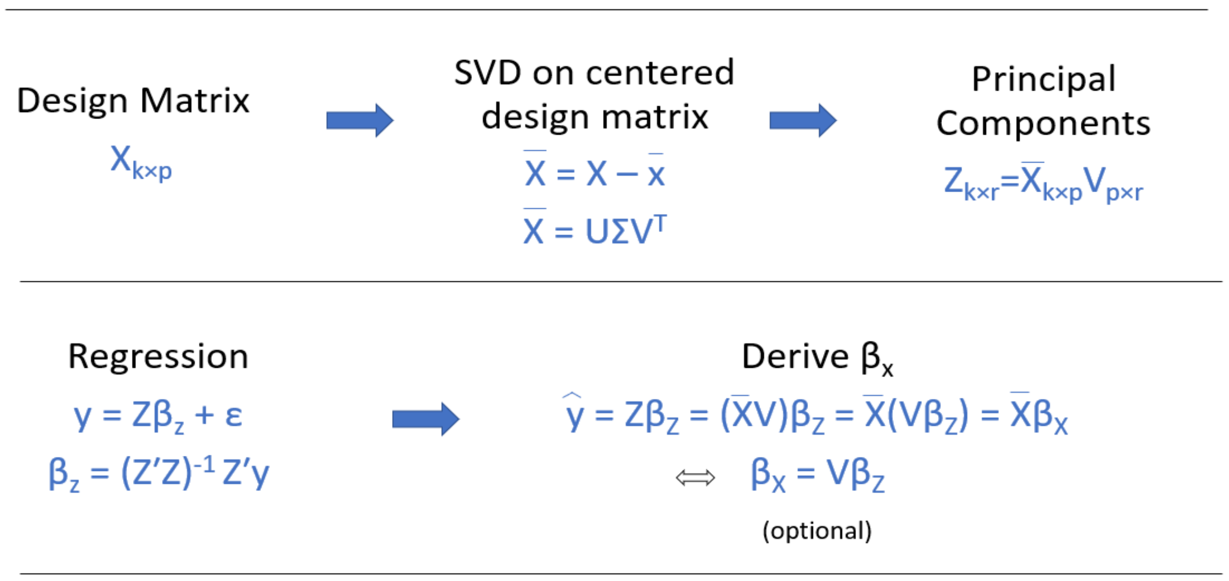

Design Matrix
$X_{k\times p}$
SVD on centered design matrix
$\bar{X} = X - \bar{x}$
$\bar{X} = U\Sigma V^T$
Principal Components
$Z_{k\times r}=\bar{X}_{k\times p}V_{p\times r}$
Regression
$y = Z\beta_z + \varepsilon$
$\beta_z = (Z'Z)^{-1}Z'y$
Derive β_X
$\hat{y} = Z\beta_Z = (\bar{X}V)\beta_Z = \bar{X}(V\beta_Z) = \bar{X}\beta_X$
$\Leftrightarrow \beta_X = V\beta_Z$
(optional)

Figure 1: The graphic above shows the PCR procedure summarized in a step-by-step process using mathematical notation. The last step labeled 'optional' is only needed if the coefficients in the model are needed to be interpreted in the context and units of the original standardized variables.

Methods

Data Description and Setup

The dataset used in the following analysis is the superconductivity dataset obtained from the UCI machine learning repository. This dataset is a good example to demonstrate PCR because it is a very high dimension dataset with mainly quantitative variables that have high multicollinearity. The data contain the main dataset used for prediction which has the critical temperature and 81 other variables on 21,263 superconductors, resulting in 82 columns and 21,263 rows. A second dataset contains information on the element composition of the superconductors, resulting in a dataset with 87 columns, representing the elements of the periodic table hydrogen through radon, and the 21,263 superconductor observations organized as rows. The goal of the analysis is to predict the critical temperature (T_c) of a superconductor, measured in Kelvin, using 81 features based on properties of the elemental composition of each superconductor material. The property features were extracted from the observed superconductors. The elemental properties are summarized in the table below (table 1). For this paper, the PCR model was built using the features dataset and excluding the elements dataset.

Property	**Units**
Atomic Mass	Atomic mass units (AMU)
First Ionization Energy	Kilo-Joules per mole (kJ/mol)
Atomic Radius	Picometer (pm)
Density	Kilograms per meters cubed (kg/m^3)
Electron Affinity	Kilo-Joules per mole (kJ/mol)
Fusion Heat	Kilo-Joules per mole(kJ/mol)
Thermal Conductivity	Watts per meter-Kelvin (W/(m K))
Valence	No units (integer)

Table 1: The table shows the main elemental property in the dataset. The table excludes the property features extracted from each superconductor (used as predictors), which include geometric mean, range, and standard deviation. A more detail explanation of the variables can be found in the relevant scientific paper of the dataset.

Exploratory Data Analysis

A major point for the exploratory data analysis section of the dataset was to detect multicollinearity in the data. Graphical representations of multicollinearity are shown in the form of a correlation matrix heatmap (figure 2) and a variance inflation factor (VIF) table (table 2).

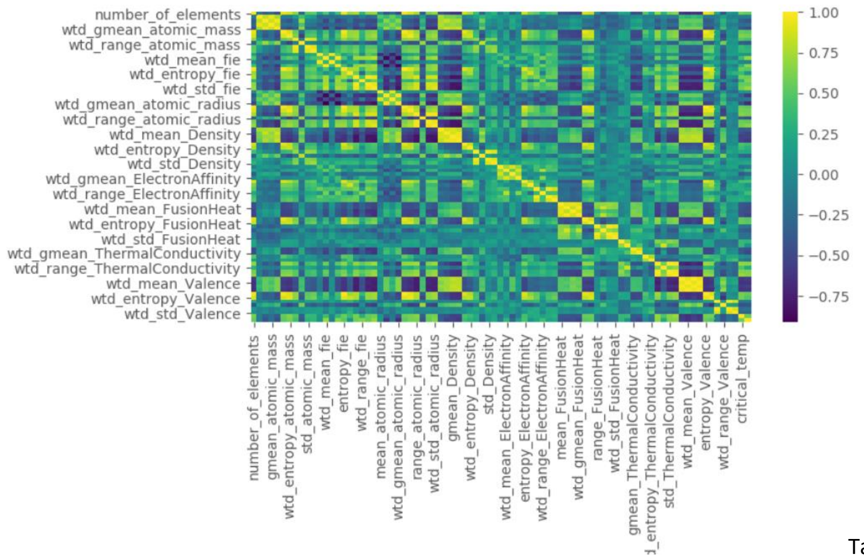
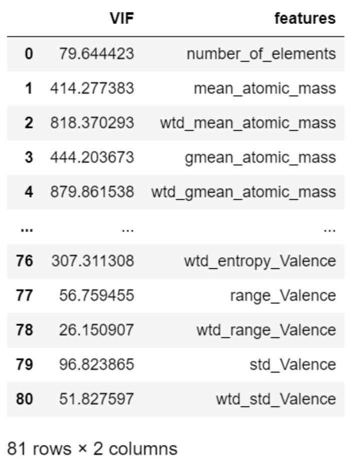


Figure 2: Correlation matrix heatmap of the data shows very high correlation between several variables.



	VIF	features
0	79.644423	number_of_elements
1	414.277383	mean_atomic_mass
2	818.370293	wtd_mean_atomic_mass
3	444.203673	gmean_atomic_mass
4	879.861538	wtd_gmean_atomic_mass
...	...	...
76	307.311308	wtd_entropy_Valence
77	56.759455	range_Valence
78	26.150907	wtd_range_Valence
79	96.823865	std_Valence
80	51.827597	wtd_std_Valence

81 rows × 2 columns

Table 2: VIF table of the predictor variables shows extremely high multicollinearity for multiple predictor variables.

The correlation matrix heatmap shows very high correlation between several variables such as the geometric mean and weighted mean features for atomic mass. The VIF table shows extremely high multicollinearity between many predictors, since virtually all features had a VIF much higher than 10. This shows the degree of multicollinearity in the data which assuredly would produce an ordinary least squares regression model that cannot be reliably interpreted in any significant way. The distribution of the response variable can be observed in the histogram constructed below (figure 3).

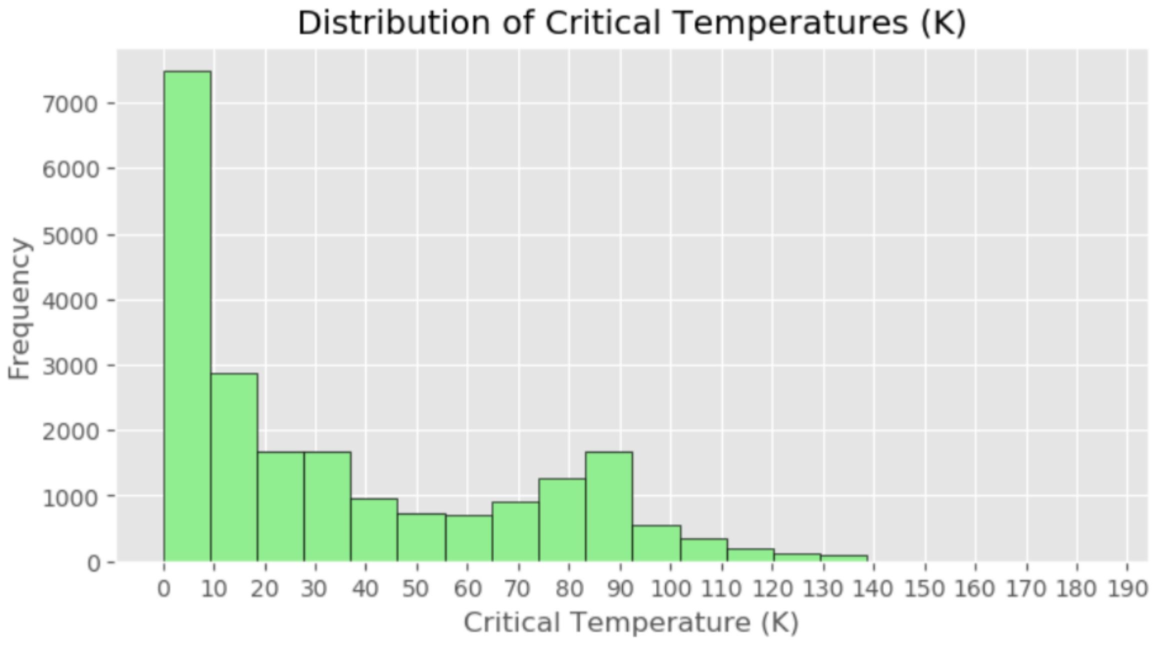


Figure 3: The histogram roughly shows the distribution of the response variable, critical temperature. The distribution is clearly right skewed since most of the critical temperatures are in between 0 K and 10 K.

The distribution of critical temperatures is clearly right skewed, indicating that most superconductors have a critical temperature that falls in between 0 K and 10 K. Furthermore, scatterplots between each predictor variable and critical temperature were made to investigate the individual relationships in the data, found in the appendix (figure 4). Many of the individual scatterplots did not clearly show a linear relationship which suggests that transformations on either the response variables or any of the predictor variables could be necessary in a full regression analysis. PCA score plots were also made to explore differences between superconductors with and without certain elements, where each element was column bind to the principal component matrix as a factor variable to make the plots. Several elements were examined but oxygen was of particular interest because the PCA score plot showed a considerable amount of separation between superconductors with oxygen and those without. This motivated building a second PCR model using oxygen content as an indicator variable. The PCA score plots for oxygen can be found in the appendix (figures 5, 6).

Model Building

The PCR model was built by doing PCA on the standardized design matrix, as opposed to the centered design matrix. This can be beneficial because standardization eliminates the influence of wildly different units on the principal components. Subsetting the principal components should be done with preserving as much variance as possible, which can be determined in a few different ways. One method is to calculate the cumulative explained variance by each subsequent principal component. The explained variance for each principal component is found by $\frac{\lambda_m}{\sum \lambda_i}$ where λ_m represents the m^{th} eigenvalue of the correlation matrix of the standardized design matrix. Another method is to keep only the principal components who have a large corresponding eigenvalue (usually those with $\lambda_m > 1$). A cumulative explained variance plot and a scree plot can both be found in the appendix of this paper (figures 7, 8). These methods show that approximately 99% of the variance in the original dataset can be explained by the first 30 principal components. Consequently, the PCR model was built by fitting the first 30 principal components against the response T_c (model 1).

To explore using categorical variables with PCR, oxygen content was used as an indicator variable. To do this, the data was organized so that the features of all 9299 superconductors that did not contain oxygen were stacked on top of the features of all 11964 superconductors that did contain oxygen. PCA was then done to this data matrix. Then a new factorized variable was created where each superconductor that contained no oxygen was assigned a '0' while each superconductor that had any recorded amount of oxygen in them was assigned a '1'. This factorized variable was then combined with the principal components accordingly (after PCA was done on the quantitative variables) to build a second PCR model by fitting the first 30 principal components with the indicator variable oxygen against the response T_c (model 2).

Generalized cross-validation (10-fold cross-validation) measures, including the coefficient of determination (R^2) and mean squared error (MSE), were used as measures to assess how well each model predicted the observed values. The appendix also contains a graphical display for each regression model to help visualize the predictive performance of each model (figures 9, 12) and residual plots for each model to see if regression residual assumptions were satisfied (figures 10, 11, 13, 14).

Results

The cross-validation summary results for each model are displayed below (table 3). Model 1 produced an MSE of 393.389 and an R^2 of 0.665. Model 2 produced an MSE of 403.463 and an R^2 of 0.656. The cross-validation measures for both models were quite similar and are not terrible results. Although not great, the cross-validation measures indicate two decent models for predictions. A visual display of the regression lines for model 2 can be seen on the next page (figure 12).

Metrics	**Model 1**	**Model 2**
MSE	393.386	403.463
R^2	0.665	0.656

Table 3: The table shows the cross-validation results for each model. Model 1 refers to fitting the response T_c against the first 30 principal components. Model 2 refers to fitting the response T_c against the first 30 principal components and with an indicator variable for oxygen content

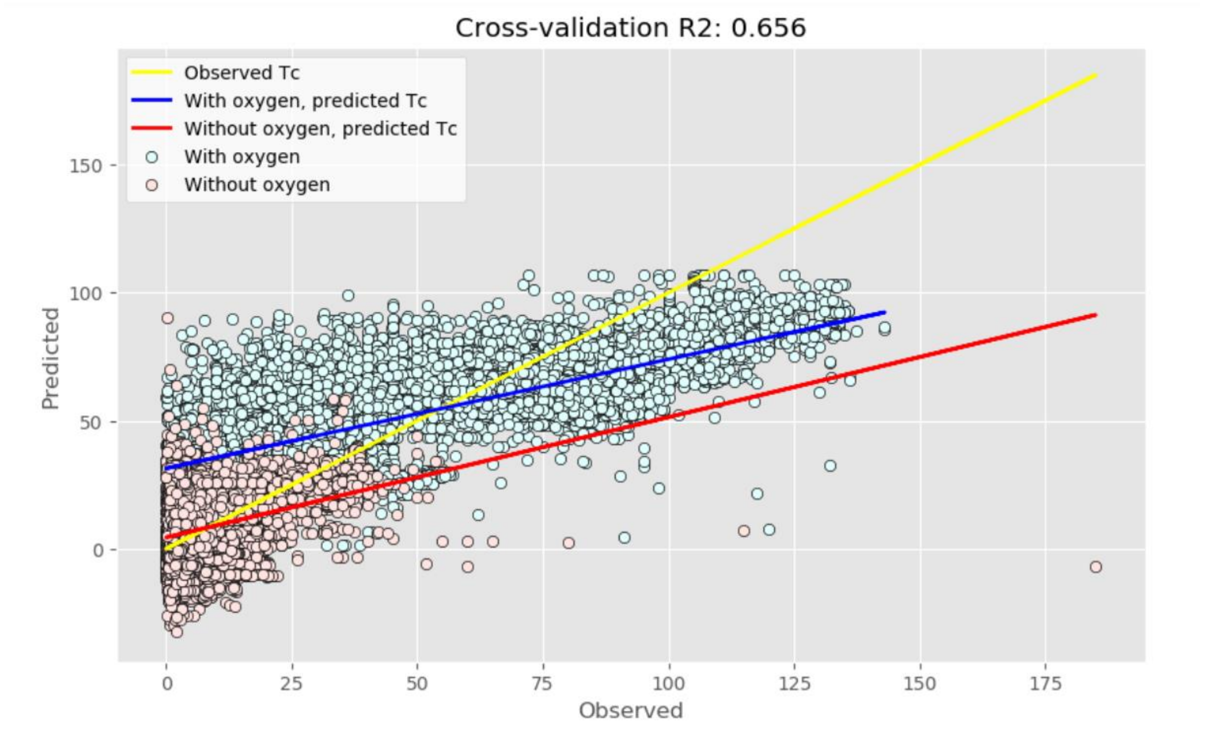


Figure 12: The above scatterplot shows a visual display of the regression lines for model 2. The yellow line represents the observed T_c values plotted against each other. The blue line represents the predicted T_c values for superconductors with oxygen while the red line represents the predicted T_c values for superconductors without oxygen. The blue dots represent predicted T_c values for superconductors with oxygen while the red dots represent predicted T_c values for superconductors without oxygen, both plotted against their corresponding observed T_c values.

Discussion

The Regression Models

The cross-validation results from the two models were not great but showed potential. PCR is still a regression problem so there are many ways to improve the model, including applying transformations to the response or predictors and conducting a residual analysis to detect leverage points or observations of high influence. Furthermore, the entire elements dataset, excluding oxygen, was left out of the model building phase of this analysis. Including more element composition information would likely increase the predictive power of the PCR model.

Pros and Cons of PCR

In summary, there are several advantages and disadvantages to PCR. The main use of PCA in general is as a dimensionality reduction technique, which has many benefits in regression. One clear benefit and the most commonly applied reason for using PCR is to mediate high multicollinearity in the data. This works well for when the researcher wants to include all possible predictor information, ruling out the use of a variable selection algorithm. Another advantage of using PCR is the data visualization features of PCA. PCA is great for data visualization because high dimension datasets can be hard to interpret. PCA allows for data visualization in a reduced dimensional space where directions of maximum variance in the dataset are plotted. This can help to see if there are clear differences between groups and to identify any patterns in the data, as shown in the PCA score plots for oxygen (figures 5, 6). Furthermore, PCA loading plots can be useful in determining the influence of the specific predictor variables on the principal components.

PCR also has disadvantages. A minor disadvantage of PCR is that the model building process is slightly more complicated than most regression methods. However, it is arguably a quite simple and straightforward process with a greater understanding of linear algebra. Another minor disadvantage is the lack of ease with analyzing categorical variables. PCA works with a centered data matrix and it does not make much computational sense to center an indicator variable. In this example, the principal component entries were grouped by an indicator variable after the PCA technique was applied to the quantitative variables.

The main disadvantage of PCR is that the model itself is difficult to interpret. It can be used for predictions but to interpret the coefficients of the model in the context of the original units and predictor variables, more transformations must be applied. Furthermore, PCA is often called a 'plug-and-play' technique because any dataset can be the input and an output will be guaranteed, regardless of the context of the original dataset. This can be seen as a disadvantage because the response variable is completely excluded from the PCA. However, this can also be seen as an advantage because it is a non-parametric technique so the output is unique and independent of the user.

Conclusion

In general, PCR is a useful tool when building a regression model with high multicollinearity. It can be used adjacent to other models such as ridge regression or lasso regression for comparisons across many models. The dimensionality reduction aspect of PCR is also extremely valuable for data simplification and visualization. Future studies should focus on exactly how and when multicollinearity is reduced. Nearly all textbooks and online sources researched for this paper claimed that the multicollinearity is reduced by only using a subset of the principal components for regression analysis. However, none of the sources indicated how large the subset can be while still effectively reducing multicollinearity. For instance, the PCR example shown in this paper used a subset of 30 principal components out of 81 total components to build the regression model. The multicollinearity in the data was assumed to be effectively reduced because the subset removed over 50 principal components, which presumably was a sufficiently large enough reduction to eliminate multicollinearity. Also, it's entirely possible that a clear technique for measuring multicollinearity in the data after subsetting the principal components was simply overlooked during the research portion of writing this paper, and I will revisit all my sources as I continue to work on this paper.

References

"3D Scatter Plots in R", plotly Graphing Libraries, 2020, https://plotly.com/r/3d-scatter-plots/

Alice, Michy, "Performing Principal Components Regression (PCR) in R", Milano R, 2016, http://www.milanor.net/blog/performing-principal-components-regression-pcr-in-r/

Bremer, Martina, "Multicollinearity," Lecture 13, Math 261A, San Jose State University, 2018

Hamidieh, Kam. *A data-driven statistical model for predicting the critical temperature of a superconductor*, Computational Materials Science, Volume 154, November 2018, Pages 346-354

Johnson, Richard, and Dean Wichern. *Applied Multivariate Statistical Analysis,* sixth edition, Pearson Education Inc., 2013.

Kanyongo, Gibbs, *Determining the Correct Number of Principal Components to Extract from a Principal Components Analysis: A Monte Carlo Study of the Accuracy of the Scree Plot*, Journal of Modern Applied Statistical Methods, Vol. 4 Issue 1, Article 13, 2005, https://digitalcommons.wayne.edu/cgi/viewcontent.cgi?article=1166&context=jmasm

Montgomery, Douglas, and Elizabeth Peck, Geoffrey Vining. *Introduction to Linear Regression Analysis*, fifth edition, John Wiley & Sons Inc., 2012

Pelliccia, D. (2019, September 14). Principal Components Regression vs Ridge Regression on NIR data in Python. https://nirpyresearch.com/pcr-vs-ridge-regression-nir-data-python/

Piccolo, Brian, "Plotting PCA Scores and Loadings in Scatterplots," RPubs by RStudio, 2017, https://www.rpubs.com/bpiccolo/pcaplots

"Principal Component Regression", Wikipedia the Free Encyclopedia, 2020, https://en.wikipedia.org/wiki/Principal_component_regression

Shimizu, K., and K. Suhara, M. Ikumo, M. I. Eremets and K. Amaya. "Superconductivity in Oxygen", Nature, 393, 767-769, 1998, https://doi.org/10.1038/31656

Sobolewska, Ewa, "Principal Component Regression," RPubs by RStudio, 2019, https://rpubs.com/esobolewska/pcr-step-by-step

"Superconductors and Superconducting Materials Information", Engineering 360, IEEE GlobalSpec, 2020. https://www.globalspec.com/learnmore/materials_chemicals_adhesives/electrical_optical_specialty_materials/superconductors_superconducting_materials

"Superconductivity Data Data Set", UCI Machine Learning Repository, 2018, https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

Suryanarayana, T.M.V., and P. B Mistry. *Principal Component Regression for Crop Yield Estimation*, Springer Singapore, 2016.

APPENDIX

Contains all figure/tables used in paper and Python/RStudio code used for analysis

Figure 1:

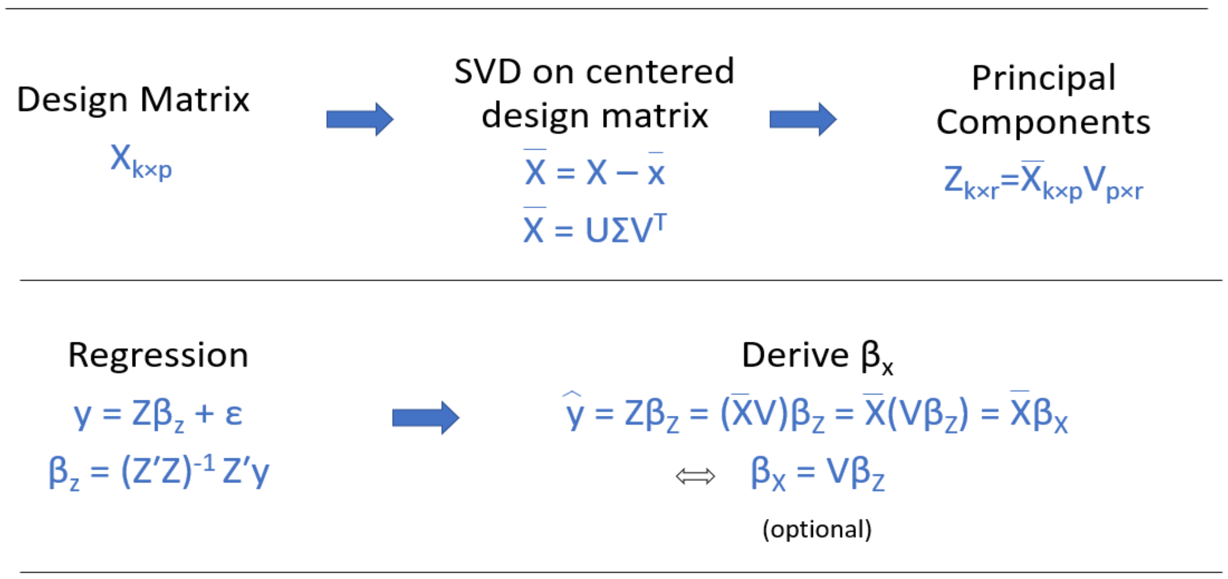

Design Matrix
$X_{k\times p}$
SVD on centered design matrix
$\overline{X} = X - \overline{x}$
$\overline{X} = U\Sigma V^T$
Principal Components
$Z_{k\times r}=\overline{X}_{k\times p}V_{p\times r}$
Regression
$y = Z\beta_z + \varepsilon$
$\beta_z = (Z'Z)^{-1}Z'y$
Derive β_x
$\hat{y} = Z\beta_Z = (\overline{X}V)\beta_Z = \overline{X}(V\beta_Z) = \overline{X}\beta_X$
$\Leftrightarrow \beta_X = V\beta_Z$
(optional)

Figure 1: The graphic above shows the PCR procedure summarized in a step-by-step process using mathematical notation. The last step labeled 'optional' is only needed if the coefficients in the model are needed to be interpreted in the context and units of the original standardized variables (PowerPoint).

Table 1:

Property	Units
Atomic Mass	Atomic mass units (AMU)
First Ionization Energy	Kilo-Joules per mole (kJ/mol)
Atomic Radius	Picometer (pm)
Density	Kilograms per meters cubed (kg/m^3)
Electron Affinity	Kilo-Joules per mole (kJ/mol)
Fusion Heat	Kilo-Joules per mole(kJ/mol)
Thermal Conductivity	Watts per meter-Kelvin (W/(m K))
Valence	No units (integer)

Table 1: The table shows the main elemental property in the dataset. The table excludes the property features extracted from each superconductor (used as predictors), which include geometric mean, range, and standard deviation. A more detail explanation of the variables can be found in the relevant scientific paper of the dataset (Word).

Figure 2:

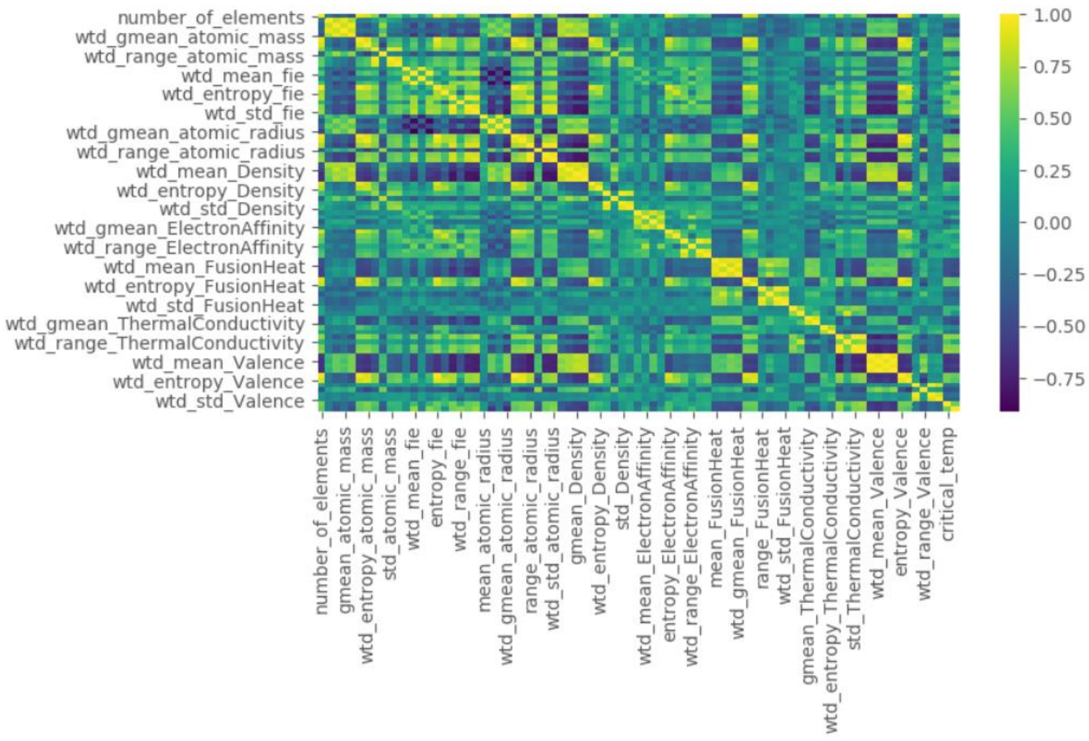
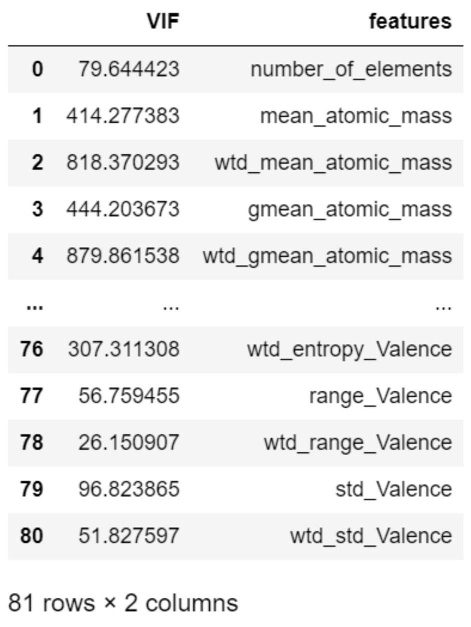


Figure 2: Correlation matrix heatmap of the data shows very high correlation between several variables. For instance, there is a high correlation (0.848) between the geometric mean and weighted mean features for atomic mass (Python).

Table 2:



	VIF	features
0	79.644423	number_of_elements
1	414.277383	mean_atomic_mass
2	818.370293	wtd_mean_atomic_mass
3	444.203673	gmean_atomic_mass
4	879.861538	wtd_gmean_atomic_mass
...	...	...
76	307.311308	wtd_entropy_Valence
77	56.759455	range_Valence
78	26.150907	wtd_range_Valence
79	96.823865	std_Valence
80	51.827597	wtd_std_Valence

81 rows × 2 columns

Table 2: VIF table of the predictor variables shows extremely high multicollinearity for multiple predictor variables. A general rule-of-thumb is that a VIF higher than 10 shows high multicollinearity in that predictor variable. Here we see nearly all predictors having extremely high multicollinearity. For instance, the feature *wtd_mean_atomic_mass* has a VIF of approximately 818.37 (Python).

Figure 3:

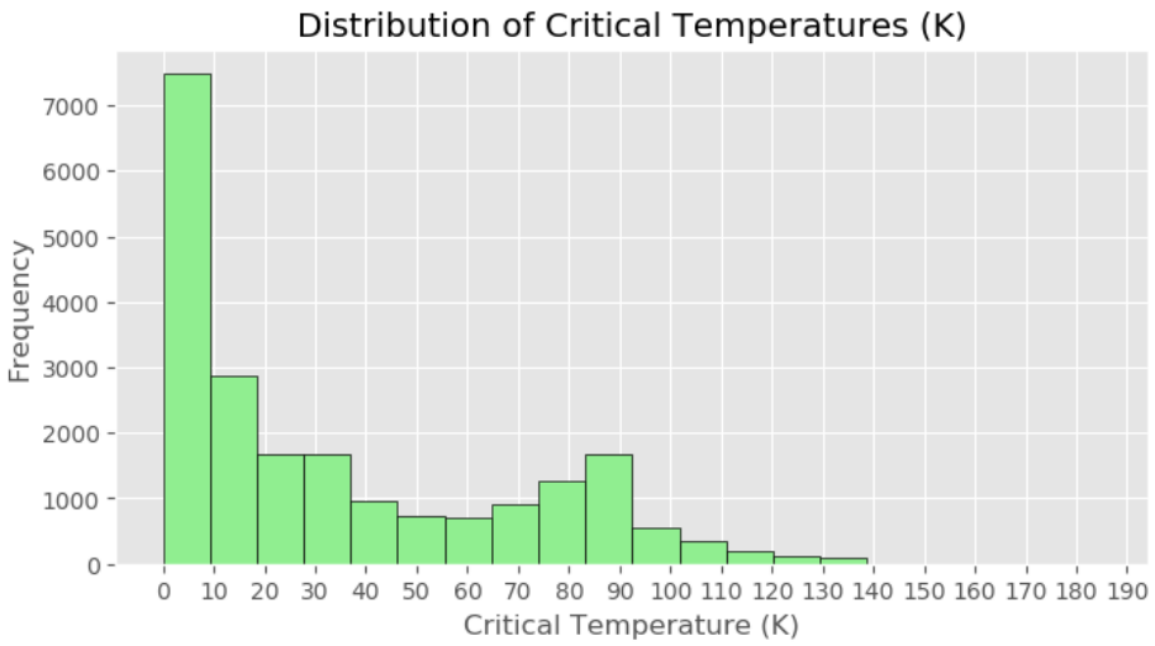


Figure 3: The histogram roughly shows the distribution of the response variable, critical temperature. The distribution is clearly right skewed since most of the critical temperatures are in between 0 K and 10 K. In theory, a conductor becomes superconductive after it drops below a certain temperature, called the critical temperature. This temperature is usually very cold which is why the critical temperature is usually measured in Kelvin. A superconductor is defined as a material that has no resistance to electrical current (Python).

Figure 4:

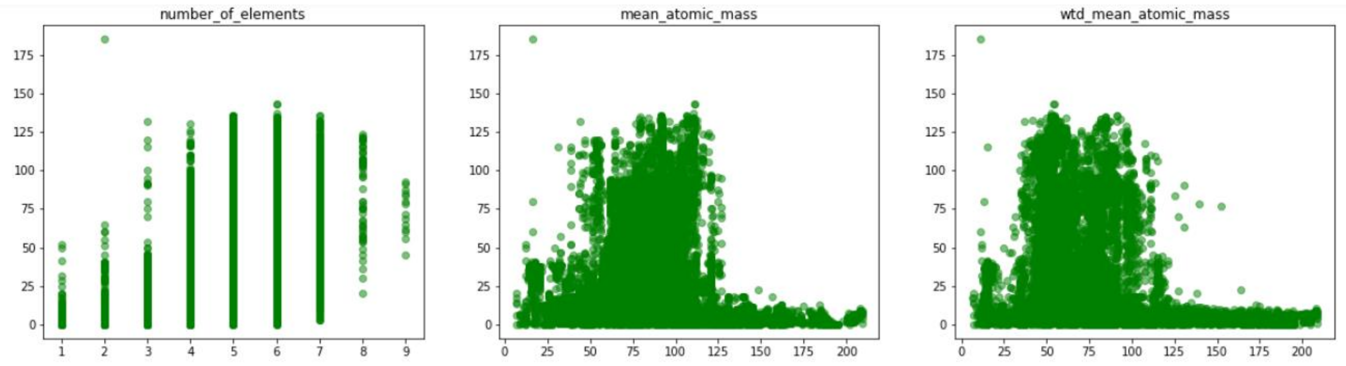


Figure 4: Here there are examples of three scatterplots made during the exploratory data analysis portion of the paper. They show critical temperature plotted against three predictor variables, *number_of_elements*, *mean_atomic_mass*, and *wtd_mean_atomic_mass*. There appears to be a weak linear relationship between the response and predictors, indicating that transformations on either the response or predictors should be explored in the model building process. Scatterplots like these were constructed for every predictor variable to briefly observe the relationship between each response and its predictors (Python).

Figure 5:

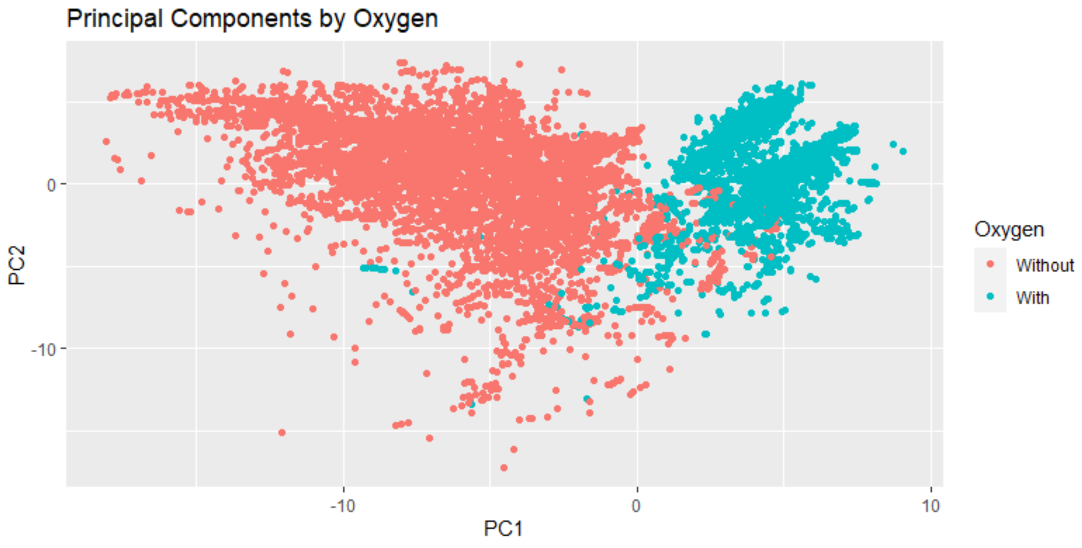


Figure 5: This is referred to as a score plot and it shows first two principal components plotted against each other, where the observations are grouped by superconductors with oxygen and those without oxygen (after PCA was calculated on the continuous variables). The score plot shows clear separation between the groups along the first principal component axis, which accounts for about 39% about the explained variance. Furthermore, there seems to be less variability within the oxygen group than within the no oxygen content group. There appears to be much less separation across the second principal component (RStudio).

Figure 6:

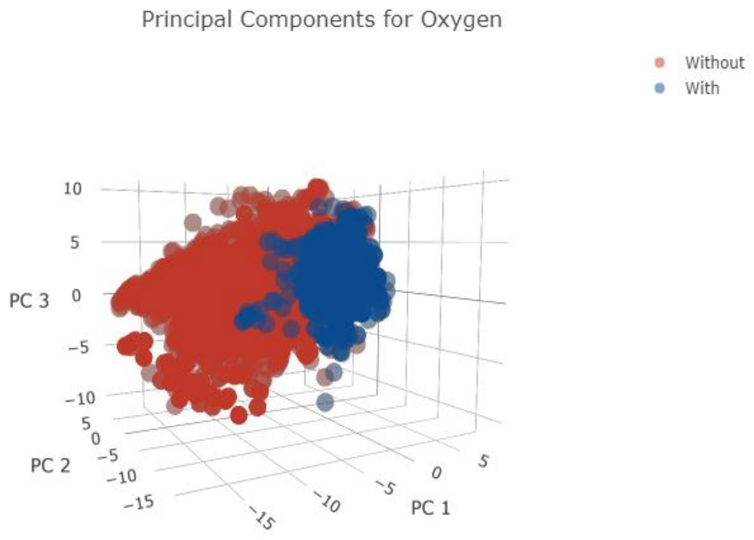


Figure 6: This plot depicts a 3D scatterplot of the first 3 principal components plotted against each other, which is an extension of figure 5 by using a third dimension. Again, we see clear separation between superconductors with oxygen and those without. The superconductors with oxygen seem to be much more similar to each other than those without oxygen (RStudio).

Figure 7:

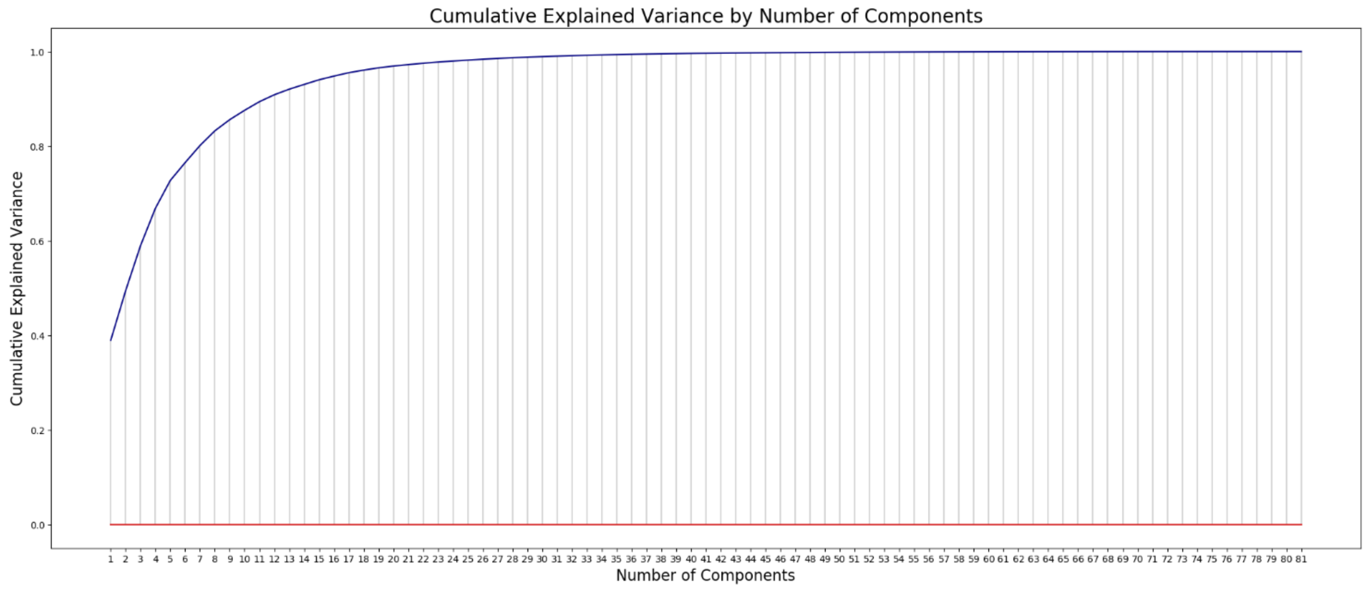


Figure 7: The above plot shows the cumulative explained variance by each subsequent principal component. The first principal component has already explained approximately 39% of the variance in the whole dataset. By 30 principal components, approximately 99% of the variance in the dataset is explained. This was the criteria used to decide on a subset of principal components to be used in the PCR model (Python).

Figure 8:

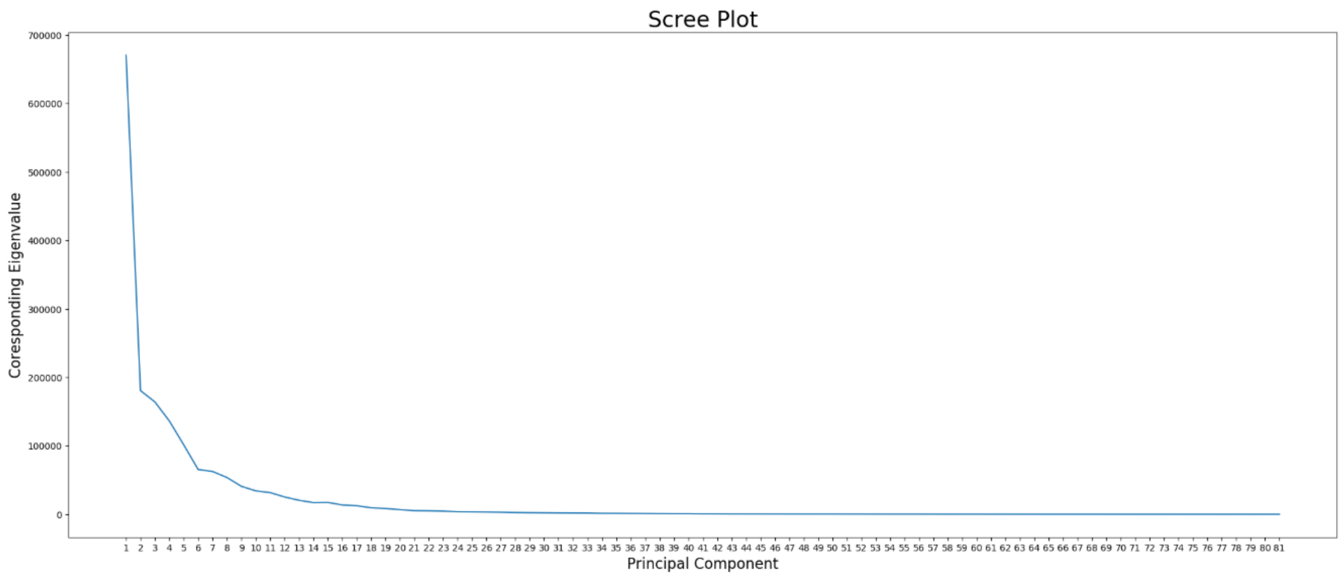


Figure 8: The above plot shows a scree plot of the principal components. This plot shows the corresponding eigenvalue plotted against its corresponding principal component. This is another method used to determine the number of principal components needed in the subset. The goal here would be to only keep the principal components that contain a 'large' enough eigenvalue (usually greater than 1). Although, in this example every principal component had an eigenvalue greater than 1. The explained cumulative variance method was used instead to attempt to subset the principal components appropriately (Python).

Figure 9:

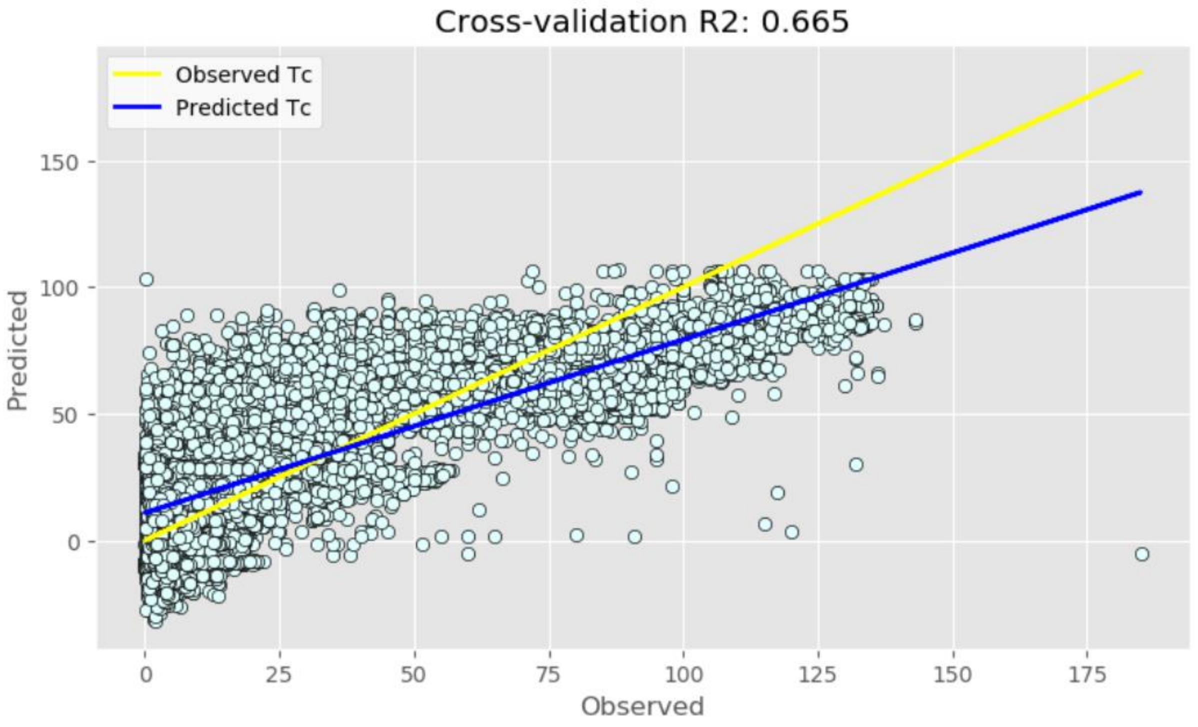
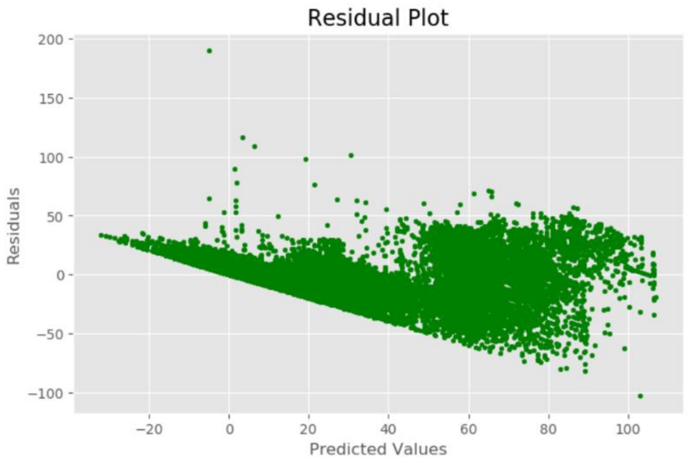
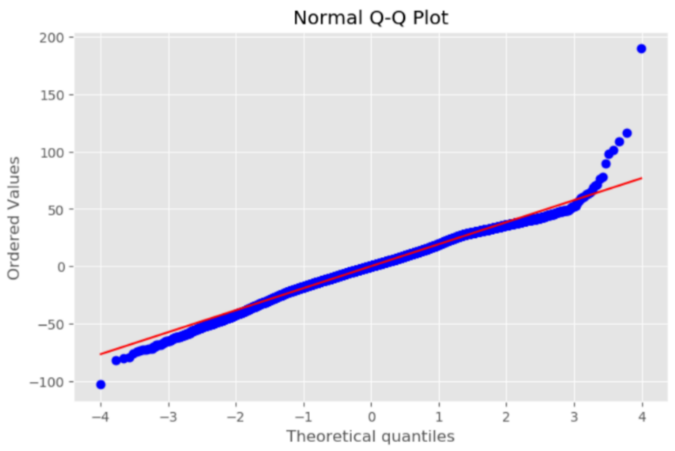


Figure 9: The above scatterplot shows a visual display of the regression lines for model 1. The yellow line represents the observed T_c values plotted against each other. The blue line represents the predicted T_c values for superconductors, based on the regression model. The blue dots represent predicted T_c values for superconductors with oxygen plotted against their corresponding observed T_c values (Python).

Figures 10 and 11:





Figures 10 and 11: These plots show the residual analysis of model 1. The residual plot on the left (figure 10) shows the residuals plotted against the predicted values. There seems to be a clear violation of the constant variance assumption because the plot takes a funnel-shaped pattern. The qq-plot on the right (figure 11) shows the observed residual quantiles plotted against the theoretical quantiles for a normal distribution. The residual quantiles roughly follow the red line except for the large quantiles, indicating skewness in the residuals and deviations from normality. These observations indicate that a transformation is needed on either the response variable or the predictor variables (Python).

Figure 12:

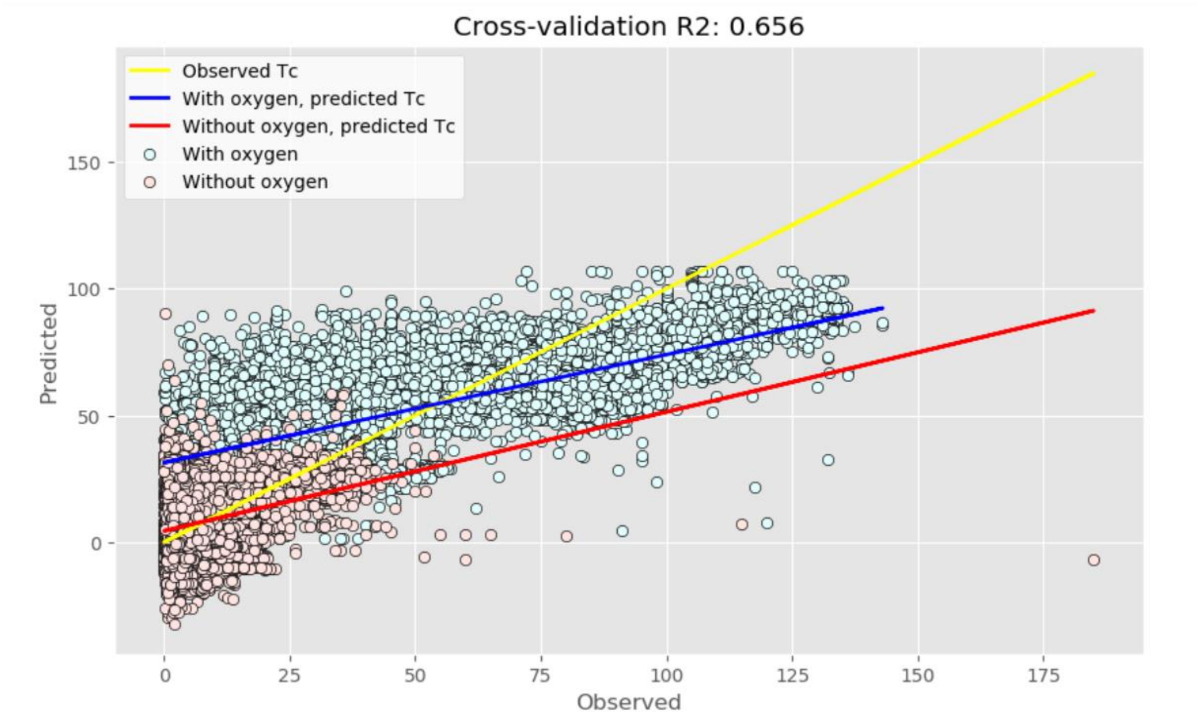
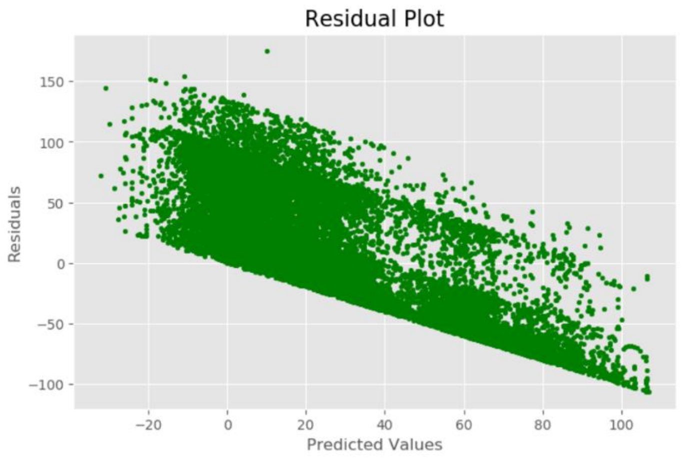
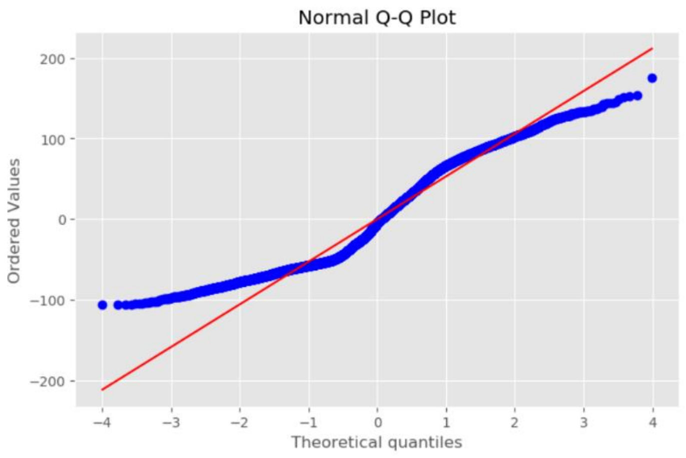


Figure 12: The above scatterplot shows a visual display of the regression lines for model 2. The yellow line represents the observed T_c values plotted against each other. The blue line represents predicted T_c values for superconductors with oxygen while the red line represents the predicted T_c values for superconductors without oxygen. The blue dots represent predicted T_c values for superconductors with oxygen while the red dots represent predicted T_c values for superconductors without oxygen, both plotted against their corresponding observed T_c values (Python).

Figures 13 and 14:





Figures 13 and 14: These plots show the residual analysis of model 2. The residual plot on the left (figure 13) shows the residuals plotted against the predicted values. There seems to be a clear violation of the constant variance assumption because the plot takes a funnel-shaped pattern. The qq-plot on the right (figure 14) shows the observed residual quantiles plotted against the theoretical quantiles for a normal distribution. The residual quantiles do not follow the red line, indicating the residuals are not normally distributed. These observations indicate that a transformation is needed on either the response variable or the predictor variables (Python).

Code Used for Analysis:

Python code

```
# for general dataframe editing
import pandas as pd
import numpy as np

# for plotting
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# for pca and pcr
import scipy.stats as stats
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error,r2_score
from numpy.linalg import eig
```

Data Setup and Cleaning

```
# import data
data = pd.read_csv('train.csv')
data_elements = pd.read_csv('unique_m.csv')

# look at dataset dimensions and first few observations
print(data.shape)
print(data_elements.shape)
data.head()
```

Exploratory Data Analysis

```
## list of all the predictor variables
predictors = ['number_of_elements', 'mean_atomic_mass', 'wtd_mean_atomic_m
ass',
       'gmean_atomic_mass', 'wtd_gmean_atomic_mass', 'entropy_atomic_mass'
,
       'wtd_entropy_atomic_mass', 'range_atomic_mass', 'wtd_range_atomic_m
ass',
       'std_atomic_mass', 'wtd_std_atomic_mass', 'mean_fie', 'wtd_mean_fie
',
       'gmean_fie', 'wtd_gmean_fie', 'entropy_fie', 'wtd_entropy_fie',
       'range_fie', 'wtd_range_fie', 'std_fie', 'wtd_std_fie',
       'mean_atomic_radius', 'wtd_mean_atomic_radius', 'gmean_atomic_radiu
s',
       'wtd_gmean_atomic_radius', 'entropy_atomic_radius',
       'wtd_entropy_atomic_radius', 'range_atomic_radius',
       'wtd_range_atomic_radius', 'std_atomic_radius', 'wtd_std_atomic_rad
ius',
       'mean_Density', 'wtd_mean_Density', 'gmean_Density',
       'wtd_gmean_Density', 'entropy_Density', 'wtd_entropy_Density',
       'range_Density', 'wtd_range_Density', 'std_Density', 'wtd_std_Densi
ty',
       'mean_ElectronAffinity', 'wtd_mean_ElectronAffinity',
       'gmean_ElectronAffinity', 'wtd_gmean_ElectronAffinity',
       'entropy_ElectronAffinity', 'wtd_entropy_ElectronAffinity',
       'range_ElectronAffinity', 'wtd_range_ElectronAffinity',
       'std_ElectronAffinity', 'wtd_std_ElectronAffinity', 'mean_FusionHea
t',
       'wtd_mean_FusionHeat', 'gmean_FusionHeat', 'wtd_gmean_FusionHeat',
       'entropy_FusionHeat', 'wtd_entropy_FusionHeat', 'range_FusionHeat',
       'wtd_range_FusionHeat', 'std_FusionHeat', 'wtd_std_FusionHeat',
       'mean_ThermalConductivity', 'wtd_mean_ThermalConductivity',
       'gmean_ThermalConductivity', 'wtd_gmean_ThermalConductivity',
       'entropy_ThermalConductivity', 'wtd_entropy_ThermalConductivity',
       'range_ThermalConductivity', 'wtd_range_ThermalConductivity',
       'std_ThermalConductivity', 'wtd_std_ThermalConductivity',
       'mean_Valence', 'wtd_mean_Valence', 'gmean_Valence',
       'wtd_gmean_Valence', 'entropy_Valence', 'wtd_entropy_Valence',
       'range_Valence', 'wtd_range_Valence', 'std_Valence', 'wtd_std_Valen
ce']


## making scatter plots for each predictor and the response
plt.figure(figsize=(20,160))
for i,j in enumerate(predictors):
    plt.subplot(27,3,i+1)
    plt.scatter(data[j],data["critical_temp"], color = 'g', alpha = 0.5)
    plt.title(predictors[i])
plt.show()



## looking at distribution of response variable

# setting plot style similar to R's ggplot
plt.style.use('ggplot')

# setting size of the plot
plt.figure(figsize = (8,4))

# plotting a histogram
plt.hist(data['critical_temp'].values, bins = 20,
        color = 'lightgreen',
        edgecolor = 'k')
plt.xlabel('Critical Temperature (K)')
plt.ylabel('Frequency')
plt.title('Distribution of Critical Temperatures (K)')
plt.xticks(np.arange(0, 200, step = 10));
```

```
# correlation matrix
corr_mat = data.corr()
corr_mat

# correlation matrix heatmap
plt.figure(figsize=(8,4))
sns.heatmap(corr_mat, cmap = 'viridis');

# VIF to detect multicollinearity
X = data.drop(['critical_temp'], axis = 1)
vif = pd.DataFrame()
vif["VIF"] = np.linalg.inv(X.corr()).diagonal()
vif["features"] = X.columns
vif
```

Principal Component Regression

```
# design matrix
X = data[predictors].values

# response vector
y = data['critical_temp'].values


## PCA by hand to look at eigenstructure and explained variance ##

# standardizing design matrix
X_std = StandardScaler().fit_transform(X)

# creating PCA object
pca = PCA()

# doing PCA to find principal components
Z = pca.fit_transform(X_std)

# creating principal components dataframe to look at
Z_df = pd.DataFrame(data = Z, columns = list(range(1,82)))
Z_df


## calculating cumulative explained variance ##

# getting eigenvalues and eigenvectors of the correlation matrix of the st
andardized design matrix
E, V = eig(X_std.T @ X_std)
print(E[0]/np.sum(E)) # explained variance for first principal component
cumvar = np.cumsum(pca.explained_variance_ratio_) # cumulative explained v
ariance for all principal components
cumvar
```

```
# making a plot for cumulative explained variance by the number of princip
al components
print(cumvar[29]) # looking at cumulative explained variance for first 30
principal components
plt.style.use('default')
plt.figure(figsize = (25,10))
plt.stem(cumvar, linefmt = 'lightgrey', markerfmt = 'navy', use_line_colle
ction = True)
plt.xticks(range(0,81), labels = list(range(1,82)))
plt.xlabel('Number of Components', fontsize = 16)
plt.ylabel('Cumulative Explained Variance', fontsize = 16);
plt.title('Cumulative Explained Variance by Number of Components', fontsiz
e = 20);


# making scree plot - shows eigenvalues by principal components
plt.style.use('default')
plt.figure(figsize = (25,10))
plt.plot(E)
plt.xticks(range(0,81), labels = list(range(1,82)))
plt.xlabel('Principal Component', fontsize = 16)
plt.ylabel('Coresponding Eigenvalue', fontsize = 16);
plt.title('Scree Plot', fontsize = 24);


## a function to easily do PCR - source of code found in references
section of paper - by 'Pelliccia' ##

def pcr(predictors, response, pc):

    ''' Step 1: PCA on input data'''
    # Define the PCA object
    pca = PCA()

    # Preprocess (2) Standardize features by removing the mean and scaling
to unit variance
    Xstd = StandardScaler().fit_transform(predictors)
    # Run PCA producing the reduced variable Xred and select the first pc
components
    Z = pca.fit_transform(Xstd)[:,:pc]
    ''' Step 2: regression on selected principal components'''
    # Create linear regression object
    regr = LinearRegression()
    # Fit
    Z_reg = regr.fit(Z, response)
    # predicted values
    y_hat = regr.predict(Z)
    # Cross-validation
    y_cv = cross_val_predict(Z_reg, Z, response, cv=10)
    # Calculate scores for OG model and cross-validation models
    R2 = r2_score(response, y_hat)
    R2_cv = r2_score(response, y_cv)
    # Calculate mean square error for OG model and cross validation models
    mse = mean_squared_error(response, y_hat)
    mse_cv = mean_squared_error(response, y_cv)

    ''' Step 3: Defining coefficients of Models'''
    # coefficients of regressors for principal components regression
    E, V = eig(Xstd.T @ Xstd)
    Beta_Z = Z_reg.coef_
    Beta_X = V[:,:pc] @ Beta_Z

    print('R2: %5.3f'  % R2)
    print('R2 CV: %5.3f'  % R2_cv)
    print('MSE: %5.3f' % mse)
    print('MSE CV: %5.3f' % mse_cv)
    print('Intercept:', Z_reg.intercept_)
    print('Coefficients:', Z_reg.coef_)
    return(Z_reg, Beta_X, y_hat, y_cv, R2, R2_cv, mse, mse_cv)


# the following several lines of code show PCR results for different subse
ts of principal components

Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 20)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 22)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 24)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 26)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 28)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 30)
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc = 32)


# just to see what the coefficients look like after transformed back to st
andardized design matrix units
print(Z_model.intercept_)
print(beta_X)


# for final model and for plotting
Z_model, beta_X, predicted, predicted_cv, r2, r2_cv, mse, mse_cv = pcr(X,
y, pc=30)


# fitting a regression line for plotting
z = np.polyfit(y, predicted, 1)

# plotting the predicted values by the observed values to show how well th
e model predicted critical temperature and also
# to plot the regression line
with plt.style.context(('ggplot')):
```

```python
    fig, ax = plt.subplots(figsize = (9, 5))
    ax.scatter(y, predicted, c = 'lightcyan', edgecolors = 'k')
    ax.plot(y, y, c = 'yellow', linewidth = 2, label = 'Observed Tc') # ca
n't make a dashed line for reasons I do not understand
    ax.plot(y, z[1] + z[0]*y, c ='blue', linewidth = 2, label = 'Predicted
Tc')
    plt.title('Cross-validation R2: ' + str(round(r2_cv,3)))
    plt.xlabel('Observed')
    plt.ylabel('Predicted')
    ax.legend(loc = 'upper left', facecolor = 'white');
    plt.show()


# a function I either made myself or found (can't remember or find source)
to find calculate five-number summary of a vector
def summary_stats(data):

    # calculate quartiles
    quartiles = np.percentile(data, [25, 50, 75])
    # calculate min/max
    data_min, data_max = data.min(), data.max()
    # print 5-number summary
    print('Min: %.3f' % data_min)
    print('Q1: %.3f' % quartiles[0])
    print('Median: %.3f' % quartiles[1])
    print('Q3: %.3f' % quartiles[2])
    print('Max: %.3f' % data_max)


# comparing five-number summaries between observed and predicted values
summary_stats(predicted)
summary_stats(y)


## plots for residual analysis ##

# setting plot style
plt.style.use('ggplot')

# plotting residual errors
plt.figure(figsize = (8,5))
plt.scatter(predicted, y - predicted,
            color = "green", s = 10)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')

# plot title
plt.title("Residual Plot", fontsize = 16);


# qq-plot of residuals
plt.figure(figsize = (8,5))
stats.probplot(y - predicted, dist="norm", plot=plt)
plt.title("Normal Q-Q Plot");
```

```
# this cell contains code that was used to compare particular elements one
at a time
# notice that I did not drop the 'Oxygen' ('O') element from the dataset '
data' - want to use it for regression analysis and
# to demonstrate using an indicator variable with PCR

data_elements = pd.read_csv('unique_m.csv')
data = pd.read_csv('train.csv')

data_elements = data_elements.drop(['critical_temp'], axis = 1)
data = pd.concat([data, data_elements], axis = 1)
data = data.drop(['H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'F', 'Ne','Na',
                  'Mg', 'Al', 'Si', 'P', 'S', 'Cl','Ar', 'K','Ca', 'Sc', '
Ti',
                  'V', 'Cr', 'Mn', 'Pb', 'Co','Ni', 'Cu', 'Zn', 'Ga', 'Ge'
,
                  'As', 'Se', 'Br', 'Kr', 'Rb','Sr', 'Y', 'Zr', 'Nb', 'Mo'
,
                  'Tc', 'Ru', 'Rh', 'Pd', 'Ag','Cd', 'In', 'Sn', 'Sb', 'Te
',
                  'I', 'Xe', 'Cs', 'Ba', 'La','Ce', 'Pr', 'Nd', 'Pm', 'Sm'
,
                  'Eu', 'Gd', 'Tb', 'Dy', 'Ho','Er', 'Tm', 'Yb', 'Lu', 'Hf
',
                  'Ta', 'W', 'Re', 'Os', 'Ir','Pt', 'Au', 'Hg', 'Tl', 'Fe'
,
                  'Bi', 'Po', 'At', 'Rn', 'material'], axis = 1)

data_elements = data_elements[['H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'O',
'F', 'Ne', 'Na', 'Mg', 'Al',
       'Si', 'P', 'S', 'Cl', 'Ar', 'K', 'Ca', 'Sc', 'Ti', 'V', 'Cr', 'Mn',
       'Fe', 'Co', 'Ni', 'Cu', 'Zn', 'Ga', 'Ge', 'As', 'Se', 'Br', 'Kr', '
Rb',
       'Sr', 'Y', 'Zr', 'Nb', 'Mo', 'Tc', 'Ru', 'Rh', 'Pd', 'Ag', 'Cd', 'I
n',
       'Sn', 'Sb', 'Te', 'I', 'Xe', 'Cs', 'Ba', 'La', 'Ce', 'Pr', 'Nd', 'P
m',
       'Sm', 'Eu', 'Gd', 'Tb', 'Dy', 'Ho', 'Er', 'Tm', 'Yb', 'Lu', 'Hf', '
Ta',
       'W', 'Re', 'Os', 'Ir', 'Pt', 'Au', 'Hg', 'Tl', 'Pb', 'Bi', 'Po', 'A
t',
       'Rn']]

data


# another five-number summary function (I definitely made this one myself)
specifically to use on the elements dataset
# I just wanted to get a sense of how many superconductors had which eleme
nts and how much of that element they had

def summary_stats(df):
    '''seven number summary of a numerical vector'''
    global summary
    names = ['Size','Min','25th Percentile','Median','75th Percentile','Ma
x','Mean','Standard Deviation']
    summary = pd.DataFrame(columns = names)
    elements = []
    for element in df.columns.values:
        if df[element].values.max() != 0:
            elements.append(element)
            x = df[df[element].values > 0]
            summaries = [len(x[element].values), round(x[element].values.m
in(),4),
                         round(np.percentile(x[element].values, 25),4),
                         round(np.percentile(x[element].values, 50),4),
                         round(np.percentile(x[element].values, 75),4),
                         round(x[element].values.max(),4),
                         round(x[element].values.mean(),4),
                         round(x[element].values.std(),4)]
            summaries = pd.DataFrame([summaries], columns = names)
            summary = summary.append(summaries)
    summary = summary.set_index([elements])
    return summary

# it shows the number of super conductors that contain a particular elemen
t 'Size' along with the five-number summary
# for instance, 299 superconductors contained Hydrogen and the superconduc
tor that had the
# most hydrogen had 14 (not sure about units)

summary_stats(data_elements)
summary.head()


# looking at all elements what were found in more than 2,000 superconducto
rs
# this is how I decided which elements to play with, eventually choosing O
xygen because of its clear PCA score plot
summary[summary.Size > 2000]


# restructuring data to do PCA and eventually introduce indicator variable
dataO = data[data['O'] > 0].reset_index()
yO = dataO['critical_temp']
dataO = dataO.drop(['index','critical_temp', 'O'], axis = 1)

dataNO = data[data['O'] == 0].reset_index()
yNO = dataNO['critical_temp']
dataNO = dataNO.drop(['index','critical_temp', 'O'], axis = 1)

y = pd.DataFrame(yNO)
y = y.append(pd.DataFrame(data = yO), ignore_index = True)

X_model = dataNO.append(dataO, ignore_index = True)

# making oxygen into an indicator variable through factorization
```

```python
Oxygen0 = data['O'][data['O'] == 0].reset_index().drop(['index'], axis = 1
)
Oxygen1 = pd.DataFrame({'O': np.ones(11964)})

Oxygen_edit = Oxygen0.append(Oxygen1, ignore_index = True)
Oxygen = Oxygen_edit.apply(lambda x: x.factorize()[0])



## PCR by hand to carefully account for indicator variable ##

# standardizing data
X_std = StandardScaler().fit_transform(X_model)

# creating PCA object
pca = PCA()

# principal components
Z = pca.fit_transform(X_std)[:,:30]

# creating principal components dataframe to look at
Z_pca = pd.DataFrame(data = Z, columns = list(range(1,31)))

# introducing indicator variable
Z_df = pd.concat([Oxygen, Z_pca], axis = 1)

# Create linear regression object
regr = LinearRegression()

# Fit
Z_reg = regr.fit(Z_df, y)

# predicted values
y_hat = regr.predict(Z_df)

# Cross-validation
y_cv = cross_val_predict(Z_reg, Z_df, y, cv=10)

# Calculate scores for OG model and cross-validation models
R2 = r2_score(y, y_hat)
R2_cv = r2_score(y, y_cv)

# Calculate mean square error for OG model and cross validation models
mse = mean_squared_error(y, y_hat)
mse_cv = mean_squared_error(y, y_cv)

print('R2: %5.3f'  % R2)
print('R2 CV: %5.3f'  % R2_cv)
print('MSE: %5.3f' % mse)
print('MSE CV: %5.3f' % mse_cv)
print('Intercept:', Z_reg.intercept_)
print('Coefficients:', Z_reg.coef_)

# making observed values into a usable vector
```

```
yvec = y['critical_temp']


# fitting regression lines for plotting
z0 = np.polyfit(yvec[:9299], y_hat[:9299], 1)
z1 = np.polyfit(yvec[9299::], y_hat[9299::], 1)

# plotting the predicted values by the observed values to show how well th
e model predicted critical temperature and also
# to plot the regression lines
with plt.style.context(('ggplot')):
    fig, ax = plt.subplots(figsize = (10, 6))
    ax.scatter(yvec[9299::], y_hat[9299::], c = 'lightcyan', edgecolors =
'k', label = 'With oxygen')
    ax.scatter(yvec[:9299], y_hat[:9299], c = 'mistyrose', edgecolors = 'k
', label= 'Without oxygen')
    ax.plot(yvec, yvec, c ='yellow', linewidth = 2, label = 'Observed Tc')
# still can't coerce a dashed line
    ax.plot(yvec[:9299], z0[1] + z0[0]*yvec[:9299], c ='red', linewidth =
2, label = 'Without oxygen, predicted Tc')
    ax.plot(yvec[9299::], z1[1] + z1[0]*yvec[9299::], c ='blue', linewidth
= 2, label = 'With oxygen, predicted Tc')
    plt.title('Cross-validation R2: ' + str(round(R2_cv,3)))
    plt.xlabel('Observed')
    plt.ylabel('Predicted')
    ax.legend(loc = 'upper left', facecolor = 'white');
    plt.show()



## plots for residual analysis ##

# setting plot style
plt.style.use('ggplot')

# plotting residual errors
plt.figure(figsize = (8,5))
plt.scatter(predicted, yvec - predicted,
            color = "green", s = 10)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')

# plot title
plt.title("Residual Plot", fontsize = 16);


# qq-plot of residuals
plt.figure(figsize = (8,5))
stats.probplot(yvec - predicted, dist="norm", plot=plt)
plt.title("Normal Q-Q Plot");
```

RStudio code

```r
#####   PCA data visualization   ###########################################

library('ggplot2')
library('plotly')

# load in the data
data_super = read.table('train.csv', header = TRUE, sep = ',')
data_elements = read.table('unique_m.csv', header = TRUE, sep = ',')


# creating categorical variable for oxygen
Oxy = data_elements['O'] > 0
Oxy = as.numeric(Oxy)
Oxy = factor(Oxy, labels = c('Without','With'))

#####    PCA     #############################################

# design matrix
X = data_super[,1:81]

# PCA
sc_pcr = prcomp(X, scale = T)
PCscores = sc_pcr$x

# combing PC scores with factorized variable 'Oxygen'
pca_data = data.frame(PCscores, Oxygen = Oxy)

# PC score plot of the first two principal components, grouped by Oxygen
ggplot(pca_data, aes(x = PC1, y = PC2, color = Oxygen)) + geom_point() + labs
(title = "Principal Components by Oxygen")

# 3D PC score plot of the first three principal components, grouped by Oxygen
fig <- plot_ly(x = PCscores[,1], y = PCscores[,2],
               z = PCscores[,3], color = Oxy,
               colors = c('#BF382A', '#0C4B8E'),
               marker = list(symbol = 'circle',
                             opacity = 0.5))

fig <- fig %>% add_markers()

fig <- fig %>% layout(title = 'Principal Components for Oxygen',
                      scene = list(xaxis = list(title = 'PC 1'),
                                   yaxis = list(title = 'PC 2'),
                                   zaxis = list(title = 'PC 3')))

fig
```