FIT5149 S2 2019 Assessment 1 Predicting the Critical Temperature of a Superconductor

15 September 2019

Contents

Student information

Family Name: Sathe
Given Name: Suyash
Student ID: 29279208
Student email: ssat0005@student.monash.edu

Programming Language

- R 3.5.1 in Jupyter Notebook

Libraries

- **psych:** Statistical Computation and multivariate analysis.
- **caret:** Model training workflow.
- **tidyverse:** Manipulating and visualizing data.
- **dplyr:** Data Manipulation.
- **ggplot2:** Graphical analysis of data.
- **GGally:** Extend GGplot2 to reduce the complexity of combining geometric objects with transformed data.
- **randomForest:** Implement Breiman's random forest algorithm for regression.
- **gbm:** Implement gradient boosted regression models.

Introduction

The objective of this project is to analyse the multivariate dataset of superconductors and predict the critical temperature given the critical properties of the superconductor's materials. The Critical temperature of a conductor is the temperature below which the material has zero electrical resistance. This temperature is dependent on various chemical properties of a material.

In this assignment, the dataset containing 21,623 records of superconducting materials corresponding to 80 chemical properties (features) was explored and analysed to obtain the best set of features that are crucial for predicting the critical temperature of a superconductor. In the pre-processing step, the filter method was used to eliminate the irrelevant features, and Pearson's correlation coefficient was used to quantify the linear relationship between two continuous variables.

After filtering the set of features, Recursive Feature Elimination was used to obtain the best subset of features. This final set of features was then utilised to train 3 different models including the linear regression model, random forest model and gradient boosted model, and the performance of these models was evaluated based on R-squared (R2) value and RMSE score. The best model was selected based on the highest R2 or lowest RMSE value. After comparing the models, the best model was utilised to identify and describe the key properties for predicting the critical temperature of a superconductor.

Data Exploration

Libraries and packages

- Load the required set of libraries and packages.

```
# load the library.

require(psych)
```

```
## Loading required package: psych
```

```
## Warning: package 'psych' was built under R version 3.5.3
```

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```
require(tidyverse)
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 3.5.3
```

```
## -- Attaching packages ---------------------------------------- tidyverse 1.2.1 --
```

```
## v tibble  1.4.2     v purrr   0.2.5
## v tidyr   0.8.1     v dplyr   0.7.6
## v readr   1.1.1     v stringr 1.3.1
## v tibble  1.4.2     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------- tidyverse_conflicts() --
## x ggplot2::%+%()   masks psych::%+%()
## x ggplot2::alpha() masks psych::alpha()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x purrr::lift()    masks caret::lift()
```

```
require(dplyr)
require(ggplot2)
require(GGally)
```

```
## Loading required package: GGally
```

```
## Warning: package 'GGally' was built under R version 3.5.3
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
```

```
##     nasa
```

```
require(randomForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:psych':
##
##     outlier
```

```
require(gbm)
```

```
## Loading required package: gbm
```

```
## Warning: package 'gbm' was built under R version 3.5.3
```

```
## Loaded gbm 2.1.5
```

- To ensure the simulation of random objects, set the seed value to a random value.

```
# ensure the results are repeatable
set.seed(798)
```

Read the Datasets

- There are two datasets provided as a part of these assignment.
- The "train.csv" contains the statistical properties of the superconductors, and the "unique_m.csv"" contains the elements present in each superconductor.

```
# Read the complete dataset
data <- read.csv('train.csv', header = TRUE)

# Read the dataset of elements
elements <- read.csv('unique_m.csv', header = TRUE)
```

- Since the *critical temperature* of a superconductor may also depend on the composition of the material, we are combining the two datasets.
- Using this, we can analyse the dependency of *critical temperature* on the chemical properties as well as the chemical composition of the super conductor.

```
# Combine both the datasets
combined_data <- cbind(data, elements)

# Check the first few elements of the dataset
head(combined_data)
```

```
##   number_of_elements mean_atomic_mass wtd_mean_atomic_mass
```

```
## 1                 4         88.94447             57.86269
## 2                 5         92.72921             58.51842
## 3                 4         88.94447             57.88524
## 4                 4         88.94447             57.87397
## 5                 4         88.94447             57.84014
## 6                 4         88.94447             57.79504
##   gmean_atomic_mass wtd_gmean_atomic_mass entropy_atomic_mass
## 1          66.36159              36.11661            1.181795
## 2          73.13279              36.39660            1.449309
## 3          66.36159              36.12251            1.181795
## 4          66.36159              36.11956            1.181795
## 5          66.36159              36.11072            1.181795
## 6          66.36159              36.09893            1.181795
##   wtd_entropy_atomic_mass range_atomic_mass wtd_range_atomic_mass
## 1               1.0623955          122.9061              31.79492
## 2               1.0577551          122.9061              36.16194
## 3               0.9759805          122.9061              35.74110
## 4               1.0222909          122.9061              33.76801
## 5               1.1292237          122.9061              27.84874
## 6               1.2252028          122.9061              20.68746
##   std_atomic_mass wtd_std_atomic_mass mean_fie wtd_mean_fie gmean_fie
## 1        51.96883            53.62253  775.425     1010.269  718.1529
## 2        47.09463            53.97987  766.440     1010.613  720.6055
## 3        51.96883            53.65627  775.425     1010.820  718.1529
## 4        51.96883            53.63940  775.425     1010.544  718.1529
## 5        51.96883            53.58877  775.425     1009.717  718.1529
## 6        51.96883            53.52115  775.425     1008.614  718.1529
##   wtd_gmean_fie entropy_fie wtd_entropy_fie range_fie wtd_range_fie
## 1      938.0168    1.305967       0.7914878     810.6      735.9857
## 2      938.7454    1.544145       0.8070782     810.6      743.1643
## 3      939.0090    1.305967       0.7736202     810.6      743.1643
## 4      938.5128    1.305967       0.7832067     810.6      739.5750
## 5      937.0256    1.305967       0.8052296     810.6      728.8071
## 6      935.0463    1.305967       0.8247426     810.6      714.4500
##    std_fie wtd_std_fie mean_atomic_radius wtd_mean_atomic_radius
## 1 323.8118    355.5630             160.25               105.5143
## 2 290.1830    354.9635             161.20               104.9714
## 3 323.8118    354.8042             160.25               104.6857
## 4 323.8118    355.1839             160.25               105.1000
## 5 323.8118    356.3193             160.25               106.3429
## 6 323.8118    357.8246             160.25               108.0000
##   gmean_atomic_radius wtd_gmean_atomic_radius entropy_atomic_radius
## 1            136.1260                84.52842              1.259244
## 2            141.4652                84.37017              1.508328
## 3            136.1260                84.21457              1.259244
## 4            136.1260                84.37135              1.259244
## 5            136.1260                84.84344              1.259244
## 6            136.1260                85.47701              1.259244
##   wtd_entropy_atomic_radius range_atomic_radius wtd_range_atomic_radius
## 1                  1.207040                205                42.91429
## 2                  1.204115                205                50.57143
## 3                  1.132547                205                49.31429
## 4                  1.173033                205                46.11429
## 5                  1.261194                205                36.51429
```

```
## 6                 1.331339                205               23.71429
##   std_atomic_radius wtd_std_atomic_radius mean_Density wtd_mean_Density
## 1          75.23754              69.23557     4654.357         2961.502
## 2          67.32132              68.00882     5821.486         3021.017
## 3          75.23754              67.79771     4654.357         2999.159
## 4          75.23754              68.52166     4654.357         2980.331
## 5          75.23754              70.63445     4654.357         2923.845
## 6          75.23754              73.32413     4654.357         2848.531
##   gmean_Density wtd_gmean_Density entropy_Density wtd_entropy_Density
## 1      724.9532          53.54381        1.033129           0.8145982
## 2     1237.0951          54.09572        1.314442           0.9148022
## 3      724.9532          53.97402        1.033129           0.7603052
## 4      724.9532          53.75849        1.033129           0.7888885
## 5      724.9532          53.11703        1.033129           0.8598109
## 6      724.9532          52.27364        1.033129           0.9323687
##   range_Density wtd_range_Density std_Density wtd_std_Density
## 1      8958.571          1579.583    3306.163        3572.597
## 2     10488.571          1667.383    3767.403        3632.649
## 3      8958.571          1667.383    3306.163        3592.019
## 4      8958.571          1623.483    3306.163        3582.371
## 5      8958.571          1491.783    3306.163        3552.669
## 6      8958.571          1316.183    3306.163        3511.262
##   mean_ElectronAffinity wtd_mean_ElectronAffinity gmean_ElectronAffinity
## 1               81.8375                  111.7271               60.12318
## 2               90.8900                  112.3164               69.83331
## 3               81.8375                  112.2136               60.12318
## 4               81.8375                  111.9704               60.12318
## 5               81.8375                  111.2407               60.12318
## 6               81.8375                  110.2679               60.12318
##   wtd_gmean_ElectronAffinity entropy_ElectronAffinity
## 1                   99.41468                1.159687
## 2                  101.16640                1.427997
## 3                  101.08215                1.159687
## 4                  100.24495                1.159687
## 5                   97.77472                1.159687
## 6                   94.57550                1.159687
##   wtd_entropy_ElectronAffinity range_ElectronAffinity
## 1                    0.7873817                 127.05
## 2                    0.8386665                 127.05
## 3                    0.7860067                 127.05
## 4                    0.7869005                 127.05
## 5                    0.7873962                 127.05
## 6                    0.7844615                 127.05
##   wtd_range_ElectronAffinity std_ElectronAffinity wtd_std_ElectronAffinity
## 1                   80.98714             51.43371                 42.55840
## 2                   81.20786             49.43817                 41.66762
## 3                   81.20786             51.43371                 41.63988
## 4                   81.09750             51.43371                 42.10234
## 5                   80.76643             51.43371                 43.45206
## 6                   80.32500             51.43371                 45.17068
##   mean_FusionHeat wtd_mean_FusionHeat gmean_FusionHeat
## 1          6.9055            3.846857         3.479475
## 2          7.7844            3.796857         4.403790
## 3          6.9055            3.822571         3.479475
```

```
## 4          6.9055           3.834714         3.479475
## 5          6.9055           3.871143         3.479475
## 6          6.9055           3.919714         3.479475
##   wtd_gmean_FusionHeat entropy_FusionHeat wtd_entropy_FusionHeat
## 1             1.040986          1.088575              0.9949982
## 2             1.035251          1.374977              1.0730938
## 3             1.037439          1.088575              0.9274794
## 4             1.039211          1.088575              0.9640310
## 5             1.044545          1.088575              1.0449695
## 6             1.051699          1.088575              1.1118503
##   range_FusionHeat wtd_range_FusionHeat std_FusionHeat wtd_std_FusionHeat
## 1           12.878             1.744571       4.599064           4.666920
## 2           12.878             1.595714       4.473363           4.603000
## 3           12.878             1.757143       4.599064           4.649635
## 4           12.878             1.744571       4.599064           4.658301
## 5           12.878             1.744571       4.599064           4.684014
## 6           12.878             1.744571       4.599064           4.717642
##   mean_ThermalConductivity wtd_mean_ThermalConductivity
## 1                107.7566                     61.01519
## 2                172.2053                     61.37233
## 3                107.7566                     60.94376
## 4                107.7566                     60.97947
## 5                107.7566                     61.08662
## 6                107.7566                     61.22947
##   gmean_ThermalConductivity wtd_gmean_ThermalConductivity
## 1                  7.062488                     0.6219795
## 2                 16.064228                     0.6197346
## 3                  7.062488                     0.6190947
## 4                  7.062488                     0.6205354
## 5                  7.062488                     0.6248777
## 6                  7.062488                     0.6307148
##   entropy_ThermalConductivity wtd_entropy_ThermalConductivity
## 1                   0.3081480                       0.2628483
## 2                   0.8474042                       0.5677061
## 3                   0.3081480                       0.2504774
## 4                   0.3081480                       0.2570451
## 5                   0.3081480                       0.2728199
## 6                   0.3081480                       0.2882356
##   range_ThermalConductivity wtd_range_ThermalConductivity
## 1                  399.9734                      57.12767
## 2                  429.9734                      51.41338
## 3                  399.9734                      57.12767
## 4                  399.9734                      57.12767
## 5                  399.9734                      57.12767
## 6                  399.9734                      57.12767
##   std_ThermalConductivity wtd_std_ThermalConductivity mean_Valence
## 1                168.8542                    138.5172        2.25
## 2                198.5546                    139.6309        2.00
## 3                168.8542                    138.5406        2.25
## 4                168.8542                    138.5289        2.25
## 5                168.8542                    138.4937        2.25
## 6                168.8542                    138.4466        2.25
##   wtd_mean_Valence gmean_Valence wtd_gmean_Valence entropy_Valence
## 1         2.257143      2.213364          2.219783        1.368922
```

```
## 2         2.257143      1.888175          2.210679        1.557113
## 3         2.271429      2.213364          2.232679        1.368922
## 4         2.264286      2.213364          2.226222        1.368922
## 5         2.242857      2.213364          2.206963        1.368922
## 6         2.214286      2.213364          2.181543        1.368922
##   wtd_entropy_Valence range_Valence wtd_range_Valence std_Valence
## 1            1.066221             1          1.085714   0.4330127
## 2            1.047221             2          1.128571   0.6324555
## 3            1.029175             1          1.114286   0.4330127
## 4            1.048834             1          1.100000   0.4330127
## 5            1.096052             1          1.057143   0.4330127
## 6            1.141474             1          1.000000   0.4330127
##   wtd_std_Valence critical_temp H He Li Be B C N O F Ne Na Mg Al Si P S Cl
## 1       0.4370588            29 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
## 2       0.4686063            26 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
## 3       0.4446966            19 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
## 4       0.4409521            22 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
## 5       0.4288095            23 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
## 6       0.4103259            23 0  0  0  0 0 0 0 4 0  0  0  0  0  0 0 0  0
##   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
## 1  0 0  0  0  0 0  0  0  0  0  0 1.0  0  0  0  0  0  0  0  0  0 0  0  0  0
## 2  0 0  0  0  0 0  0  0  0  0  0 0.9  0  0  0  0  0  0  0  0  0 0  0  0  0
## 3  0 0  0  0  0 0  0  0  0  0  0 1.0  0  0  0  0  0  0  0  0  0 0  0  0  0
## 4  0 0  0  0  0 0  0  0  0  0  0 1.0  0  0  0  0  0  0  0  0  0 0  0  0  0
## 5  0 0  0  0  0 0  0  0  0  0  0 1.0  0  0  0  0  0  0  0  0  0 0  0  0  0
## 6  0 0  0  0  0 0  0  0  0  0  0 1.0  0  0  0  0  0  0  0  0  0 0  0  0  0
##   Tc Ru Rh Pd  Ag Cd In Sn Sb Te I Xe Cs   Ba   La Ce Pr Nd Pm Sm Eu Gd Tb
## 1  0  0  0  0 0.0  0  0  0  0  0 0  0  0 0.20 1.80  0  0  0  0  0  0  0  0
## 2  0  0  0  0 0.1  0  0  0  0  0 0  0  0 0.10 1.90  0  0  0  0  0  0  0  0
## 3  0  0  0  0 0.0  0  0  0  0  0 0  0  0 0.10 1.90  0  0  0  0  0  0  0  0
## 4  0  0  0  0 0.0  0  0  0  0  0 0  0  0 0.15 1.85  0  0  0  0  0  0  0  0
## 5  0  0  0  0 0.0  0  0  0  0  0 0  0  0 0.30 1.70  0  0  0  0  0  0  0  0
## 6  0  0  0  0 0.0  0  0  0  0  0 0  0  0 0.50 1.50  0  0  0  0  0  0  0  0
##   Dy Ho Er Tm Yb Lu Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
## 1  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 2  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 3  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 4  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 5  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 6  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0
##   critical_temp              material
## 1            29        Ba0.2La1.8Cu1O4
## 2            26 Ba0.1La1.9Ag0.1Cu0.9O4
## 3            19        Ba0.1La1.9Cu1O4
## 4            22      Ba0.15La1.85Cu1O4
## 5            23        Ba0.3La1.7Cu1O4
## 6            23        Ba0.5La1.5Cu1O4
```

Overview of the dataset

- Check the dimensionality of the dataset. This dataset has 21263 rows and 170 columns, i.e., this combined dataset has 170 features.

```r
# Check the dimensions of the dataset
dim(combined_data)
```

```
## [1] 21263   170
```

```r
# Check for null values
sum(is.na(combined_data))
```

```
## [1] 0
```

- Checking for null values in the dataset, there are 0 null values in this dataset.

```r
# Check for null values
sum(is.na(combined_data))
```

```
## [1] 0
```

- Get an overview of the structure of this dataset.

```r
# Check the structure of the dataset
str(combined_data)
```

```
## 'data.frame':    21263 obs. of  170 variables:
##  $ number_of_elements            : int  4 5 4 4 4 4 4 4 4 4 ...
##  $ mean_atomic_mass              : num  88.9 92.7 88.9 88.9 88.9 ...
##  $ wtd_mean_atomic_mass          : num  57.9 58.5 57.9 57.9 57.8 ...
##  $ gmean_atomic_mass             : num  66.4 73.1 66.4 66.4 66.4 ...
##  $ wtd_gmean_atomic_mass         : num  36.1 36.4 36.1 36.1 36.1 ...
##  $ entropy_atomic_mass           : num  1.18 1.45 1.18 1.18 1.18 ...
##  $ wtd_entropy_atomic_mass       : num  1.062 1.058 0.976 1.022 1.129 ...
##  $ range_atomic_mass             : num  123 123 123 123 123 ...
##  $ wtd_range_atomic_mass         : num  31.8 36.2 35.7 33.8 27.8 ...
##  $ std_atomic_mass               : num  52 47.1 52 52 52 ...
##  $ wtd_std_atomic_mass           : num  53.6 54 53.7 53.6 53.6 ...
##  $ mean_fie                      : num  775 766 775 775 775 ...
##  $ wtd_mean_fie                  : num  1010 1011 1011 1011 1010 ...
##  $ gmean_fie                     : num  718 721 718 718 718 ...
##  $ wtd_gmean_fie                 : num  938 939 939 939 937 ...
##  $ entropy_fie                   : num  1.31 1.54 1.31 1.31 1.31 ...
##  $ wtd_entropy_fie               : num  0.791 0.807 0.774 0.783 0.805 ...
##  $ range_fie                     : num  811 811 811 811 811 ...
##  $ wtd_range_fie                 : num  736 743 743 740 729 ...
##  $ std_fie                       : num  324 290 324 324 324 ...
##  $ wtd_std_fie                   : num  356 355 355 355 356 ...
##  $ mean_atomic_radius            : num  160 161 160 160 160 ...
##  $ wtd_mean_atomic_radius        : num  106 105 105 105 106 ...
##  $ gmean_atomic_radius           : num  136 141 136 136 136 ...
##  $ wtd_gmean_atomic_radius       : num  84.5 84.4 84.2 84.4 84.8 ...
##  $ entropy_atomic_radius         : num  1.26 1.51 1.26 1.26 1.26 ...
##  $ wtd_entropy_atomic_radius     : num  1.21 1.2 1.13 1.17 1.26 ...
##  $ range_atomic_radius           : int  205 205 205 205 205 205 205 171 171 171 ...
##  $ wtd_range_atomic_radius       : num  42.9 50.6 49.3 46.1 36.5 ...
##  $ std_atomic_radius             : num  75.2 67.3 75.2 75.2 75.2 ...
##  $ wtd_std_atomic_radius         : num  69.2 68 67.8 68.5 70.6 ...
##  $ mean_Density                  : num  4654 5821 4654 4654 4654 ...
##  $ wtd_mean_Density              : num  2962 3021 2999 2980 2924 ...
##  $ gmean_Density                 : num  725 1237 725 725 725 ...
```

```
##  $ wtd_gmean_Density              : num  53.5 54.1 54 53.8 53.1 ...
##  $ entropy_Density                : num  1.03 1.31 1.03 1.03 1.03 ...
##  $ wtd_entropy_Density            : num  0.815 0.915 0.76 0.789 0.86 ...
##  $ range_Density                  : num  8959 10489 8959 8959 8959 ...
##  $ wtd_range_Density              : num  1580 1667 1667 1623 1492 ...
##  $ std_Density                    : num  3306 3767 3306 3306 3306 ...
##  $ wtd_std_Density                : num  3573 3633 3592 3582 3553 ...
##  $ mean_ElectronAffinity          : num  81.8 90.9 81.8 81.8 81.8 ...
##  $ wtd_mean_ElectronAffinity      : num  112 112 112 112 111 ...
##  $ gmean_ElectronAffinity         : num  60.1 69.8 60.1 60.1 60.1 ...
##  $ wtd_gmean_ElectronAffinity     : num  99.4 101.2 101.1 100.2 97.8 ...
##  $ entropy_ElectronAffinity       : num  1.16 1.43 1.16 1.16 1.16 ...
##  $ wtd_entropy_ElectronAffinity   : num  0.787 0.839 0.786 0.787 0.787 ...
##  $ range_ElectronAffinity         : num  127 127 127 127 127 ...
##  $ wtd_range_ElectronAffinity     : num  81 81.2 81.2 81.1 80.8 ...
##  $ std_ElectronAffinity           : num  51.4 49.4 51.4 51.4 51.4 ...
##  $ wtd_std_ElectronAffinity       : num  42.6 41.7 41.6 42.1 43.5 ...
##  $ mean_FusionHeat                : num  6.91 7.78 6.91 6.91 6.91 ...
##  $ wtd_mean_FusionHeat            : num  3.85 3.8 3.82 3.83 3.87 ...
##  $ gmean_FusionHeat               : num  3.48 4.4 3.48 3.48 3.48 ...
##  $ wtd_gmean_FusionHeat           : num  1.04 1.04 1.04 1.04 1.04 ...
##  $ entropy_FusionHeat             : num  1.09 1.37 1.09 1.09 1.09 ...
##  $ wtd_entropy_FusionHeat         : num  0.995 1.073 0.927 0.964 1.045 ...
##  $ range_FusionHeat               : num  12.9 12.9 12.9 12.9 12.9 ...
##  $ wtd_range_FusionHeat           : num  1.74 1.6 1.76 1.74 1.74 ...
##  $ std_FusionHeat                 : num  4.6 4.47 4.6 4.6 4.6 ...
##  $ wtd_std_FusionHeat             : num  4.67 4.6 4.65 4.66 4.68 ...
##  $ mean_ThermalConductivity       : num  108 172 108 108 108 ...
##  $ wtd_mean_ThermalConductivity   : num  61 61.4 60.9 61 61.1 ...
##  $ gmean_ThermalConductivity      : num  7.06 16.06 7.06 7.06 7.06 ...
##  $ wtd_gmean_ThermalConductivity  : num  0.622 0.62 0.619 0.621 0.625 ...
##  $ entropy_ThermalConductivity    : num  0.308 0.847 0.308 0.308 0.308 ...
##  $ wtd_entropy_ThermalConductivity: num  0.263 0.568 0.25 0.257 0.273 ...
##  $ range_ThermalConductivity      : num  400 430 400 400 400 ...
##  $ wtd_range_ThermalConductivity  : num  57.1 51.4 57.1 57.1 57.1 ...
##  $ std_ThermalConductivity        : num  169 199 169 169 169 ...
##  $ wtd_std_ThermalConductivity    : num  139 140 139 139 138 ...
##  $ mean_Valence                   : num  2.25 2 2.25 2.25 2.25 2.25 2.25 2.25 2.25 2.25 ...
##  $ wtd_mean_Valence               : num  2.26 2.26 2.27 2.26 2.24 ...
##  $ gmean_Valence                  : num  2.21 1.89 2.21 2.21 2.21 ...
##  $ wtd_gmean_Valence              : num  2.22 2.21 2.23 2.23 2.21 ...
##  $ entropy_Valence                : num  1.37 1.56 1.37 1.37 1.37 ...
##  $ wtd_entropy_Valence            : num  1.07 1.05 1.03 1.05 1.1 ...
##  $ range_Valence                  : int  1 2 1 1 1 1 1 1 1 1 ...
##  $ wtd_range_Valence              : num  1.09 1.13 1.11 1.1 1.06 ...
##  $ std_Valence                    : num  0.433 0.632 0.433 0.433 0.433 ...
##  $ wtd_std_Valence                : num  0.437 0.469 0.445 0.441 0.429 ...
##  $ critical_temp                  : num  29 26 19 22 23 23 11 33 36 31 ...
##  $ H                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ He                             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Li                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Be                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ B                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ C                              : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ N                                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ O                                 : num  4 4 4 4 4 4 4 4 4 4 ...
##  $ F                                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Ne                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Na                                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Mg                                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Al                                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Si                                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ P                                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ S                                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Cl                                : num  0 0 0 0 0 0 0 0 0 0 ...
##   [list output truncated]
```

Train and Test Data

- The combined dataset is divided into train and test data.
- The train dataset contains 80% records whereas test dataset contains 20% records.
- The train dataset is used to train the models where as the test dataset is used for validating the performance of the model.
- Here, we are not using cross validation since it will be computationally expensive to perform k-fold cross validation on 21263 records.

```
# Indices for training data (80%)
train_indices <- round(0.80*dim(combined_data)[1])

# Training dataset
train <- combined_data[0:train_indices,]

# Testing dataset
test <- combined_data[train_indices:dim(combined_data)[1],]

# Dimensions of train data
dim(train)
```

```
## [1] 17010   170
```

Unique features

- We will use the training dataset for further exploration of data.
- Identify the unique features in the dataset and print them.
- The dataset contains 169 unique features.This indicates there is atleast 1 redundant feature.
- We will identify the redundant features in the feature reduction step and remove them for further analysis and model implementation.

```
# Get the unique features from the dataset
unique_features <- apply(train, 2, function(x) length(unique(x)))

#print the unique features
cat("The numbers of unique values for each attribute are:",length(unique_features), "\n\n")
```

```
## The numbers of unique values for each attribute are: 170
```

```
# Get a headshot of unique features
head(sort(unique_features, decreasing = T))
```

```
##               material    wtd_mean_atomic_mass   wtd_gmean_atomic_mass
##                  12587                   12287                   12287
```

```
## wtd_entropy_atomic_mass       wtd_gmean_Density         wtd_entropy_fie
##                   12242                   12232                   12214
```

Feature variance

- Get an overview of variance associated with teh features.
- We are going to use the var function to check the variance of the different features.

```
# Drop the last column (Materials)
train <- train[-dim(train)[2]]

# Check the variance of all features. Sort in descending order
variance_of_features  <- as.data.frame(as.table(sort(apply(train, 2, var), decreasing = T)))
names(variance_of_features) <- c("Feature", "Variance")
head(variance_of_features)
```

```
##             Feature Variance
## 1 wtd_gmean_Density 17199454
## 2     range_Density 15656398
## 3     gmean_Density 14546840
## 4  wtd_mean_Density 11471498
## 5      mean_Density  8606243
## 6 wtd_range_Density  6360871
```

- Get the list of features having zero variance.
- It can be observed that the variance of the elements (or features) He, Ar, Ne, Xe, Kr, Xe, Pm, Po and At is zero.
- If we look at the dataset their value is 0 for all the super conductors which indicates that they are not present in any super conductors.

```
# Let's find which variables have zero variance.
zero_variance <- which(apply(train, 2, var) == 0)

cat("Features whose variance is 0 are:\n\n")
```

```
## Features whose variance is 0 are:
```

```
print(zero_variance)
```

```
##  He  Ne  Ar  Kr  Xe  Pm  Po  At  Rn
##  84  92 100 118 136 143 166 167 168
```

Highly correlated features

- The check_correlation() function is created to identify the highly correlated features in the dataset. This function takes the dataset and the correlation cutoff as input and print the features whose correlation is greater than the threshold.
- In this project, the correlation cutoff for highly correlated features is set to 0.8.
- The training dataset is passed to the check_correlation() function and the cutoff correlation value is set to 0.8. The output of this function is the list of features whose correlation is greater that 0.8.

```
# Function to print the correlated features whose correlation > cutoff
check_correlation <- function(data, cutoff){
  cor_matrix <- cor(data)

  for (i in 1:nrow(cor_matrix)){
    correlations <-  which((abs(cor_matrix[i,i:ncol(cor_matrix)]) > cutoff) & (cor_matrix[i,i:ncol(cor_
```

```
    if(length(correlations)> 0){
      lapply(correlations,FUN =  function(x) (cat(paste(colnames(data)[i], ":",colnames(data)[x]), "\n"
    }
  }
}

#check_correlation(dplyr::select(train, -zero_variance), 0.8)
```

- Since the output of this function is a very big list, the following is the head of 10 highly correlated features:
  - number_of_elements : entropy_atomic_mass
  - number_of_elements : wtd_entropy_atomic_mass
  - number_of_elements : entropy_fie
  - number_of_elements : entropy_atomic_radius
  - number_of_elements : wtd_entropy_atomic_radius
  - number_of_elements : entropy_Density
  - number_of_elements : entropy_ElectronAffinity
  - number_of_elements : entropy_FusionHeat
  - number_of_elements : wtd_entropy_FusionHeat
  - number_of_elements : entropy_Valence

Features correlated to the target variable

- The cor() function is used to get the correlation matrix having correlation between different features.
- This correlation matrix is used to get the correlation of features with the critical temperature.
- This matrix is further used to filter the features whose correlation with the "critical temp" is less that 0.1, i.e., the features whose correlation with the target variable is very low.
- There are 70 features whose correlation with the "critical temp" is less than 0.1.

```
# Create a correlation matrix
corr_matrix <- cor(dplyr::select(train, -zero_variance))

# Create a dataframe that contains the correlation of features with the critical temperature
corr_df <- as.data.frame(corr_matrix[, "critical_temp"])

# Set the name of the column to "critical_temp"
names(corr_df) <- c("critical_temp")

# Add a column to store the absolute value of the "critical_temp" correlation coefficient
corr_df['absolute_correlation'] = abs(corr_df[,"critical_temp"])

# Get the features whose absolute value of correlation with the critical_temp is < 0.1
correlated_features_df <- as.data.frame(t(corr_df[corr_df[,"absolute_correlation"] < 0.1,]))
```

```r
# Get the names of those features
names_of_features <- names(correlated_features_df)

# Number of features whose correlation with the critical_temp is < 0.1
length(names_of_features)
```

```
## [1] 70
```

- Following is the list of features whose correlation with the *Critical Temperature* is less than 0.1.

```r
cat("Features whose correlation with the critical_temp is < 0.1 are:\n\n")
```

```
## Features whose correlation with the critical_temp is < 0.1 are:
```

```r
cat(names_of_features)
```

```
## gmean_fie mean_atomic_radius wtd_entropy_ThermalConductivity range_Valence H Li Be C N F Na Mg Al Si
```

Feature Reduction

Filter Method

In this section, we are reducing the set of features from the training set in order to reduce the data redundancy as well as improve the performance of the machine learning algorithm. The primary objective is to reduce the complexity of a model and make it easier to interpret. Feature selection also improves the accuracy of a model and also reduces overfitting.

Filter method is used in this step and features are selected on the basis of their statistical scores and correlation with the target variable. After reducing the set of features using the filter method, wrapper method using stepwise selection approach is used to obtain the best subset of features to be used in the model.

Drop the duplicate columns

- To reduce the redundancy in the dataset,the first step is to remove the duplicate features.
- The following code identifies the duplicated columns in the dataset and removes them.
- duplicated(t(train)) gives the list of duplicated features. These features are removed from the dataset.
- It is observed that there were 10 duplicated features in the dataset.

```
# Drop the duplicate columns from the dataset
train <- train[!duplicated(t(train))]

# Check the dimensions of the updated dataset
dim(train)
```

```
## [1] 17010   160
```

Remove columns with near zero or near zero variance

- Some variables do not contain much information. For example:
  - Constants: They do not have any variance in their values.
  - Nearly constant features: They have low variation in values.
- In the data exploration step, we have identified that the elements (or features) He, Ar, Ne, Xe, Kr, Xe, Pm, Po and At have 0 variance. These features are removed from our training dataset.
- The new dataset is reduced to 154 features.

```
# Remove features that have 0 variance
train_1 <- select(train, -zero_variance)

# Check the dimensions of the updated dataset
dim(train_1)
```

```
## [1] 17010   154
```

- Features that have very low variance are nearly constant and do not contain substantial information. Since they contain less information, they tend not to have any impact on our model.
- Hence, extremely low variance features are removed from the dataset prior to modelling.
- The nearZeroVar() function from the *caret* library is used to remove the the features from the dataset that have extremely low variance.
- It can be observed that dimensionality of the dataset is reduced significantly to 85 thereby removing 69 features.

```
# Identify near zero variance predictors: remove_cols
remove_cols <- nearZeroVar(train_1, names = TRUE)

# Get all column names from the train set: all_cols
```

```r
all_cols <- names(train_1)

# Remove from data: train set
train_2 <- train_1[ , setdiff(all_cols, remove_cols)]

# Check the dimensions of the updated dataset
dim(train_2)
```

```
## [1] 17010    85
```

Remove highly correlated features

- To improve the performance of the model and reduce it's complexity, the highly correlated features are identified using the findCorrelation() function.
- This function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove pair-wise correlations.
- Since highly correlated features impart same information to the model and can be redundant, these features are removed from the dataset to improve the interpretability of the model.
- In this part, the features that have correlation greater than 80% are removed.
- Following is the list of features whose correlation is greater than 0.8 and should be removed from our list. The pair-wise list is identified in the data exploration.

```r
# Create a correlation matrix
corr_matrix <- cor(train_2)

# Get the list of features that are highly corrrelated
highly_correlated_features <- findCorrelation(corr_matrix, cutoff=0.8, names = T)

cat("List of redundant features:\n\n")
```

```
## List of redundant features:
```

```r
print(highly_correlated_features)
```

```
##  [1] "range_fie"                  "wtd_entropy_atomic_radius"
##  [3] "range_atomic_radius"        "wtd_std_fie"
##  [5] "wtd_std_atomic_radius"      "wtd_entropy_atomic_mass"
##  [7] "entropy_Valence"            "entropy_fie"
##  [9] "wtd_entropy_Valence"        "std_fie"
## [11] "number_of_elements"         "entropy_atomic_radius"
## [13] "wtd_gmean_Density"          "wtd_std_ThermalConductivity"
## [15] "std_atomic_radius"          "gmean_Density"
## [17] "wtd_gmean_Valence"          "entropy_atomic_mass"
## [19] "wtd_mean_Valence"           "wtd_entropy_FusionHeat"
## [21] "gmean_Valence"              "entropy_ElectronAffinity"
## [23] "wtd_gmean_atomic_radius"    "entropy_FusionHeat"
## [25] "wtd_mean_fie"               "wtd_mean_Density"
## [27] "range_atomic_mass"          "entropy_Density"
## [29] "wtd_mean_atomic_radius"     "range_ElectronAffinity"
## [31] "wtd_gmean_atomic_mass"      "wtd_entropy_fie"
## [33] "wtd_std_ElectronAffinity"   "wtd_std_atomic_mass"
## [35] "gmean_FusionHeat"           "wtd_gmean_FusionHeat"
## [37] "gmean_ThermalConductivity"  "wtd_mean_atomic_mass"
## [39] "wtd_mean_FusionHeat"        "wtd_range_Density"
## [41] "gmean_atomic_mass"          "range_Density"
## [43] "gmean_atomic_radius"        "wtd_std_Density"
```

```
## [45] "wtd_mean_ElectronAffinity"    "wtd_std_FusionHeat"
## [47] "wtd_mean_ThermalConductivity" "mean_ElectronAffinity"
## [49] "gmean_fie"                    "wtd_std_Valence"
## [51] "std_Valence"
```

- Check the dimensionality of the dataset after removing highly correlated features.
- The updated list of features contain 34 columns.

```
# Remove highly correlated features
train_3 <- dplyr::select(train_2, -highly_correlated_features)

# Check the dimensions of the updated dataset
dim(train_3)
```

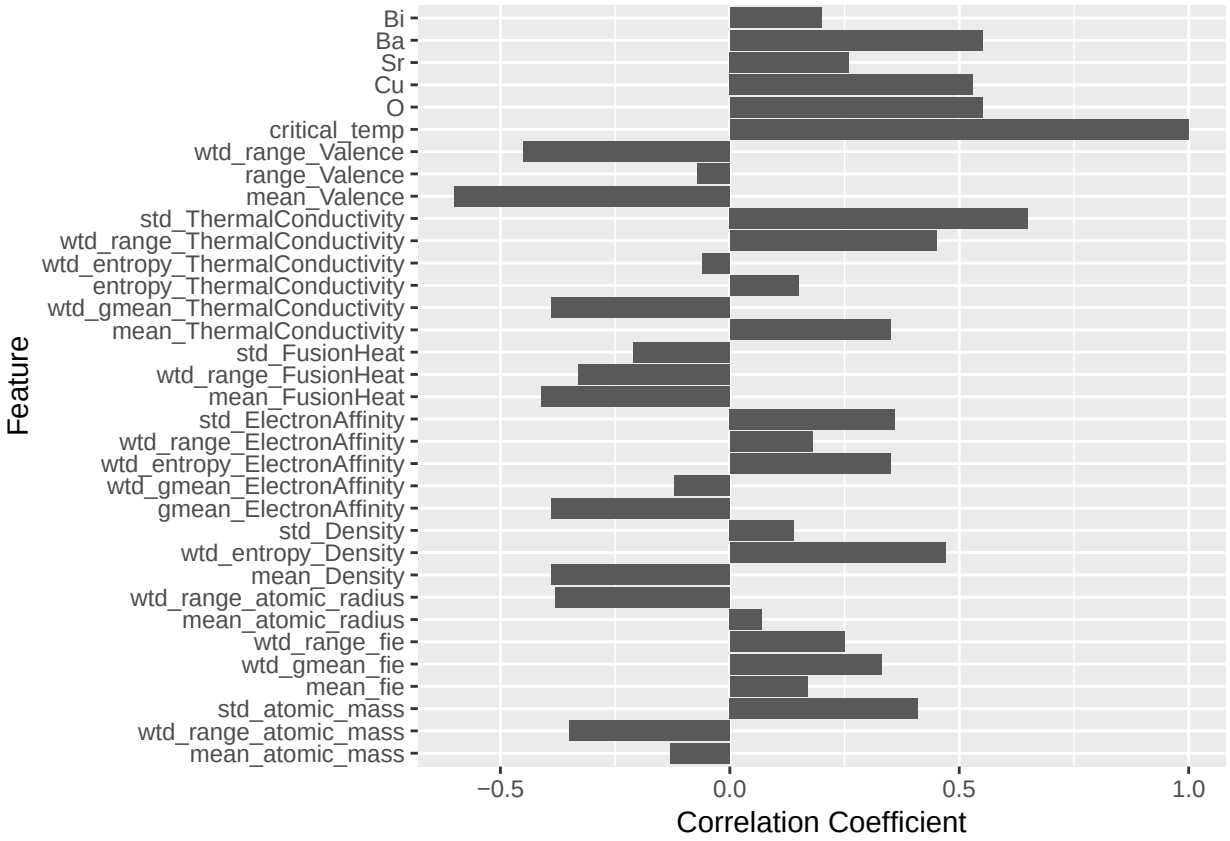
```
## [1] 17010    34
```

Drop features that have low correlation with the target variable

- Feature having low correlation with the target variable do not improve the prediction capability of the model.
- This implies that there is no information in the feature that predicts the target.
- The following code identifies the features whose correlation with the "Critical temperature" is greater than 0.1.

```
# Create a new correlation matrix
corr_matrix_2 <- round(cor(train_3), 2)

# Create a dataframe that contains the correlation of features with the critical temperature
corr_df <- as.data.frame(corr_matrix_2[, "critical_temp"])

# Set the name of the column to "critical_temp"
names(corr_df) <- c("critical_temp")

# Add a column to store the absolute value of the "critical_temp" correlation coefficient
corr_df['absolute_correlation'] = abs(corr_df[,"critical_temp"])
```

```
# Reduced set of features
feature <- names(as.data.frame(t(corr_df)))

# Correlation coeff with critical_temp
coef <- corr_df[,"critical_temp"]

# Dataframe of feature and correlation coeff
feature_df <- data.frame(Feature = feature, Coef = coef )

# Visualze the plot of features and its correlation with critical_temp
ggplot(feature_df, aes(x = factor(Feature, levels = Feature), y = Coef)) +
    geom_bar(stat = "identity") +
    coord_flip() +
    xlab("Feature") +
    ylab("Correlation Coefficient")
```



- The features whose correlation with "Critical temperature" is greater than 0.1 are as follows.
- There are 30 features relevant for predicting the "Critical temperature" of a super conductor. We will use these features for implementing the model.

```
# Get the features whose absolute value of correlation with the critical_temp is > 0.1
correlated_features_df <- as.data.frame(t(corr_df[corr_df[,"absolute_correlation"] > 0.1,]))

# Get the names of those features
names_of_features <- names(correlated_features_df)

# Print the features whose absolute value of correlation with the critical_temp is > 0.1
names(dplyr::select(correlated_features_df, -c("critical_temp")))
```

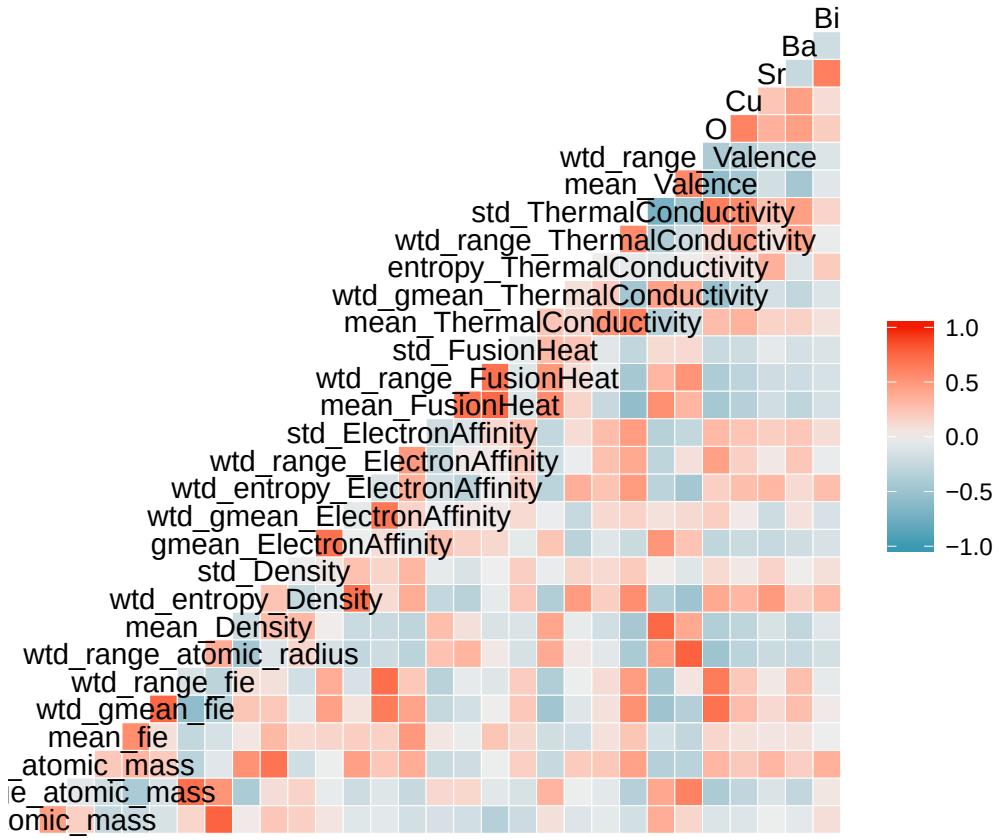
```
##  [1] "mean_atomic_mass"              "wtd_range_atomic_mass"
##  [3] "std_atomic_mass"               "mean_fie"
##  [5] "wtd_gmean_fie"                 "wtd_range_fie"
##  [7] "wtd_range_atomic_radius"       "mean_Density"
##  [9] "wtd_entropy_Density"           "std_Density"
## [11] "gmean_ElectronAffinity"        "wtd_gmean_ElectronAffinity"
## [13] "wtd_entropy_ElectronAffinity"  "wtd_range_ElectronAffinity"
## [15] "std_ElectronAffinity"          "mean_FusionHeat"
## [17] "wtd_range_FusionHeat"          "std_FusionHeat"
## [19] "mean_ThermalConductivity"      "wtd_gmean_ThermalConductivity"
## [21] "entropy_ThermalConductivity"   "wtd_range_ThermalConductivity"
## [23] "std_ThermalConductivity"       "mean_Valence"
## [25] "wtd_range_Valence"             "O"
## [27] "Cu"                            "Sr"
```

```
## [29] "Ba"                              "Bi"
```

Final subset of features

- Create a dataframe which contains the final subset of features.
- The final dataset obtained after the filteration step contains 30 features which are used for moel building.

```
# Create a new dataframe of features whose correlation with the critical_temp is > 0.1
train_4 <- select(train_3, names_of_features)

# Visualize features whose correlation with the critical_temp is > 0.1
ggcorr(dplyr::select(train_4, -c("critical_temp")))
```



Wrapper Method

Based on the features obtained from the Filter method, the wrapper method selects the best subset of features and train the model using them. Recursive feature elimination or step-wise feature selection is used to select the optimal subset of features.

The train() function from the *caret* package is used to perform the stepwise selection of features.In this method, the tuning parameter *nvmax* is used to obtain the optimum number of features to be incorporated in the model to obtain better predictions. Since we have 30 set of features, the range of nvmax is 1:30.

The result of this function gives the optimum number of features to be incorporated in the model.

K-fold Cross validation

- The trainControl() function controls the K-fold cross validation.
- In this method, 10-fold cross validation is used to estimate the average prediction error for every combination of predictors.

```
# Set up repeated k-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)
```

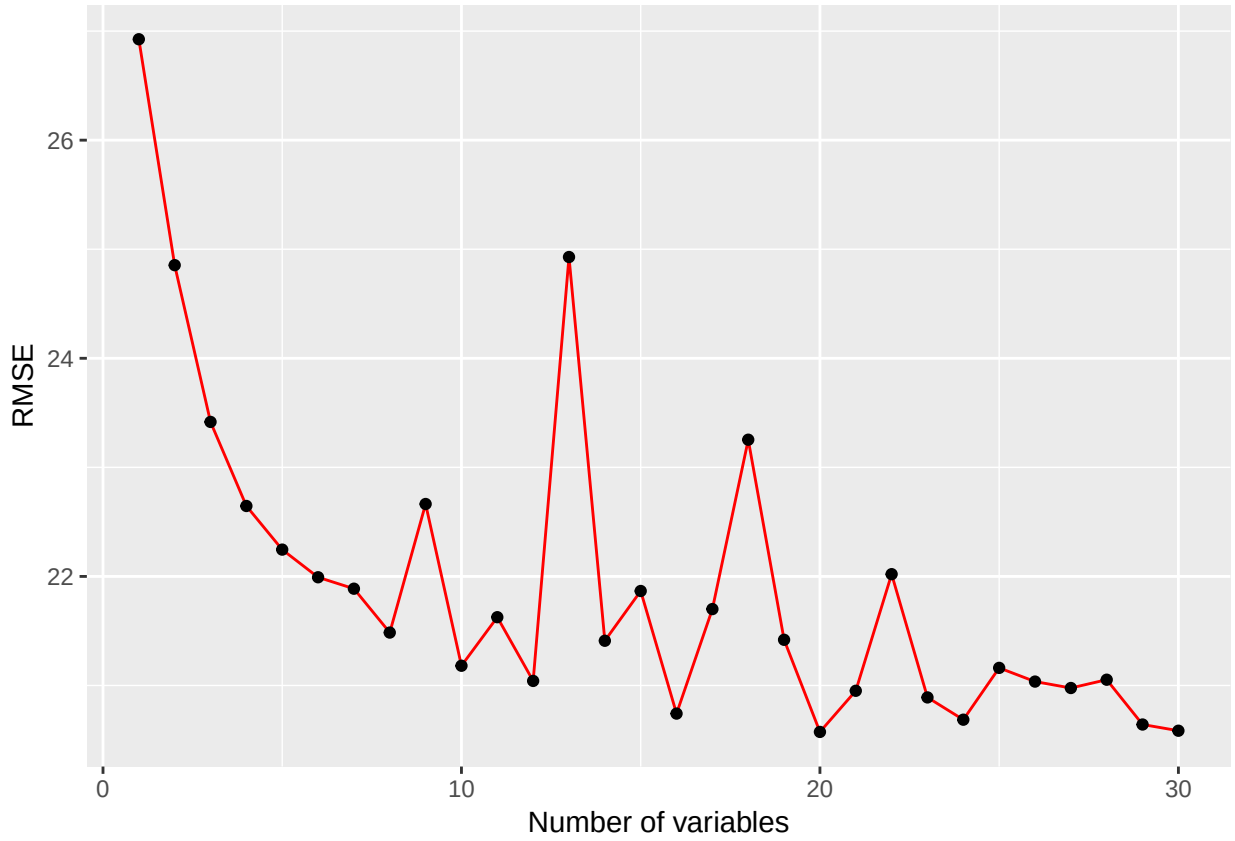
Step-wise feature selection

- The "leapSeq" method to perform step-wise selection to fit linear regression.

```
# Set seed for reproducibility
set.seed(123)

# Train the model
step.model <- train(critical_temp ~ .,
                    data = train_4,
                    method = "leapSeq",
                    tuneGrid = data.frame(nvmax = 1:30),
                    trControl = train_control
                    )

# Model results
model_result <- step.model$results
```

- The above model result is used to get the optimal number of features in the dataset.
- From thhe below graph it is observes that th e optimal number of features is 20.

```
# Plot Model accuracy
ggplot(data=model_result, aes(x=nvmax, y=RMSE), ylim = c(0,100)) +
    geom_line(color="red")+
    geom_point() +
    labs(x = "Number of variables", y = "RMSE")
```




```
# Select the best model with lowest RMSE (optimal number of variables)
step.model$bestTune
```

```
##    nvmax
## 20    20
```

- The best tune of the model is obtained at $nvmax = 20$.
- Therefore, the best 20 features to be incorporated in the model are as follows.
- These features are the most appropriate ones for model devdelpoment.

```
# Final model coefficients
cat(names(coef(step.model$finalModel, 20)))
```

```
## (Intercept) mean_atomic_mass wtd_range_atomic_mass std_atomic_mass mean_fie wtd_range_fie mean_Densi
```

Model Development

The features obtained from the feature reduction step are utilised for the development of model. In this section 3 types of models are developed including:

- Linear Regression Model
- Random Forest Model
- Gradient Boosted Model

Linear Regression Model

- The linear model is build using the 20 features obtained from the wrapper method of feature reduction step.
- The summary statistics show that the model has a significantly low p-value and a R2 value of 0.664 on the training set.

```
# Get the linear model
linear_model_1 <- lm(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + mean_
                     mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Ele
                     mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                     wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                     std_ThermalConductivity + O + Ba + Bi,
                   data = train_4)

summary(linear_model_1)
```

```
##
## Call:
## lm(formula = critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass +
##     std_atomic_mass + mean_fie + wtd_range_fie + mean_Density +
##     gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_ElectronAffinity +
##     mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat +
##     mean_ThermalConductivity + wtd_gmean_ThermalConductivity +
##     entropy_ThermalConductivity + wtd_range_ThermalConductivity +
##     std_ThermalConductivity + O + Ba + Bi, data = train_4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -243.693  -11.220    0.511   12.836  107.387
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -29.863522   2.457376 -12.153  < 2e-16 ***
## mean_atomic_mass                0.190181   0.012767  14.896  < 2e-16 ***
## wtd_range_atomic_mass          -0.104692   0.007474 -14.008  < 2e-16 ***
## std_atomic_mass                 0.156152   0.011090  14.080  < 2e-16 ***
## mean_fie                        0.053311   0.002705  19.710  < 2e-16 ***
## wtd_range_fie                  -0.015457   0.001407 -10.983  < 2e-16 ***
## mean_Density                   -0.002455   0.000138 -17.788  < 2e-16 ***
## gmean_ElectronAffinity          0.111119   0.013040   8.521  < 2e-16 ***
## wtd_gmean_ElectronAffinity     -0.265359   0.011031 -24.055  < 2e-16 ***
## wtd_entropy_ElectronAffinity  -22.199872   1.203679 -18.443  < 2e-16 ***
## mean_FusionHeat                 0.307401   0.031384   9.795  < 2e-16 ***
## wtd_range_FusionHeat            0.121140   0.025335   4.782 1.75e-06 ***
## std_FusionHeat                 -0.551351   0.037252 -14.801  < 2e-16 ***
## mean_ThermalConductivity        0.078465   0.012132   6.467 1.02e-10 ***
```

```
## wtd_gmean_ThermalConductivity  -0.194434   0.010509 -18.501  < 2e-16 ***
## entropy_ThermalConductivity    26.314331   0.896356  29.357  < 2e-16 ***
## wtd_range_ThermalConductivity   0.188072   0.007240  25.979  < 2e-16 ***
## std_ThermalConductivity         0.136679   0.011944  11.444  < 2e-16 ***
## O                               1.110724   0.078513  14.147  < 2e-16 ***
## Ba                              9.337363   0.208746  44.731  < 2e-16 ***
## Bi                              4.951275   0.312484  15.845  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.53 on 16989 degrees of freedom
## Multiple R-squared:  0.6644, Adjusted R-squared:  0.664
## F-statistic:  1681 on 20 and 16989 DF,  p-value: < 2.2e-16
```

- Using this model to predict the "Critical temperature" of the superconductor on the test set gives an RMSE of 16.99 and R2 of 0.49.

```
# Predict the "Critical temperature" using linear model
predictions_lm1 <- predict(linear_model_1, test)

# Model performance
data.frame(
  RMSE = RMSE(predictions_lm1, test$critical_temp),
  R2 = R2(predictions_lm1, test$critical_temp)
)
```

```
##       RMSE        R2
## 1 16.99911 0.4981891
```

- The filteration method does not take into account the multicollinearity among the features in the model.
- Hence, we identify the multicollinearity in the model using the VIF (Variance Inflation Factor).
- The features having VIF >10 are highly collinear with the other parameters in the model and should be removed from the set of features.
- The vif() function is used to find the Variance Inflation Factor of each feature.

```
print(car::vif(linear_model_1))
```

```
##              mean_atomic_mass         wtd_range_atomic_mass
##                      5.754406                      1.793277
##               std_atomic_mass                     mean_fie
##                      2.005149                      1.716917
##                 wtd_range_fie                  mean_Density
##                      4.039359                      6.611650
##        gmean_ElectronAffinity    wtd_gmean_ElectronAffinity
##                      5.300851                      4.507465
##  wtd_entropy_ElectronAffinity              mean_FusionHeat
##                      4.184253                      5.268640
##          wtd_range_FusionHeat                std_FusionHeat
##                      3.651318                      4.609268
##      mean_ThermalConductivity wtd_gmean_ThermalConductivity
##                      7.699809                      7.505500
##   entropy_ThermalConductivity wtd_range_ThermalConductivity
##                      3.020660                      3.986942
##       std_ThermalConductivity                             O
##                     19.583621                      3.712040
##                            Ba                            Bi
##                      1.882282                      1.675814
```

- The features whose VIF>10 are removed and the summary statistics of the model are observed.
- It seems that the residual standard error has increased slightly on the training set.
- However, comparing the performance on the test datset, the R2 value has increased from 0.498 to 0.506.
- Thus, the new linear model without the feature whose vif>10 has better performance of the training set.

```
# Get the linear model
linear_model_2 <- lm(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + mean_
                     mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Ele
                     mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                     wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                     O + Ba + Bi,
                   data = train_4)

summary(linear_model_2)
```

```
##
## Call:
## lm(formula = critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass +
##     std_atomic_mass + mean_fie + wtd_range_fie + mean_Density +
##     gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_ElectronAffinity +
##     mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat +
##     mean_ThermalConductivity + wtd_gmean_ThermalConductivity +
##     entropy_ThermalConductivity + wtd_range_ThermalConductivity +
##     O + Ba + Bi, data = train_4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -240.829  -11.398    0.382   12.949  111.465
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -2.892e+01  2.465e+00 -11.730  < 2e-16 ***
## mean_atomic_mass               2.115e-01  1.268e-02  16.679  < 2e-16 ***
## wtd_range_atomic_mass         -1.142e-01  7.456e-03 -15.322  < 2e-16 ***
## std_atomic_mass                1.751e-01  1.101e-02  15.908  < 2e-16 ***
## mean_fie                       4.979e-02  2.697e-03  18.456  < 2e-16 ***
## wtd_range_fie                 -1.013e-02  1.333e-03  -7.599 3.13e-14 ***
## mean_Density                  -2.652e-03  1.374e-04 -19.296  < 2e-16 ***
## gmean_ElectronAffinity         7.826e-02  1.277e-02   6.129 9.03e-10 ***
## wtd_gmean_ElectronAffinity    -2.548e-01  1.103e-02 -23.093  < 2e-16 ***
## wtd_entropy_ElectronAffinity  -1.498e+01  1.029e+00 -14.558  < 2e-16 ***
## mean_FusionHeat                3.068e-01  3.150e-02   9.738  < 2e-16 ***
## wtd_range_FusionHeat           1.021e-01  2.538e-02   4.022 5.79e-05 ***
## std_FusionHeat                -5.442e-01  3.739e-02 -14.556  < 2e-16 ***
## mean_ThermalConductivity       1.967e-01  6.390e-03  30.776  < 2e-16 ***
## wtd_gmean_ThermalConductivity -2.751e-01  7.822e-03 -35.173  < 2e-16 ***
## entropy_ThermalConductivity    2.057e+01  7.457e-01  27.590  < 2e-16 ***
## wtd_range_ThermalConductivity  2.365e-01  5.894e-03  40.130  < 2e-16 ***
## O                              1.110e+00  7.881e-02  14.089  < 2e-16 ***
## Ba                             9.216e+00  2.093e-01  44.039  < 2e-16 ***
## Bi                             5.019e+00  3.136e-01  16.002  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 20.61 on 16990 degrees of freedom
## Multiple R-squared:  0.6618, Adjusted R-squared:  0.6614
## F-statistic:  1750 on 19 and 16990 DF,  p-value: < 2.2e-16
```

```
# Predict the "Critical temperature" using linear model
predictions_lm2 <- predict(linear_model_2, test)

# Model performance
data.frame(
  RMSE = RMSE(predictions_lm2, test$critical_temp),
  R2 = R2(predictions_lm2, test$critical_temp)
)
```

```
##       RMSE        R2
## 1 16.85935 0.5059919
```

Random Forest Model

- Random Forest is another machine learning algoritm for performing regression tasks.
- In thi step, a Random Forest model with default parameters is implemented on the training set.
- The default number of trees in this case is 500 and the number of variables in each split (mtry) is 6.

```
set.seed(1234)

# Default model
model_rf1 <- randomForest(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + r
                   mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Elec
                   mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                   wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                   std_ThermalConductivity + O + Ba + Bi, data = train_4)

# Print the results
print(model_rf1)
```

```
##
## Call:
##  randomForest(formula = critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass +      std_atomic_ma
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 92.4625
##                     % Var explained: 92.63
```

- The performance of the model is tested on the training set.
- The model records an RMSE of 13.78 and R2 of 0.675.

```
# Predict the RMSE using model 1
RMSE_model_rf1 = RMSE(predict(model_rf1, test), test$critical_temp)

# Predict the R-squared using model 1
R2_model_rf1 = R2(predict(model_rf1, test), test$critical_temp)


# Model performance
data.frame(
  RMSE = RMSE_model_rf1,
```

```
  R2 =  R2_model_rf1
)
```

```
##       RMSE        R2
## 1 13.78596 0.6752211
```

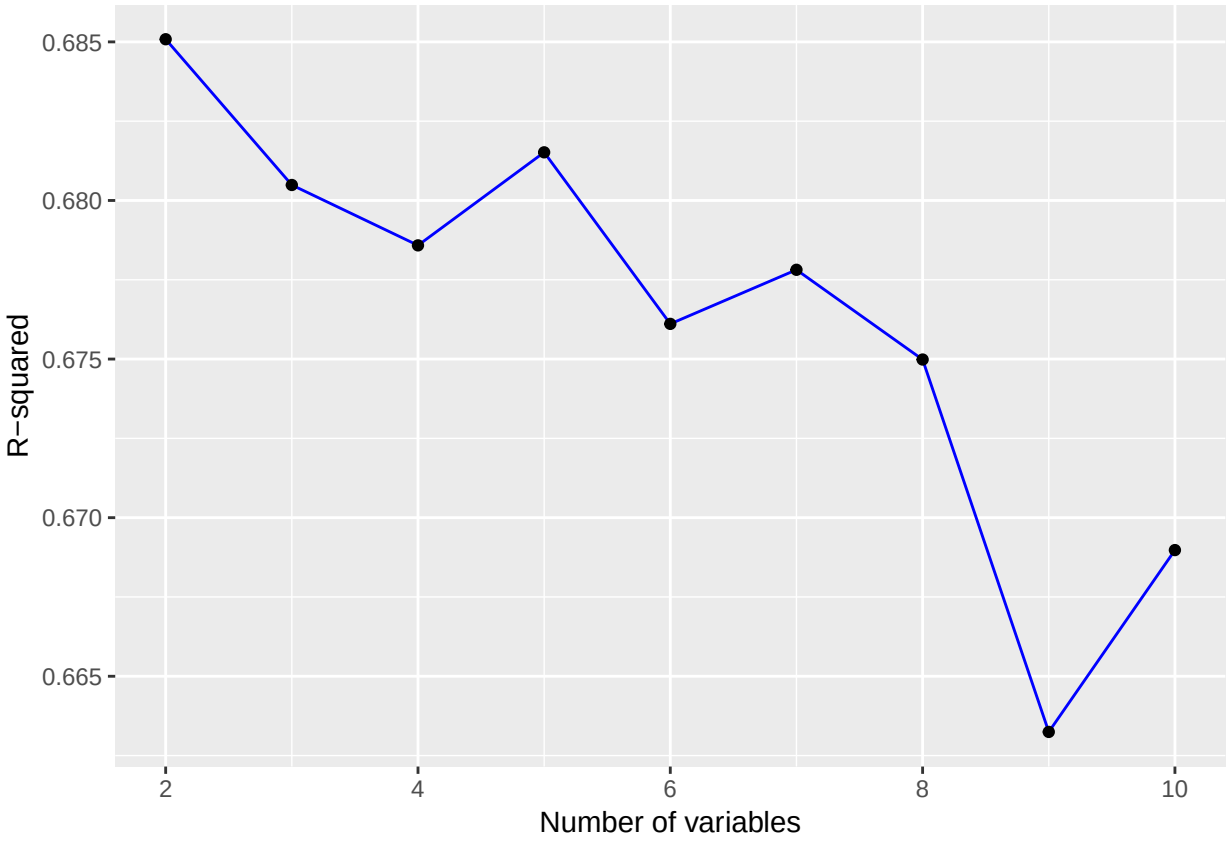
Tuning the parameters of Random Forest model

- To tune the model, it is tested with the values of mtry from 1 to 10.
- Observing the performance of the model for each value of mtry, the R2 value is maximum at mtry = 2.
- Hence, it can be concluded that the optimum number of variables tried at each split is 2.

```
set.seed(1234)

# Number of variables sampled at each split
mtry_seq <- c(2:10)

# RMSE for each mtry
RMSE = c()

# R-squared for each mtry
R2 = c()

# Loop to search the best mtry
for (each in mtry_seq)
{
    model_rf <- randomForest(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass
                    mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Ele
                    mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                    wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                    std_ThermalConductivity + O + Ba + Bi, data = train_4, ntree = 500, mtry = each)

    # Predict the "Critical temperature" using linear model
    predictions_rf <- predict(model_rf, test)

    RMSE = c(RMSE, RMSE(predictions_rf, test$critical_temp))
    R2 = c(R2, R2(predictions_rf, test$critical_temp))
}
```

- Check the performance of the model.

```
# Model performance
rf_model_result <- data.frame(
    mtry = mtry_seq,
    RMSE = RMSE,
    R2 = R2
)

# Visualize the Model performance
ggplot(data=rf_model_result, aes(x=mtry, y=R2), ylim = c(0,1)) +
    geom_line(color="blue")+
    geom_point() +
    labs(x = "Number of variables", y = "R-squared")
```



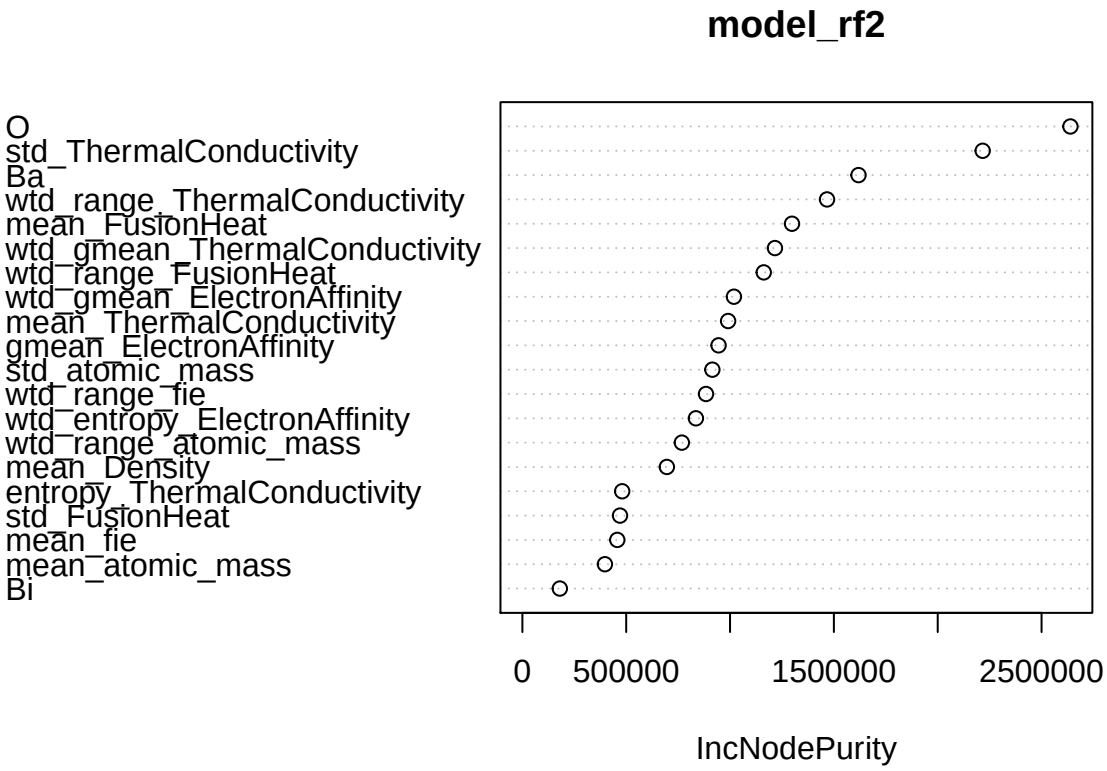
- The model with the highest R2 value is selected and it performance is tested.
- It is observed that the value of R2 has improved from 0.675 to 0.684.

```
set.seed(1293)

# Model with features is chosen for each iteration (mtry = 2)
model_rf2 <- randomForest(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + m
                  mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Elec
                  mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                  wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalConduc
                  std_ThermalConductivity + O + Ba + Bi, data = train_4, ntree = 500, mtry = 2)

# Print the results
print(model_rf2)
```

```
##
## Call:
##  randomForest(formula = critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass +      std_atomic_ma
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 95.08746
##                     % Var explained: 92.42
```

- The variable importance is used to identify the top 20 important features.

```r
# To check important variables
varImpPlot(model_rf2)
```



- Implement a random forest model with the top 20 important features and compare the performance of the models.
- The second random forest model has the maximum value of R2 and is comparatively a better model.

```r
# Considering top 20 features to fine tune random forest
top20_rf_features <- as.data.frame(t((importance(model_rf2, type = 1))))

set.seed(1233)

# Model with top 20 features
model_rf3 <- randomForest(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass +
                            mean_fie + wtd_range_fie + mean_Density + gmean_ElectronAffinity + wtd_gmea
                            wtd_entropy_ElectronAffinity + mean_FusionHeat + wtd_range_FusionHeat + st
                            mean_ThermalConductivity + wtd_gmean_ThermalConductivity + entropy_Thermal
                            wtd_range_ThermalConductivity + std_ThermalConductivity + O + Ba + Bi, data


# Predict the RMSE using model 2
RMSE_model_rf2 = RMSE(predict(model_rf2, test), test$critical_temp)

# Predict the R-squared using model 2
R2_model_rf2 = R2(predict(model_rf2, test), test$critical_temp)

# Predict the RMSE using model 3
```

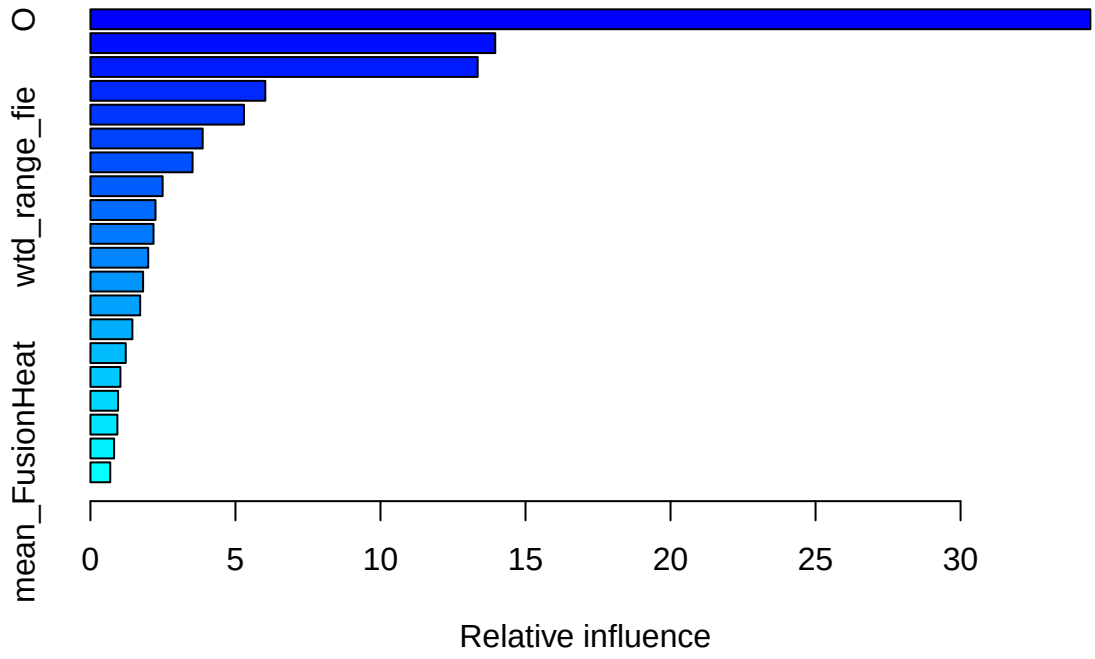
```
RMSE_model_rf3 = RMSE(predict(model_rf3, test), test$critical_temp)

# Predict the R-squared using model 3
R2_model_rf3 = R2(predict(model_rf3, test), test$critical_temp)

# Random forest model comparison
rf_model_comparion <- data.frame(
    Model = c(1:3),
    RMSE = c(RMSE_model_rf1, RMSE_model_rf2, RMSE_model_rf3),
    R2 = c(R2_model_rf1, R2_model_rf2, R2_model_rf3)
)

# Compare the performance of random forest models
rf_model_comparion
```

```
##   Model     RMSE        R2
## 1     1 13.78596 0.6752211
## 2     2 13.50491 0.6849245
## 3     3 13.74985 0.6753302
```

Gradient Boosted Modeling

- In this model, R's GBM (Gradient Boosted Modeling) package is used to implement the boosting model.
- The number of trees in GBM are smaller than the random forest.
- This model uses *gaussian distribution* for the residual error loss.
- The default number of trees are 10,000 and the learning rate is 0.01. Since there are 21,000+ observations, the interaction depth is set to 8.
- The summary statistics of the model are as follows. It gives a variable importance plot for the gbm model.

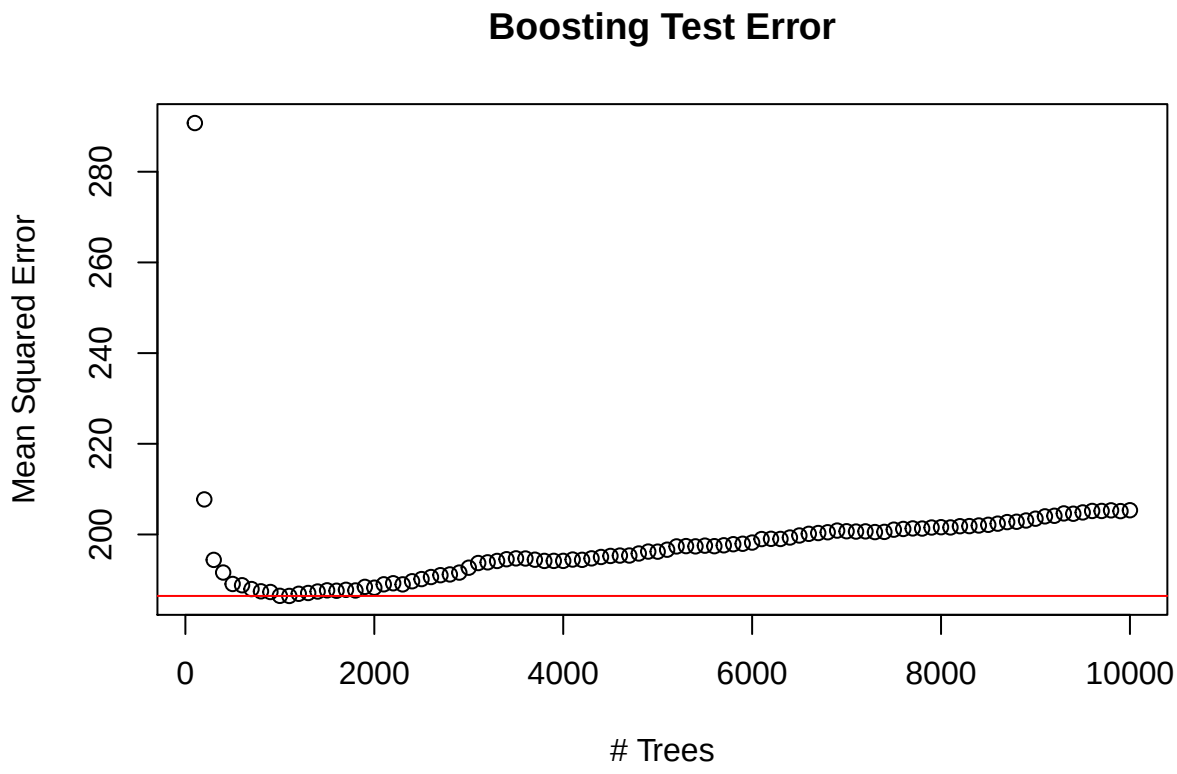
```
model_gbm = gbm(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + mean_fie +
                  mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Ele
                  mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                  wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                  std_ThermalConductivity + O + Ba + Bi,
                  data = train_4, distribution = "gaussian", n.trees = 10000, shrinkage = 0.01, intera

summary(model_gbm)
```




```
##                                                          var    rel.inf
## O                                                          O 34.4743537
## wtd_gmean_ThermalConductivity wtd_gmean_ThermalConductivity 13.9561862
## std_ThermalConductivity             std_ThermalConductivity 13.3504365
## wtd_gmean_ElectronAffinity       wtd_gmean_ElectronAffinity  6.0293831
## Ba                                                      Ba  5.2930832
## gmean_ElectronAffinity               gmean_ElectronAffinity  3.8694641
## std_atomic_mass                             std_atomic_mass  3.5210545
## wtd_range_fie                                 wtd_range_fie  2.4887595
## wtd_range_ThermalConductivity wtd_range_ThermalConductivity  2.2433473
## wtd_entropy_ElectronAffinity   wtd_entropy_ElectronAffinity  2.1749550
## wtd_range_atomic_mass                 wtd_range_atomic_mass  1.9926272
## mean_Density                                   mean_Density  1.8161702
## mean_ThermalConductivity           mean_ThermalConductivity  1.7144237
## entropy_ThermalConductivity     entropy_ThermalConductivity  1.4458223
## wtd_range_FusionHeat                   wtd_range_FusionHeat  1.2169804
## mean_atomic_mass                           mean_atomic_mass  1.0331124
## mean_fie                                           mean_fie  0.9520590
## std_FusionHeat                               std_FusionHeat  0.9290732
## Bi                                                      Bi  0.8156960
## mean_FusionHeat                             mean_FusionHeat  0.6830126
```

- Predicting the target variable on the test set using this model, the performance is tested for number of trees ranging from 100 to 10,000.
- It can be observed from the boosting test error graph that the minimum test error is observed for number of trees = 1100.

```r
# Sequence of number of trees
n.trees = seq(from = 100, to = 10000, by = 100)

# prediction matrix
predmat = predict(model_gbm, newdata = test, n.trees = n.trees)

# Boosting error graph
boost.err = with(test, apply( (predmat - critical_temp)^2, 2, mean) )
plot(n.trees, boost.err, ylab = "Mean Squared Error", xlab = "# Trees", main = "Boosting Test Error")
abline(h = min(boost.err), col = "red")
```



- Check the number of trees corresponding to the minimun boosting error.

```r
# Dataframe to store error corresponding to the number of trees
gbm_error <- data.frame(
    Number_of_Trees = n.trees,
    Test_Error = boost.err

)

gbm_error[gbm_error$Test_Error == min(gbm_error[,"Test_Error"]), ]
```

```
##      Number_of_Trees Test_Error
## 1100            1100   186.4439
```

- Thus, the best gbm model is obtained for nummber of trees = 1100.
- Check the performance of the model corresponding to n.tree = 1100.

- The RMSE of the model is 13.665 and the r2 is 0.661.

```
gbm_model2 <- gbm(critical_temp ~ mean_atomic_mass + wtd_range_atomic_mass + std_atomic_mass + mean_fie
                  mean_Density + gmean_ElectronAffinity + wtd_gmean_ElectronAffinity + wtd_entropy_Ele
                  mean_FusionHeat + wtd_range_FusionHeat + std_FusionHeat + mean_ThermalConductivity +
                  wtd_gmean_ThermalConductivity + entropy_ThermalConductivity + wtd_range_ThermalCondu
                  std_ThermalConductivity + O + Ba + Bi,
                  data = train_4, distribution = "gaussian", n.trees = 1100, shrinkage = 0.01, interac

# Model performance
data.frame(
  RMSE = RMSE(predict(gbm_model2, test, n.trees = 1100), test$critical_temp),
  R2 = R2(predict(gbm_model2, test, n.trees = 1100), test$critical_temp)
)
```

```
##       RMSE        R2
## 1 13.82742 0.6550567
```

Model Comparison

Three different types of regression models were implemented to perform regression analysis on the dataset of super conductors. The following models were implemented for predicting the "Critical temperature" of a super conductor:

- Linear Model
- Random Forest Model
- Gradient Boosted Model

The mertics used to compare the performance of the models was the RMSE value and the R2 score.

Performance metrics:

```
# Model performance
performance <- data.frame(
  Model = c("Linear Model", "Random Forest", "Gradient Boosted Model"),
  RMSE = c(RMSE(predictions_lm2, test$critical_temp), RMSE_model_rf2, RMSE(predict(gbm_model2, test, n.
  R2 = c(R2(predictions_lm2, test$critical_temp), R2_model_rf2, R2(predict(gbm_model2, test, n.trees =
)

# Print model performance
performance
```

```
##                    Model     RMSE        R2
## 1           Linear Model 16.85935 0.5059919
## 2          Random Forest 13.50491 0.6849245
## 3 Gradient Boosted Model 13.82742 0.6550567
```

- From the above comparison, the random forest model has the best metrics for RMSE as well as R2. The random forest model has the lowest root mean square and has the highest R2 score.
- Hence it is the best model amongst all.
- The second best performing model is the gradient boosted model with an RMSE of 13.63 and R2 of 0.66.
- The linear model is the least performing model among the 3. with RMSE of 16.85935 and R2 of 0.5059919.

Conclusion

- Regression models were successfully created to predict the critical temperature of superconductors using features derived from the properties of the elements in the superconductors.
- The initial dataset contained 80 features. These features were combined with the elements to infer the most dominant elements in the superconductors and their correlation to the Critical temperature.
- The total number of features was 167 which were reduced to a final subset of 20 significant features. The filter and wrapper feature reduction techniques were used to reduce the features.
- The final set of features used for predicting the critical temperature are: *mean_atomic_mass, wtd_range_atomic_mass, std_atomic_mass, mean_fie, wtd_range_fie, mean_Density, gmean_ElectronAffinity, wtd_gmean_ElectronAffinity, wtd_entropy_ElectronAffinity, mean_FusionHeat, wtd_range_FusionHeat, std_FusionHeat, mean_ThermalConductivity, wtd_gmean_ThermalConductivity, entropy_ThermalConductivity, wtd_range_ThermalConductivity, std_ThermalConductivity, O, Ba, Bi.*
- Three regression models were implemented for the regression task including the linear model, random forest model and the gradient boosted model.
- Comparing the performance of the 3 models, the random forest model has the best performance metrics.
- The Best model has **RMSE: 13.50** and **R2: 0.68**.

Reference

1. https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2
2. http://rpubs.com/Mentors_Ubiqum/Zero_Variance
3. https://campus.datacamp.com/courses/machine-learning-toolbox/preprocessing-your-data?ex=13
4. https://www.rdocumentation.org/packages/caret/versions/6.0-84/topics/findCorrelation
5. https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/
6. https://rpubs.com/sediaz/Correlations
7. http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/154-stepwise-regression-essentials-in-r/
8. http://www.sthda.com/english/articles/39-regression-model-diagnostics/160-multicollinearity-essentials-and-vif-in-r/
9. https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-s
10. https://www.r-bloggers.com/how-to-implement-random-forests-in-r/
11. https://www.datacamp.com/community/tutorials/decision-trees-R