FIT5149 Assignment - 1 :- Predicting critical temperature of superconductors

Student information

- Family Name: Mohamed Hayath
- Given Name: Mohaab Hassan
- Student ID: 29626889
- Student email: mmoh0064@student.monash.edu

Programming Language: R 3.5.1 in Jupyter Notebook

R Libraries used:

- ggplot2
- dplyr
- psych
- reshape2
- RFmarkerDetector
- GGally
- corrplot
- caret
- pedometrics
- car
- mctest
- randomForest
- xgboost
- saeRobust
- ggcorrplot
- tidyverse
- dvisc
- glmnet ## Table of Contents

1. Introduction

In the first assignment of this unit, we are asked to predict critical temperature of superconductors given its properties. On a brief note, we are required to perform two tasks which will be carried out in this notebook,

1] Predict the critical temperature Tc given some chemical properties of a material

2] Explain your prediction and the associated findings? For example, describe the key properties associated with the response variable.

To begin, we are provided a dataset from the superconducting material database maintained by Japan's National Institute for Material Science. So we begin by reading the data set into R and importing all required libraries for model building and processing the data.

In [1]:

```
library('ggplot2')
library('dplyr')
library('GGally')
library('corrplot')
library('caret')
library('car')
library('mctest')
```

```
library('randomForest')
library('xgboost')
library('leaps')
library('MASS')
library('tidyverse')
library('glmnet')
library('ggcorrplot')
options(warn=-1)
```

```
Registered S3 methods overwritten by 'ggplot2':
  method         from
  [.quosures     rlang
  c.quosures     rlang
  print.quosures rlang

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2

Attaching package: 'GGally'

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

    nasa

corrplot 0.84 loaded
Loading required package: lattice
Loading required package: carData

Attaching package: 'car'

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

    recode

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


Attaching package: 'xgboost'

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

    slice


Attaching package: 'MASS'

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

    select

Registered S3 method overwritten by 'rvest':
  method            from
  read_xml.response xml2
-- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
v tibble  2.1.1     v purrr   0.3.2
v tidyr   0.8.3     v stringr 1.4.0
v readr   1.3.1     v forcats 0.4.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x randomForest::combine() masks dplyr::combine()
x dplyr::filter()         masks stats::filter()
x dplyr::lag()            masks stats::lag()
x purrr::lift()           masks caret::lift()
x randomForest::margin()  masks ggplot2::margin()
x car::recode()           masks dplyr::recode()
x MASS::select()          masks dplyr::select()
x xgboost::slice()        masks dplyr::slice()
x purrr::some()           masks car::some()
Loading required package: Matrix

Attaching package: 'Matrix'

The following object is masked from 'package:tidyr':

    expand

Loading required package: foreach

Attaching package: 'foreach'

The following objects are masked from 'package:purrr':

    accumulate, when

Loaded glmnet 2.0-16
```

We now read the dataset using the read.csv command in R

In [2]:

```
data <- read.csv('train.csv')
```

In [3]:

```
head(data)
```

number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atomic_mass
4	88.94447	57.86269	66.36159	36.11661	1.18179
5	92.72921	58.51842	73.13279	36.39660	1.44930
4	88.94447	57.88524	66.36159	36.12251	1.18179
4	88.94447	57.87397	66.36159	36.11956	1.18179
4	88.94447	57.84014	66.36159	36.11072	1.18179
4	88.94447	57.79504	66.36159	36.09893	1.18179

In [4]:

```
glimpse(data)
```

```
Observations: 21,263
Variables: 82
$ number_of_elements              <int> 4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 5,...
$ mean_atomic_mass                <dbl> 88.94447, 92.72921, 88.94447, 88.94...
$ wtd_mean_atomic_mass            <dbl> 57.86269, 58.51842, 57.88524, 57.87...
$ gmean_atomic_mass               <dbl> 66.36159, 73.13279, 66.36159, 66.36...
$ wtd_gmean_atomic_mass           <dbl> 36.11661, 36.39660, 36.12251, 36.11...
$ entropy_atomic_mass             <dbl> 1.181795, 1.449309, 1.181795, 1.181...
$ wtd_entropy_atomic_mass         <dbl> 1.0623955, 1.0577551, 0.9759805, 1....
$ range_atomic_mass               <dbl> 122.9061, 122.9061, 122.9061, 122.9...
$ wtd_range_atomic_mass           <dbl> 31.79492, 36.16194, 35.74110, 33.76...
$ std_atomic_mass                 <dbl> 51.96883, 47.09463, 51.96883, 51.96...
$ wtd_std_atomic_mass             <dbl> 53.62253, 53.97987, 53.65627, 53.63...
$ mean_fie                        <dbl> 775.425, 766.440, 775.425, 775.425,...
```

```
$ wtd_mean_fie                    <dbl> 1010.269, 1010.613, 1010.820, 1010....
$ gmean_fie                       <dbl> 718.1529, 720.6055, 718.1529, 718.1...
$ wtd_gmean_fie                   <dbl> 938.0168, 938.7454, 939.0090, 938.5...
$ entropy_fie                     <dbl> 1.305967, 1.544145, 1.305967, 1.305...
$ wtd_entropy_fie                 <dbl> 0.7914878, 0.8070782, 0.7736202, 0....
$ range_fie                       <dbl> 810.6, 810.6, 810.6, 810.6, 810.6, ...
$ wtd_range_fie                   <dbl> 735.9857, 743.1643, 743.1643, 739.5...
$ std_fie                         <dbl> 323.8118, 290.1830, 323.8118, 323.8...
$ wtd_std_fie                     <dbl> 355.5630, 354.9635, 354.8042, 355.1...
$ mean_atomic_radius              <dbl> 160.25, 161.20, 160.25, 160.25, 160...
$ wtd_mean_atomic_radius          <dbl> 105.5143, 104.9714, 104.6857, 105.1...
$ gmean_atomic_radius             <dbl> 136.1260, 141.4652, 136.1260, 136.1...
$ wtd_gmean_atomic_radius         <dbl> 84.52842, 84.37017, 84.21457, 84.37...
$ entropy_atomic_radius           <dbl> 1.259244, 1.508328, 1.259244, 1.259...
$ wtd_entropy_atomic_radius       <dbl> 1.207040, 1.204115, 1.132547, 1.173...
$ range_atomic_radius             <int> 205, 205, 205, 205, 205, 205, 205, ...
$ wtd_range_atomic_radius         <dbl> 42.91429, 50.57143, 49.31429, 46.11...
$ std_atomic_radius               <dbl> 75.23754, 67.32132, 75.23754, 75.23...
$ wtd_std_atomic_radius           <dbl> 69.23557, 68.00882, 67.79771, 68.52...
$ mean_Density                    <dbl> 4654.357, 5821.486, 4654.357, 4654....
$ wtd_mean_Density                <dbl> 2961.502, 3021.017, 2999.159, 2980....
$ gmean_Density                   <dbl> 724.9532, 1237.0951, 724.9532, 724....
$ wtd_gmean_Density               <dbl> 53.54381, 54.09572, 53.97402, 53.75...
$ entropy_Density                 <dbl> 1.0331288, 1.3144422, 1.0331288, 1....
$ wtd_entropy_Density             <dbl> 0.8145982, 0.9148022, 0.7603052, 0....
$ range_Density                   <dbl> 8958.571, 10488.571, 8958.571, 8958...
$ wtd_range_Density               <dbl> 1579.583, 1667.383, 1667.383, 1623....
$ std_Density                     <dbl> 3306.163, 3767.403, 3306.163, 3306....
$ wtd_std_Density                 <dbl> 3572.597, 3632.649, 3592.019, 3582....
$ mean_ElectronAffinity           <dbl> 81.8375, 90.8900, 81.8375, 81.8375,...
$ wtd_mean_ElectronAffinity       <dbl> 111.7271, 112.3164, 112.2136, 111.9...
$ gmean_ElectronAffinity          <dbl> 60.12318, 69.83331, 60.12318, 60.12...
$ wtd_gmean_ElectronAffinity      <dbl> 99.41468, 101.16640, 101.08215, 100...
$ entropy_ElectronAffinity        <dbl> 1.159687, 1.427997, 1.159687, 1.159...
$ wtd_entropy_ElectronAffinity    <dbl> 0.7873817, 0.8386665, 0.7860067, 0....
$ range_ElectronAffinity          <dbl> 127.05, 127.05, 127.05, 127.05, 127...
$ wtd_range_ElectronAffinity      <dbl> 80.98714, 81.20786, 81.20786, 81.09...
$ std_ElectronAffinity            <dbl> 51.43371, 49.43817, 51.43371, 51.43...
$ wtd_std_ElectronAffinity        <dbl> 42.55840, 41.66762, 41.63988, 42.10...
$ mean_FusionHeat                 <dbl> 6.9055, 7.7844, 6.9055, 6.9055, 6.9...
$ wtd_mean_FusionHeat             <dbl> 3.846857, 3.796857, 3.822571, 3.834...
$ gmean_FusionHeat                <dbl> 3.479475, 4.403790, 3.479475, 3.479...
$ wtd_gmean_FusionHeat            <dbl> 1.0409860, 1.0352511, 1.0374394, 1....
$ entropy_FusionHeat              <dbl> 1.088575, 1.374977, 1.088575, 1.088...
$ wtd_entropy_FusionHeat          <dbl> 0.9949982, 1.0730938, 0.9274794, 0....
$ range_FusionHeat                <dbl> 12.878, 12.878, 12.878, 12.878, 12....
$ wtd_range_FusionHeat            <dbl> 1.744571, 1.595714, 1.757143, 1.744...
$ std_FusionHeat                  <dbl> 4.599064, 4.473363, 4.599064, 4.599...
$ wtd_std_FusionHeat              <dbl> 4.666920, 4.603000, 4.649635, 4.658...
$ mean_ThermalConductivity        <dbl> 107.75665, 172.20532, 107.75665, 10...
$ wtd_mean_ThermalConductivity    <dbl> 61.01519, 61.37233, 60.94376, 60.97...
$ gmean_ThermalConductivity       <dbl> 7.062488, 16.064228, 7.062488, 7.06...
$ wtd_gmean_ThermalConductivity   <dbl> 0.6219795, 0.6197346, 0.6190947, 0....
$ entropy_ThermalConductivity     <dbl> 0.3081480, 0.8474042, 0.3081480, 0....
$ wtd_entropy_ThermalConductivity <dbl> 0.2628483, 0.5677061, 0.2504774, 0....
$ range_ThermalConductivity       <dbl> 399.9734, 429.9734, 399.9734, 399.9...
$ wtd_range_ThermalConductivity   <dbl> 57.12767, 51.41338, 57.12767, 57.12...
$ std_ThermalConductivity         <dbl> 168.8542, 198.5546, 168.8542, 168.8...
$ wtd_std_ThermalConductivity     <dbl> 138.5172, 139.6309, 138.5406, 138.5...
$ mean_Valence                    <dbl> 2.25, 2.00, 2.25, 2.25, 2.25, 2.25,...
$ wtd_mean_Valence                <dbl> 2.257143, 2.257143, 2.271429, 2.264...
$ gmean_Valence                   <dbl> 2.213364, 1.888175, 2.213364, 2.213...
$ wtd_gmean_Valence               <dbl> 2.219783, 2.210679, 2.232679, 2.226...
$ entropy_Valence                 <dbl> 1.368922, 1.557113, 1.368922, 1.368...
$ wtd_entropy_Valence             <dbl> 1.066221, 1.047221, 1.029175, 1.048...
$ range_Valence                   <int> 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ wtd_range_Valence               <dbl> 1.0857143, 1.1285714, 1.1142857, 1....
$ std_Valence                     <dbl> 0.4330127, 0.6324555, 0.4330127, 0....
$ wtd_std_Valence                 <dbl> 0.4370588, 0.4686063, 0.4446966, 0....
$ critical_temp                   <dbl> 29.0, 26.0, 19.0, 22.0, 23.0, 23.0,...
```

The dataset consists of 21,263 observations with 83 columns/features.

As specified in the description of the assignment, the data consists mainly of the following

properties of elements,

Number of elements :- Indicates number of elements for each row.
Atomic Mass :- The mass of an atom of a chemical element expressed in atomic mass units. It is approximately equivalent to the number of protons and neutrons in the atom.
First Ionization Energy :- The minimum amount of energy required to remove the most loosely bound electron, the valence electron, of an isolated neutral gaseous atom or molecule.
Atomic Radius :- The atomic radius of a chemical element is a measure of the size of its atoms, usually the mean or typical distance from the center of the nucleus to the boundary of the surrounding shells of electrons.
Density :- The density, or more precisely, the volumetric mass density, of a substance is its mass per unit volume.
Electron Affinity :- The electron affinity of an atom or molecule is defined as the amount of energy released or spent when an electron is added to a neutral atom or molecule in the gaseous state to form a negative ion.
Fusion Heat :- The change in an atom's enthalpy resulting from providing energy, typically heat, to a specific quantity of the substance to change its state from a solid to a liquid, at constant pressure.
Thermal Conductivity :- A measure of an atom's ability to conduct heat. It is commonly denoted by, or. Heat transfer occurs at a lower rate in materials of low thermal conductivity than in materials of high thermal conductivity.
Valence :- A valence electron is an outer shell electron that is associated with an atom, and that can participate in the formation of a chemical bond if the outer shell is not closed; in a single covalent bond, both atoms in the bond contribute one valence electron in order to form a shared pair.

For each of the above properties, we are provided with ten features for the element properties,

Mean :- The central value of a discrete set of numbers.
Weighted mean :- The geographic center of a set of points as adjusted for the influence of a value associated with each point.
Geometric mean :- The geometric mean is a mean or average, which indicates the central tendency or typical value of a set of numbers by using the product of their values.
Weighted geometric mean :- If we introduce additional weights in calculation to the geometric mean, we obtain weighted geometric mean.
Entropy :- A thermodynamic quantity representing the unavailability of a system's thermal energy for conversion into mechanical work, often interpreted as the degree of disorder or randomness in the system.
Weighted entropy :- Weighted entropy is the measure of information supplied by a probablistic experiment whose elementary events are characterized both by their objective probabilities and by some qualitative (objective or subjective) weights. Range :- Set of difference between the largest and smallest value.
Weighted range :- Similar to calculating range but with addition of some weights.
Standard deviation :- A quantity expressing by how much the members of a group differ from the mean value for the group.
Weighted standard deviation :- A weighted standard deviation allows you to apply a weight, or relative significance to each value in a set of values. Values with a higher value for their weight are considered as more significant to a sample as compared to the other values in a sample.

To begin with, we will first randomly spit the data to training and testing with a 80:20 split.

In [5]:

```
set.seed(123)
# Random sample indexes
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
test_index <- setdiff(1:nrow(data), train_index)

# Build X_train, y_train, X_test, y_test
train_data <- data[train_index, ]
test_data <- data[test_index,]
```

As shown from the output indices below, we can see that the data is split randomly to train and test and their dimensions add up to the dimension of the dataset provided. Thus, we have succesfully created a train/test split. Moving further, we will now work with the training data to perform EDA, feature selection and model buiding and test it on the testing data.

In [6]:

```
head(train_data)
```

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atom
18847	6	60.60051	71.50999	46.72851	55.49176	1
18895	4	57.44445	60.35930	56.06791	58.81637	1
2986	6	84.71115	78.84015	66.61372	64.69108	1

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atom
1842	7	112.95469	60.86673	82.42970	36.81151	1
3371	6	78.67813	59.21927	58.87964	36.12306	1
11638	3	49.59452	37.84410	37.11177	24.83395	0

In [7]:

```
head(test_data)
```

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atomic_
3	4	88.94447	57.88524	66.36159	36.12251	1.18
8	4	76.51772	57.17514	59.31010	35.89137	1.19
12	5	111.27357	63.71346	82.79332	37.93423	1.40
24	4	76.51772	55.70984	59.31010	35.42195	1.19
34	4	76.44456	65.83465	59.35667	48.95599	1.19
35	4	96.03285	77.27947	69.51593	53.61112	1.15

In [8]:

```
dim(train_data)
```

17010 82

In [9]:

```
dim(test_data)
```

4253 82

In [10]:

```
dim(data)
```

21263 82

2. Data Exploration

In this section, we will perform some exploratory analysis on the dataset, followed by using some filter methods to get rid of unnecessary features. In brief, we will be making use of 3 filter methods to get rid of unneccessary features,

- Removing features with more than 50% NA values.
- Removing features with low variance(10%).
- Removing highly correlated features.

We begin with searching for any NA/null values in the dataset.

In [11]:

```
sum(is.na(train_data))
```

0

As we can see, we don't have any missing data in the dataset. We can now proceed to the next section where we check for variables with low variance.

We now check for features which have low variance. This can be done by using the nearZeroVar function from the caret

package and filter our variables to remove those which have less than 10% variance.

In [12]:

```
x <- nearZeroVar(train_data, saveMetrics = TRUE)
x[x$percentUnique<10,]
x_new <- x[x$percentUnique>10,] # Filters out variables with variance below 10%.
```

	freqRatio	percentUnique	zeroVar	nzv
number_of_elements	1.292273	0.05291005	FALSE	FALSE
range_atomic_mass	1.023551	5.82010582	FALSE	FALSE
range_fie	3.018448	4.78542034	FALSE	FALSE
mean_atomic_radius	1.617886	5.39682540	FALSE	FALSE
range_atomic_radius	1.958866	1.13462669	FALSE	FALSE
range_Density	4.253769	4.95590829	FALSE	FALSE
range_ElectronAffinity	1.122555	3.64491476	FALSE	FALSE
range_FusionHeat	13.266385	3.34509112	FALSE	FALSE
range_ThermalConductivity	23.618234	2.56907701	FALSE	TRUE
mean_Valence	1.025624	0.36449148	FALSE	FALSE
gmean_Valence	1.184107	1.36390359	FALSE	FALSE
entropy_Valence	1.194136	1.80482069	FALSE	FALSE
range_Valence	1.446777	0.04115226	FALSE	FALSE
std_Valence	1.174943	0.71134627	FALSE	FALSE

In [13]:

```
features<-rownames(x_new) # we assign the names of the features to a variable so as to get rid of
the features with low variance
```

In [14]:

```
data_train_sub<-train_data[, (colnames(train_data) %in% features)] # subset the data
dim(data_train_sub)
```

17010 68

We now proceed by removing highly correlated variables from the dataset. Removal of highly correlated variables is pivotal as correlation implies that there is little information contained in any linear combination of the concerned features. Removing the correlation by reducing the number of features can be seen to smooth out noise and simplify the model.

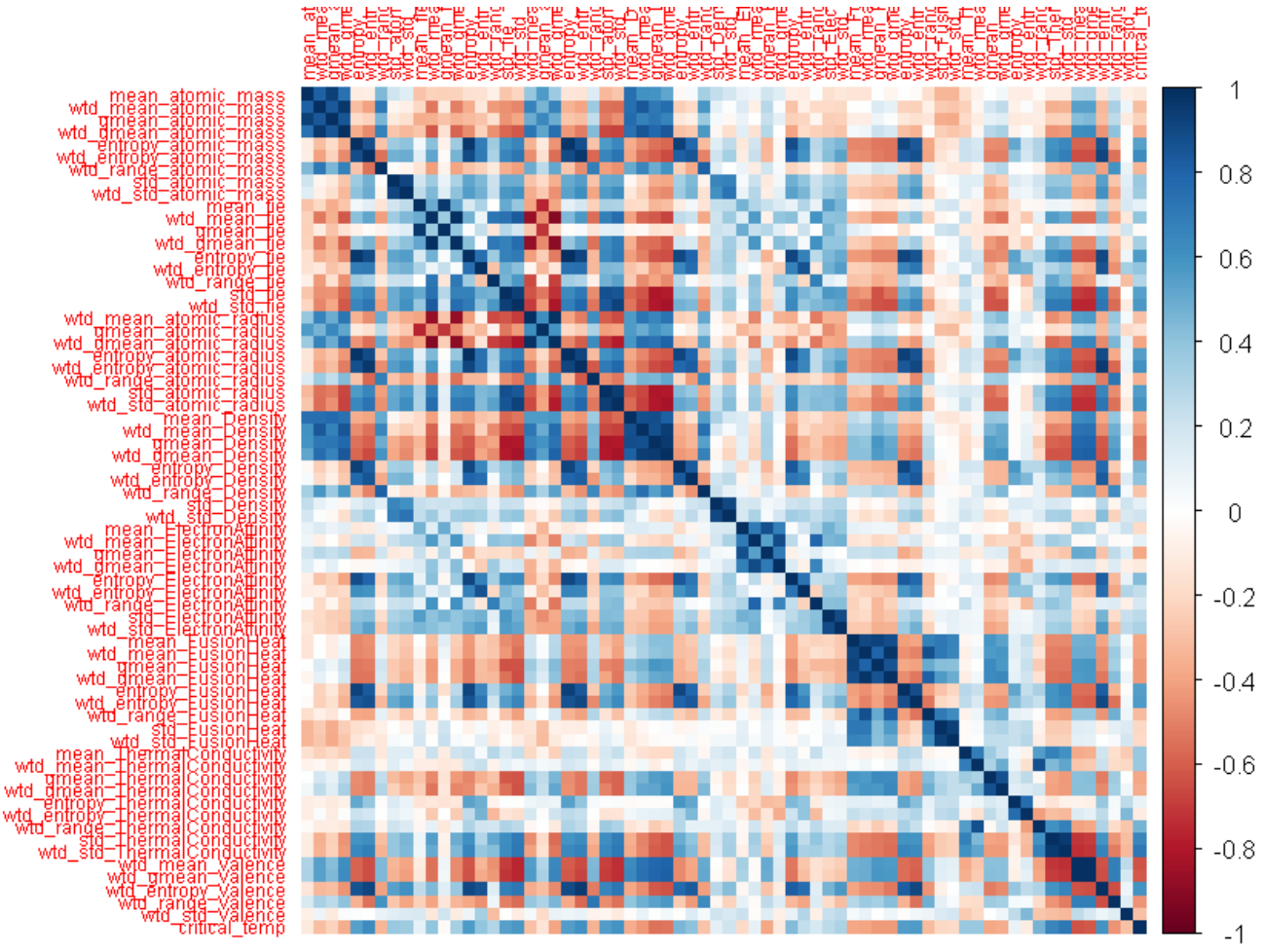
We begin by creating a correlation matrix of the data(after removing low variance variables).

In [15]:

```
correlation_matrix <- cor(data_train_sub)
```

In [16]:

```
corrplot(correlation_matrix, method="color",tl.cex = 0.6)
```



As seen from above, there seem to be a lot of highly correlated features which will lead to more noise and hence hinder the model performance. However, the above plot looks really clumsy due to a large number of features present in the dataset. Thus to make it more simpler, we will create multiple correlation plots with the properties of the elements.

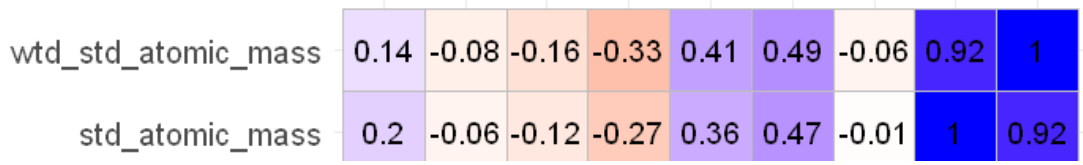
In [17]:

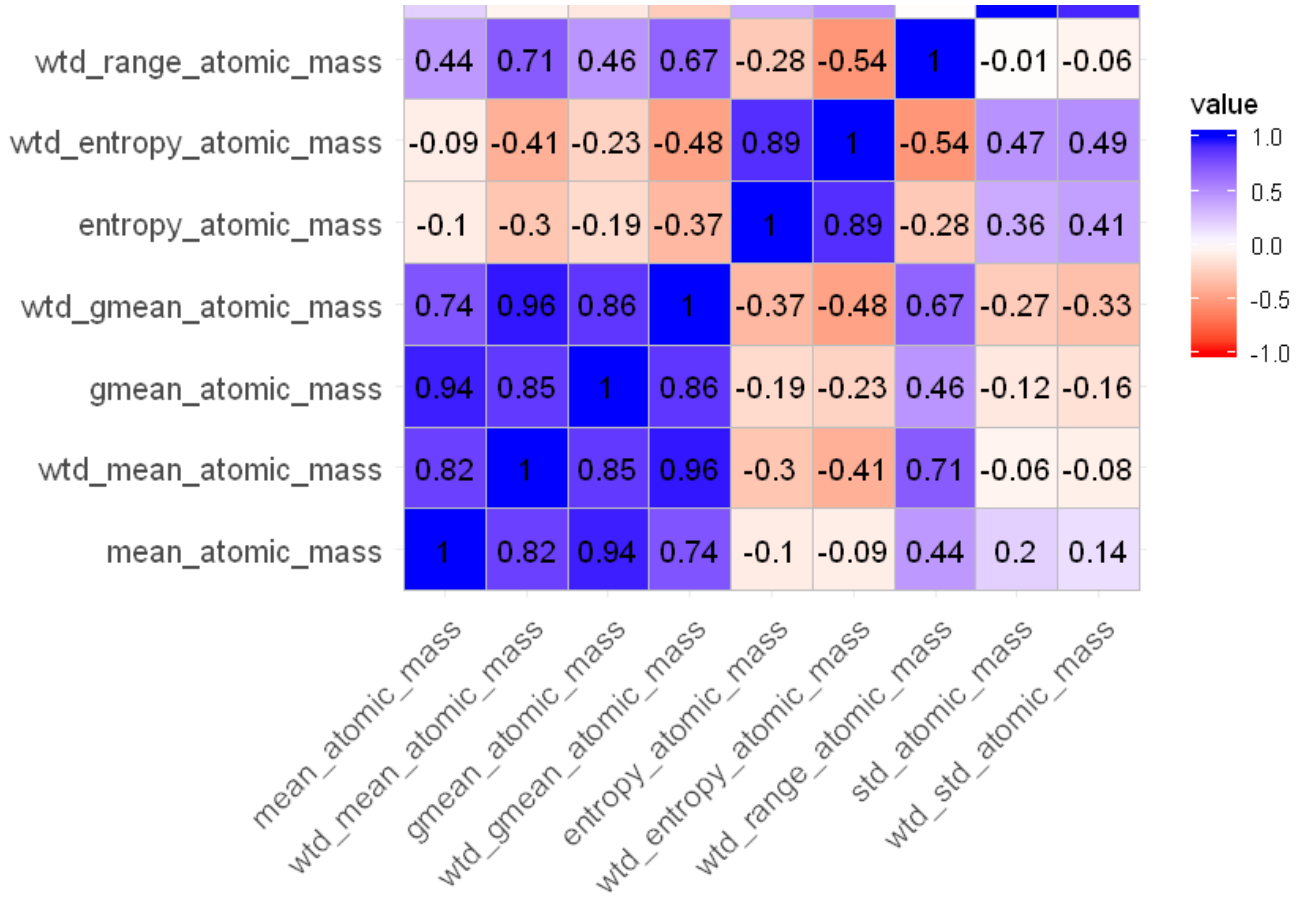
```
atomic_mass <- data_train_sub %>% dplyr::select(contains("atomic_mass"))
fie <- data_train_sub %>% dplyr::select(contains("fie"))
atomic_radius <- data_train_sub %>% dplyr::select(contains("atomic_radius"))
density <- data_train_sub %>% dplyr::select(contains("Density"))
electron_affinity <- data_train_sub %>% dplyr::select(contains("ElectronAffinity"))
fusion_heat <- data_train_sub %>% dplyr::select(contains("FusionHeat"))
thermal_conductivity <- data_train_sub %>% dplyr::select(contains("ThermalConductivity"))
valence <- data_train_sub %>% dplyr::select(contains("Valence"))
```

In [18]:

```
ggcorrplot(cor(atomic_mass),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low =
"red", high =  "blue", mid = "white", midpoint = 0)
```

```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```

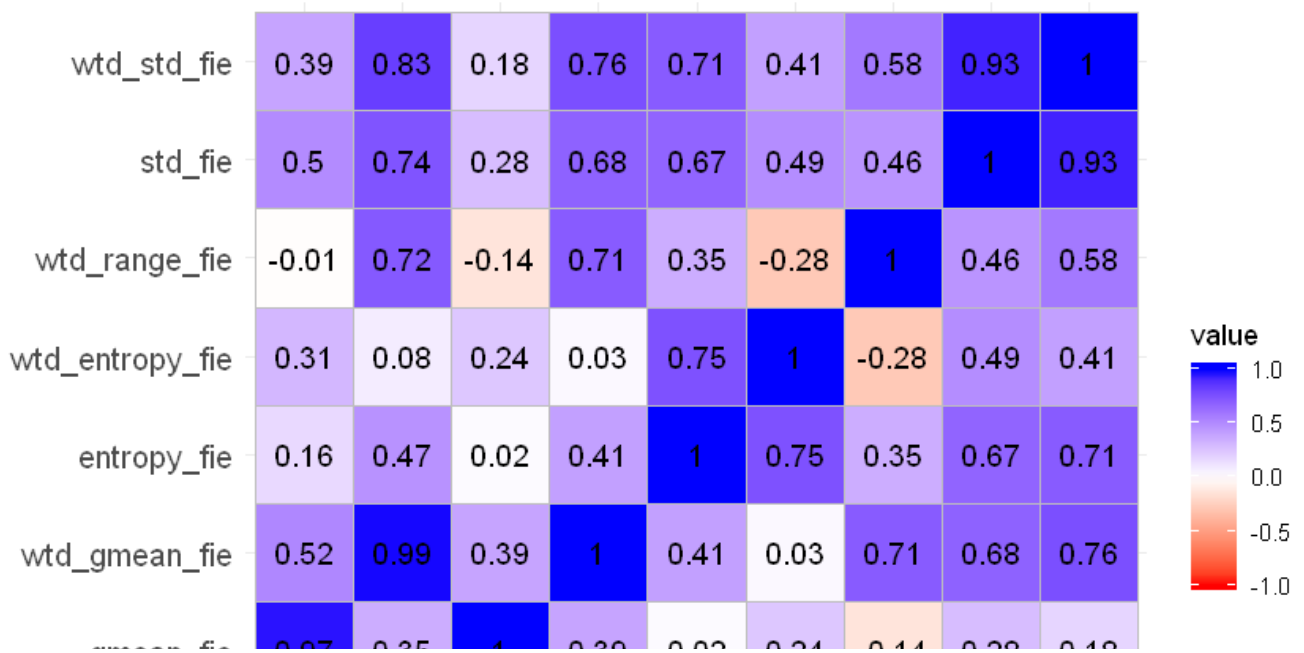


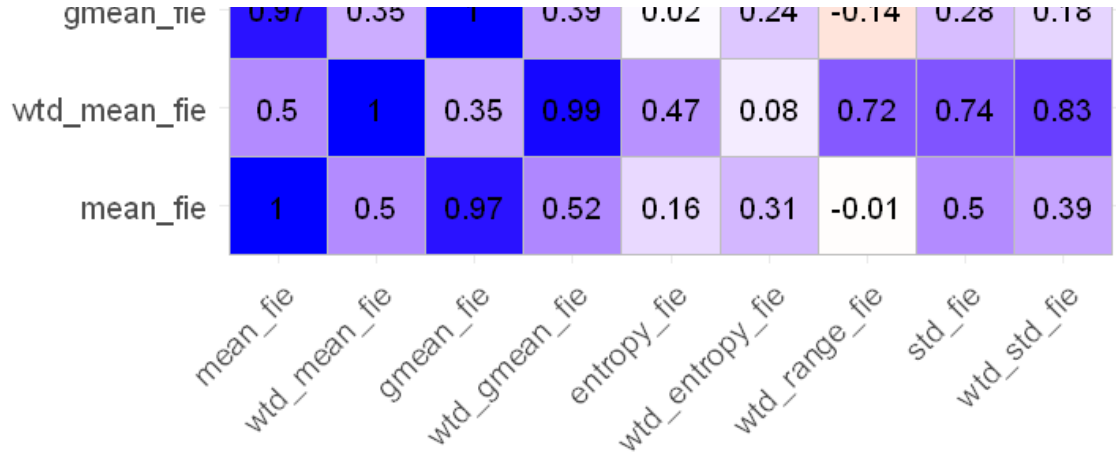


In [19]:

```
ggcorrplot(cor(fie),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low = "red",
high =  "blue", mid = "white", midpoint = 0)
```

```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```

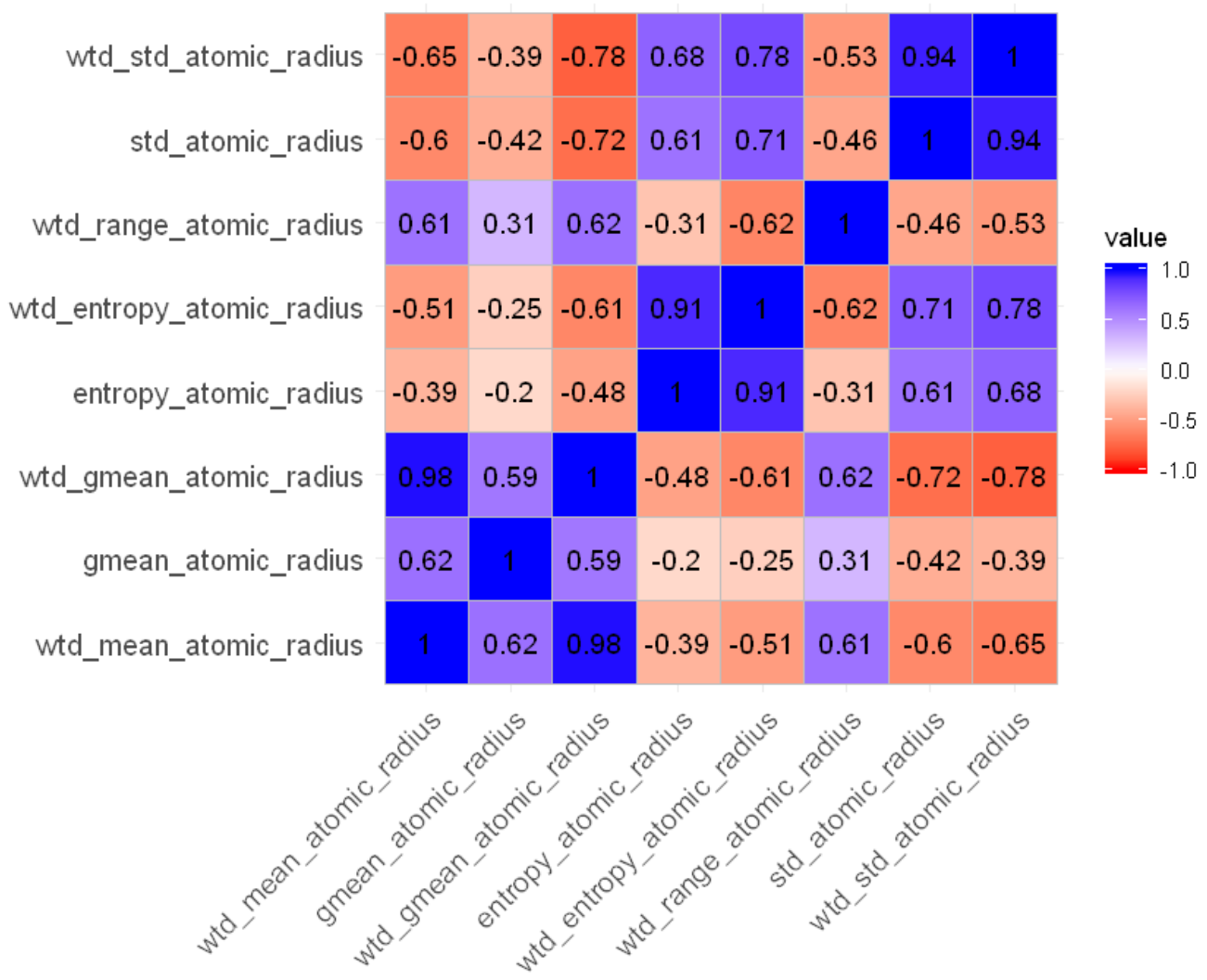




In [20]:

```
ggcorrplot(cor(atomic_radius),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low
= "red", high =  "blue", mid = "white", midpoint = 0)
```

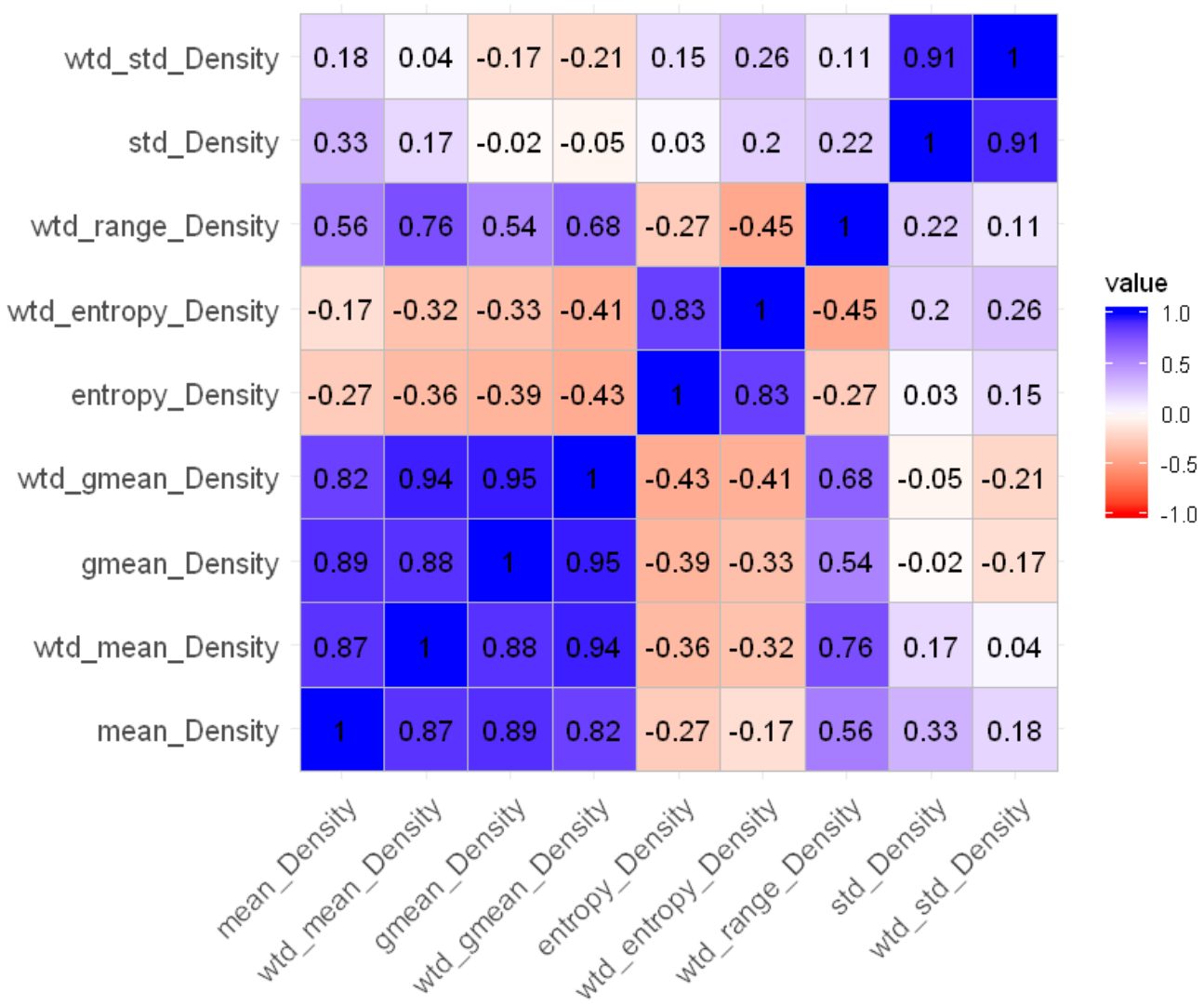
```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```



In [21]:

```
ggcorrplot(cor(density),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low = "red", high =  "blue", mid = "white", midpoint = 0)
```

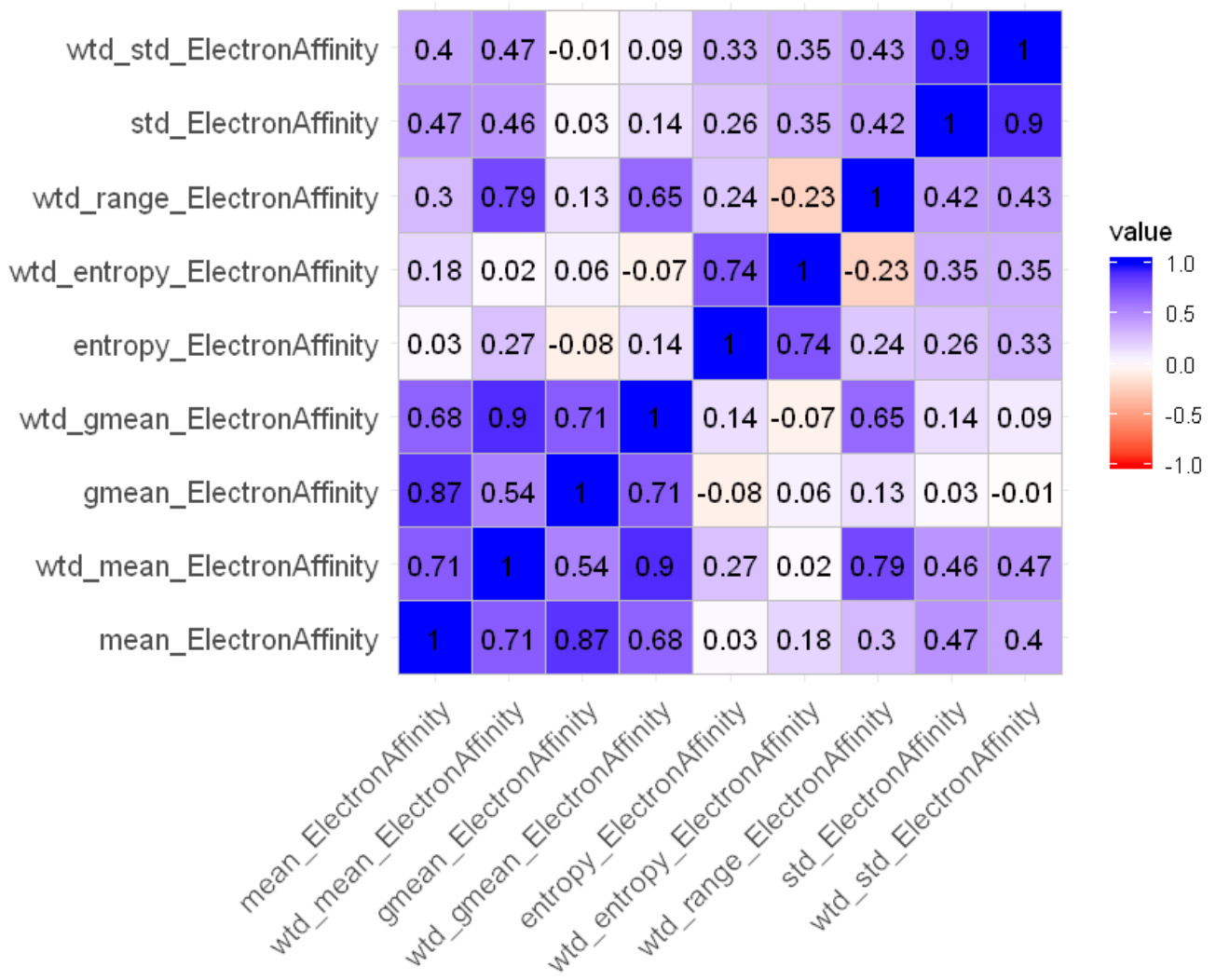
```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```



In [22]:

```
ggcorrplot(cor(electron_affinity),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low = "red", high =  "blue", mid = "white", midpoint = 0)
```

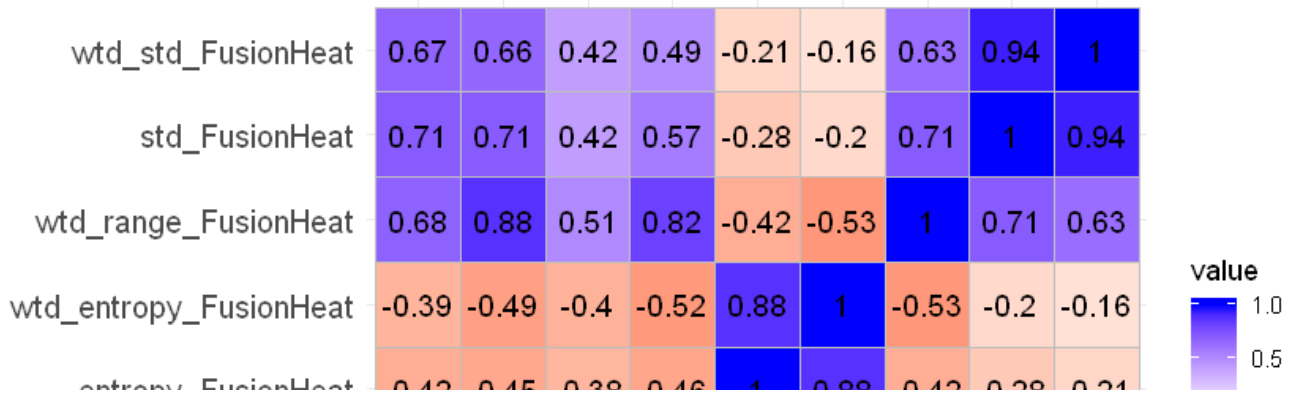
```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```

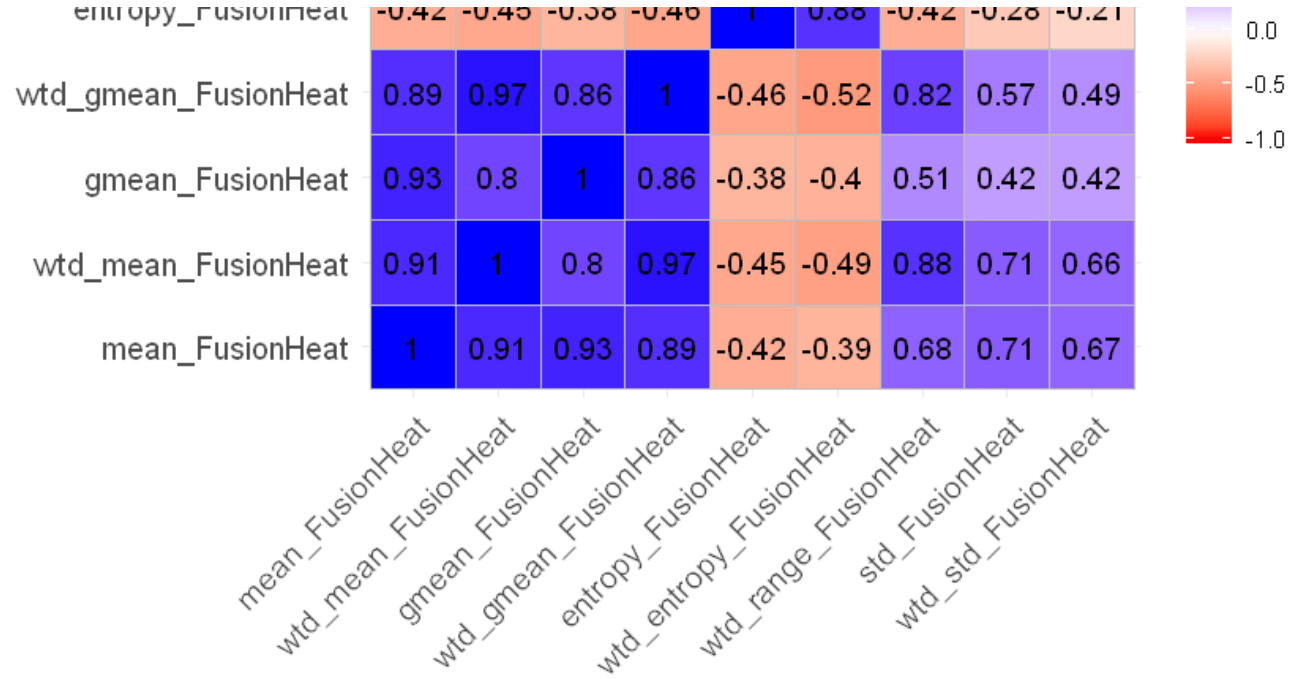


In [23]:

```
ggcorrplot(cor(fusion_heat),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low =
"red", high =  "blue", mid = "white", midpoint = 0)
```

```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```

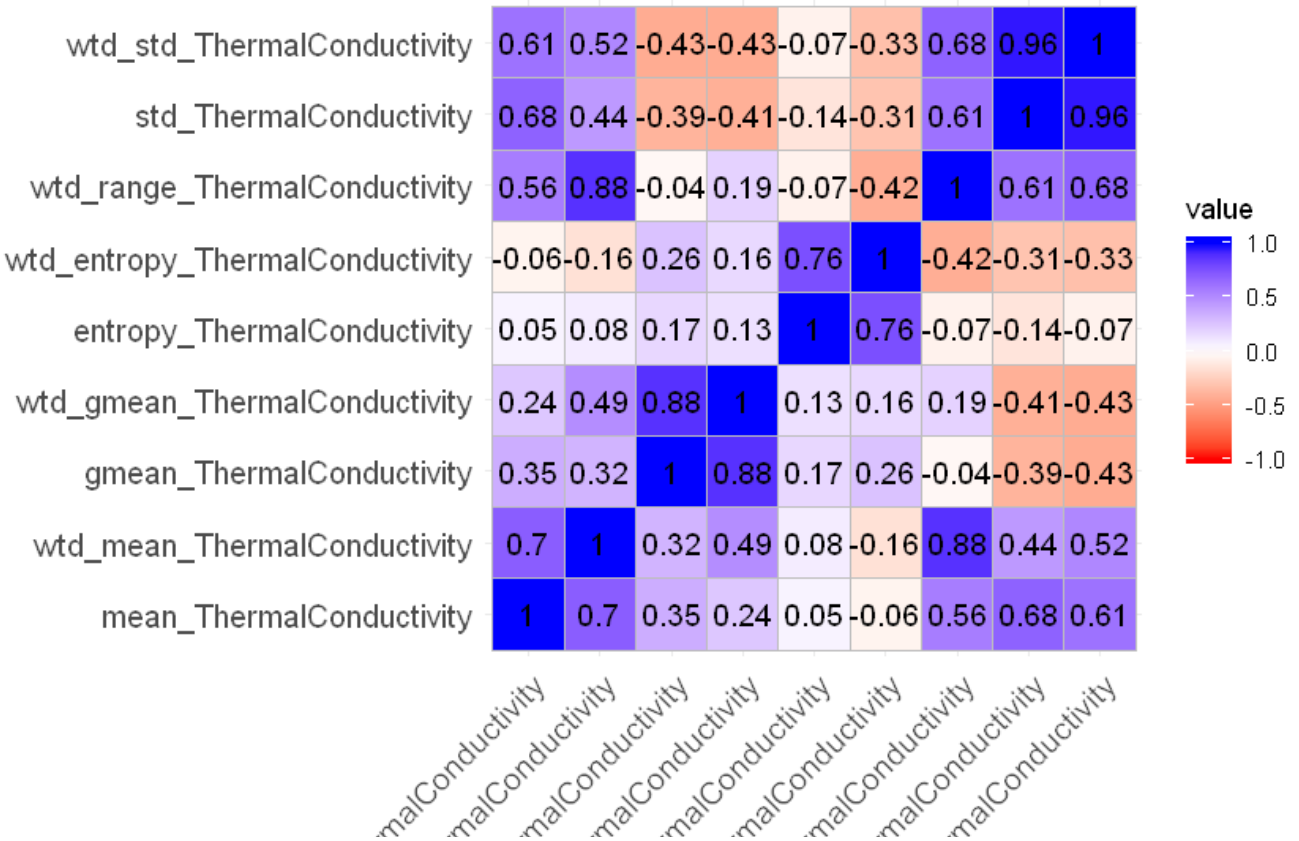


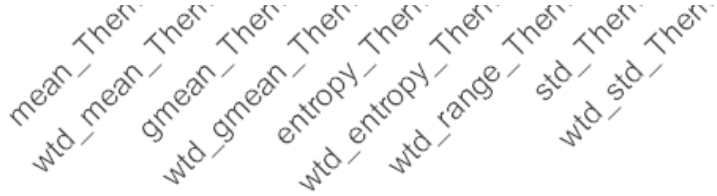


In [24]:

```
ggcorrplot(cor(thermal_conductivity),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,
1), low = "red", high =  "blue", mid = "white", midpoint = 0)
```

```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```

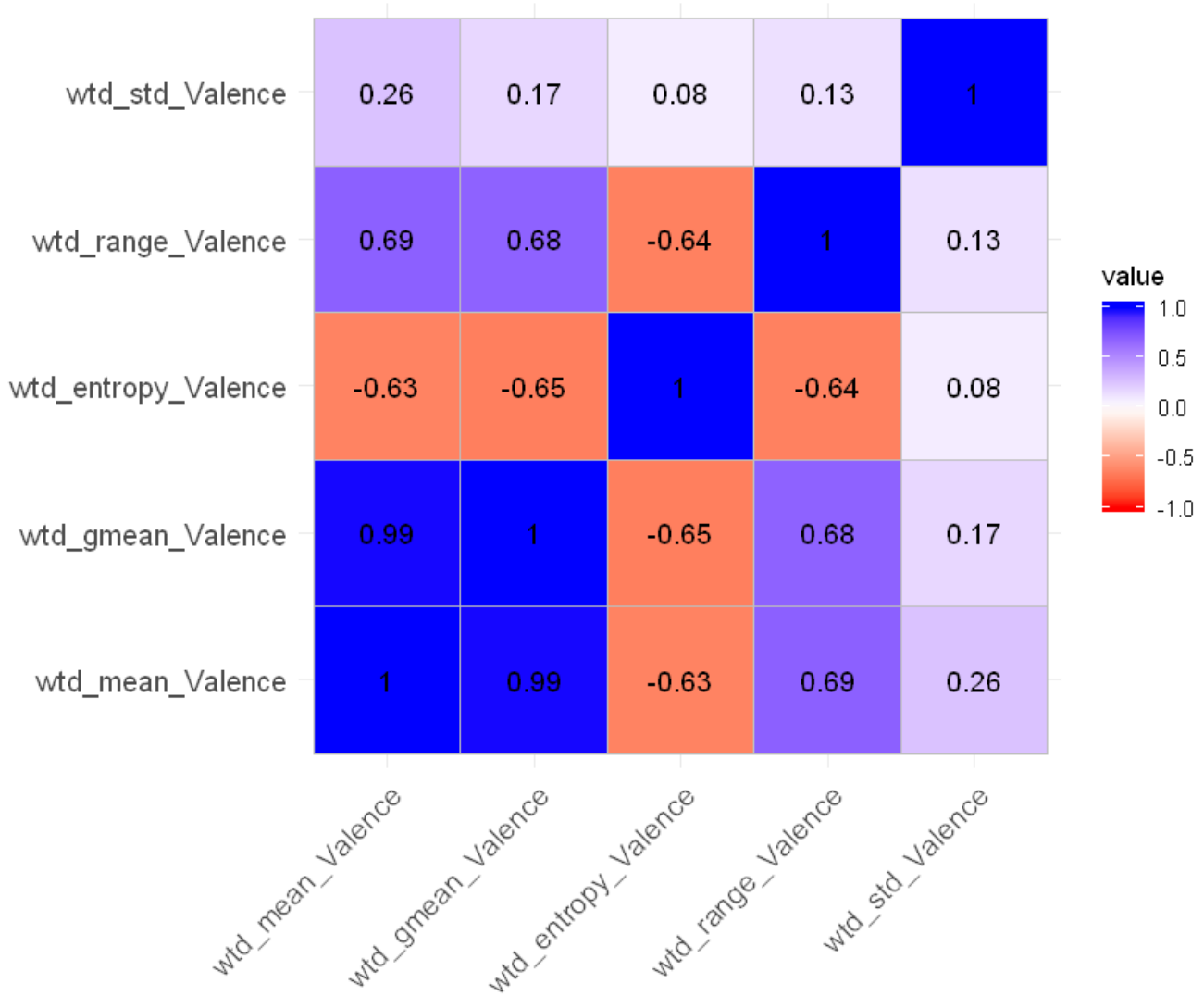




In [25]:

```
ggcorrplot(cor(valence),lab = TRUE,colors="blue") + scale_fill_gradient2(limit = c(-1,1), low = "red", high =  "blue", mid = "white", midpoint = 0)
```

```
Scale for 'fill' is already present. Adding another scale for 'fill', which
will replace the existing scale.
```



One major trend in the above correlation plots is the high correlation between weighted standard deviation and standard deviation of the properties. There is also a high correlation between different measurements of mean(mean, weighted mean, geometric mean and

weighted geometric mean) in atomic mass, fusion heat, electron affinity and density.

To remove highly correlated features, we will however use the first correlation matrix as it contains the interaction of the feature with all the other features. Thus, to get rid of these features, we use a function from the caret package, which is findCorrelation. The function searches for absolute values of pair-wise correlations. If variables have a high correlation, the function looks at the mean absolute correlation of each variable and removes the variable with the largest mean absolute correlation. We have a set a cutoff at 0.8, which means any set of variables that have a correlation of 0.8 or more, we will retain only one such feature and get rid of the rest.

In [26]:

```
remove_features <- findCorrelation(correlation_matrix,cutoff = 0.8,names=TRUE)
remove_features
```

'wtd_std_fie' 'wtd_entropy_atomic_radius' 'wtd_std_atomic_radius' 'wtd_gmean_Density' 'wtd_entropy_atomic_mass' 'wtd_entropy_Valence' 'std_fie' 'entropy_fie' 'entropy_atomic_radius' 'gmean_Density' 'wtd_gmean_Valence' 'wtd_std_ThermalConductivity' 'entropy_atomic_mass' 'wtd_gmean_atomic_radius' 'wtd_entropy_FusionHeat' 'wtd_mean_fie' 'entropy_ElectronAffinity' 'wtd_mean_Density' 'entropy_FusionHeat' 'wtd_mean_atomic_radius' 'wtd_gmean_atomic_mass' 'entropy_Density' 'wtd_gmean_FusionHeat' 'gmean_FusionHeat' 'wtd_gmean_ThermalConductivity' 'wtd_entropy_fie' 'wtd_std_atomic_mass' 'wtd_std_ElectronAffinity' 'wtd_mean_atomic_mass' 'wtd_mean_FusionHeat' 'wtd_range_Density' 'gmean_atomic_mass' 'wtd_mean_ElectronAffinity' 'std_Density' 'gmean_fie' 'wtd_std_FusionHeat' 'mean_ElectronAffinity' 'wtd_mean_ThermalConductivity'

The above features are the highly correlated ones and below the code will subset the data further to exclude these features

In [27]:

```
data_train_split <- data_train_sub[, !(colnames(data_train_sub) %in% remove_features)]
dim(data_train_split)
test_data_split <- test_data[, (colnames(test_data) %in% colnames(data_train_split))]
dim(test_data_split)
```

17010 30

4253 30

3. Variable Identification and Explanation

We now perform model based feature selection, which is wrapper and embedded methods.

In our first approach, we build a stepwise linear regression model on the data. Step wise regression consists of iterative adding and removing predictors in order to find the subset of the predictors which results in best performance model. We will perform a backward regression i.e. beginning with a full model, we iteratively remove variables from the model tillwe optimize our model performance.

I set the threshold at 10 as a decrease in number of features causes an increase in BIC and RSS and a decrease in the adjusted r squared. Increasing the number of features didn't have any significant impact either as the improvement was fairly constant. Thus, we set nvmax to 10.

In [28]:

```
regfit.full <- regsubsets(critical_temp ~ .,data=data_train_split,nvmax=10,method = 'backward')
reg.summary <- summary(regfit.full)
reg.summary
```

```
Subset selection object
Call: regsubsets.formula(critical_temp ~ ., data = data_train_split,
    nvmax = 10, method = "backward")
29 Variables  (and intercept)
                              Forced in Forced out
mean_atomic_mass                  FALSE      FALSE
wtd_range_atomic_mass             FALSE      FALSE
std_atomic_mass                   FALSE      FALSE
```

```
                                  _       _
mean_fie                          FALSE      FALSE
wtd_gmean_fie                     FALSE      FALSE
wtd_range_fie                     FALSE      FALSE
gmean_atomic_radius               FALSE      FALSE
wtd_range_atomic_radius           FALSE      FALSE
std_atomic_radius                 FALSE      FALSE
mean_Density                      FALSE      FALSE
wtd_entropy_Density               FALSE      FALSE
wtd_std_Density                   FALSE      FALSE
gmean_ElectronAffinity            FALSE      FALSE
wtd_gmean_ElectronAffinity        FALSE      FALSE
wtd_entropy_ElectronAffinity      FALSE      FALSE
wtd_range_ElectronAffinity        FALSE      FALSE
std_ElectronAffinity              FALSE      FALSE
mean_FusionHeat                   FALSE      FALSE
wtd_range_FusionHeat              FALSE      FALSE
std_FusionHeat                    FALSE      FALSE
mean_ThermalConductivity          FALSE      FALSE
gmean_ThermalConductivity         FALSE      FALSE
entropy_ThermalConductivity       FALSE      FALSE
wtd_entropy_ThermalConductivity   FALSE      FALSE
wtd_range_ThermalConductivity     FALSE      FALSE
std_ThermalConductivity           FALSE      FALSE
wtd_mean_Valence                  FALSE      FALSE
wtd_range_Valence                 FALSE      FALSE
wtd_std_Valence                   FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: backward
          mean_atomic_mass wtd_range_atomic_mass std_atomic_mass mean_fie
1  ( 1 )  " "              " "                   " "             " "
2  ( 1 )  " "              " "                   " "             " "
3  ( 1 )  " "              " "                   " "             " "
4  ( 1 )  " "              " "                   " "             " "
5  ( 1 )  " "              " "                   " "             " "
6  ( 1 )  " "              " "                   " "             " "
7  ( 1 )  " "              " "                   " "             " "
8  ( 1 )  " "              " "                   " "             " "
9  ( 1 )  " "              " "                   " "             " "
10  ( 1 ) " "              " "                   " "             "*"
          wtd_gmean_fie wtd_range_fie gmean_atomic_radius
1  ( 1 )  " "           " "           " "
2  ( 1 )  " "           " "           " "
3  ( 1 )  " "           " "           " "
4  ( 1 )  " "           " "           " "
5  ( 1 )  " "           " "           " "
6  ( 1 )  " "           " "           " "
7  ( 1 )  " "           " "           " "
8  ( 1 )  " "           " "           " "
9  ( 1 )  " "           " "           " "
10  ( 1 ) " "           " "           " "
          wtd_range_atomic_radius std_atomic_radius mean_Density
1  ( 1 )  " "                     " "               " "
2  ( 1 )  " "                     " "               " "
3  ( 1 )  " "                     " "               " "
4  ( 1 )  " "                     "*"               " "
5  ( 1 )  " "                     "*"               " "
6  ( 1 )  " "                     "*"               " "
7  ( 1 )  " "                     "*"               " "
8  ( 1 )  "*"                     "*"               " "
9  ( 1 )  "*"                     "*"               " "
10  ( 1 ) "*"                     "*"               " "
          wtd_entropy_Density wtd_std_Density gmean_ElectronAffinity
1  ( 1 )  " "                 " "             " "
2  ( 1 )  " "                 " "             " "
3  ( 1 )  " "                 " "             " "
4  ( 1 )  " "                 " "             " "
5  ( 1 )  " "                 " "             " "
6  ( 1 )  " "                 " "             " "
7  ( 1 )  " "                 " "             " "
8  ( 1 )  " "                 " "             " "
9  ( 1 )  " "                 " "             " "
10  ( 1 ) " "                 " "             " "
          wtd_gmean_ElectronAffinity wtd_entropy_ElectronAffinity
1  ( 1 )  " "                        " "
2  ( 1 )  " "                        " "
3  ( 1 )  "*"                        " "
4  ( 1 )  "*"                        " "
```

```
5  ( 1 )  "*"                          " "
6  ( 1 )  "*"                          " "
7  ( 1 )  "*"                          " "
8  ( 1 )  "*"                          " "
9  ( 1 )  "*"                          " "
10  ( 1 ) "*"                          " "
          wtd_range_ElectronAffinity std_ElectronAffinity mean_FusionHeat
1  ( 1 )  " "                         " "                 " "
2  ( 1 )  " "                         " "                 " "
3  ( 1 )  " "                         " "                 " "
4  ( 1 )  " "                         " "                 " "
5  ( 1 )  " "                         " "                 " "
6  ( 1 )  " "                         " "                 " "
7  ( 1 )  " "                         " "                 " "
8  ( 1 )  " "                         " "                 " "
9  ( 1 )  " "                         " "                 " "
10  ( 1 ) " "                         " "                 " "
          wtd_range_FusionHeat std_FusionHeat mean_ThermalConductivity
1  ( 1 )  " "                  " "            " "
2  ( 1 )  " "                  " "            "*"
3  ( 1 )  " "                  " "            "*"
4  ( 1 )  " "                  " "            "*"
5  ( 1 )  " "                  " "            "*"
6  ( 1 )  " "                  " "            "*"
7  ( 1 )  " "                  " "            "*"
8  ( 1 )  " "                  " "            "*"
9  ( 1 )  " "                  "*"            "*"
10  ( 1 ) " "                  "*"            "*"
          gmean_ThermalConductivity entropy_ThermalConductivity
1  ( 1 )  "*"                        " "
2  ( 1 )  "*"                        " "
3  ( 1 )  "*"                        " "
4  ( 1 )  "*"                        " "
5  ( 1 )  "*"                        " "
6  ( 1 )  "*"                        "*"
7  ( 1 )  "*"                        "*"
8  ( 1 )  "*"                        "*"
9  ( 1 )  "*"                        "*"
10  ( 1 ) "*"                        "*"
          wtd_entropy_ThermalConductivity wtd_range_ThermalConductivity
1  ( 1 )  " "                             " "
2  ( 1 )  " "                             " "
3  ( 1 )  " "                             " "
4  ( 1 )  " "                             " "
5  ( 1 )  " "                             " "
6  ( 1 )  " "                             " "
7  ( 1 )  " "                             "*"
8  ( 1 )  " "                             "*"
9  ( 1 )  " "                             "*"
10  ( 1 ) " "                             "*"
          std_ThermalConductivity wtd_mean_Valence wtd_range_Valence
1  ( 1 )  " "                     " "               " "
2  ( 1 )  " "                     " "               " "
3  ( 1 )  " "                     " "               " "
4  ( 1 )  " "                     " "               " "
5  ( 1 )  " "                     " "               " "
6  ( 1 )  " "                     " "               " "
7  ( 1 )  " "                     " "               " "
8  ( 1 )  " "                     " "               " "
9  ( 1 )  " "                     " "               " "
10  ( 1 ) " "                     " "               " "
          wtd_std_Valence
1  ( 1 )  " "
2  ( 1 )  " "
3  ( 1 )  " "
4  ( 1 )  " "
5  ( 1 )  "*"
6  ( 1 )  "*"
7  ( 1 )  "*"
8  ( 1 )  "*"
9  ( 1 )  "*"
10  ( 1 ) "*"
```

In [29]:

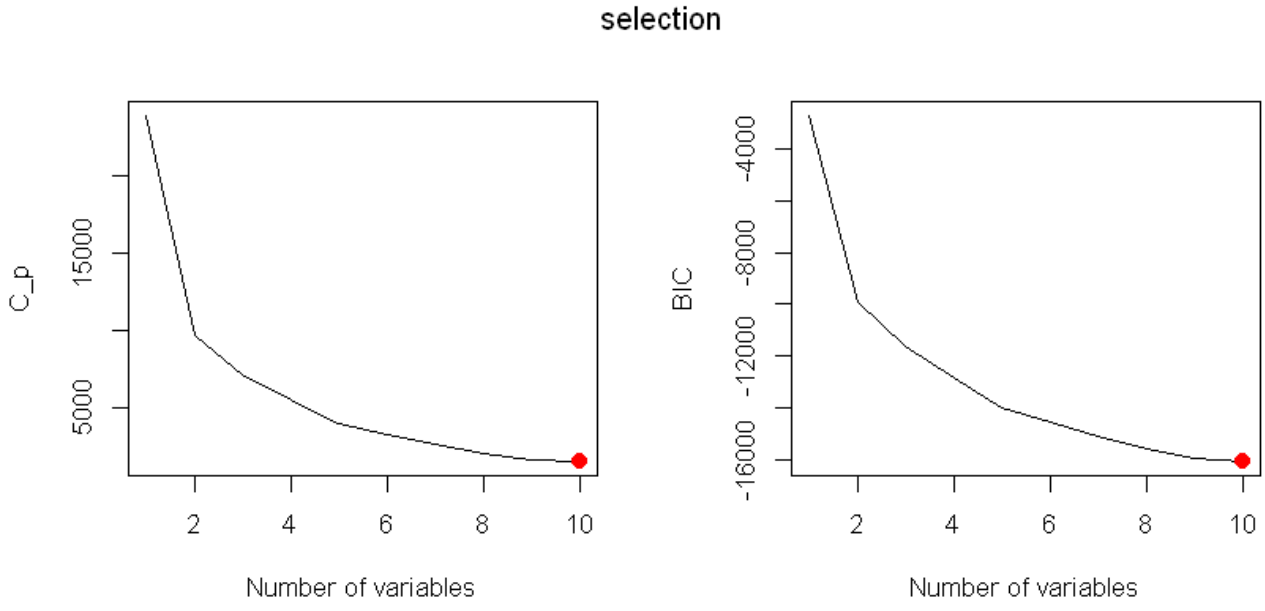
```
mincp<-which.min(reg.summary$cp)
```

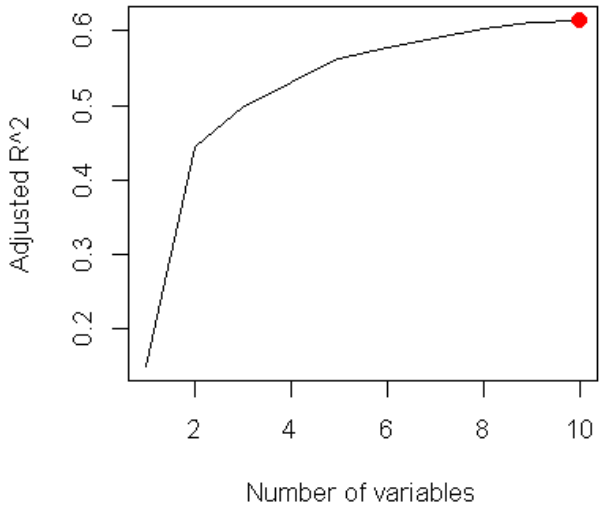
```
coef(regfit.full, mincp)
```

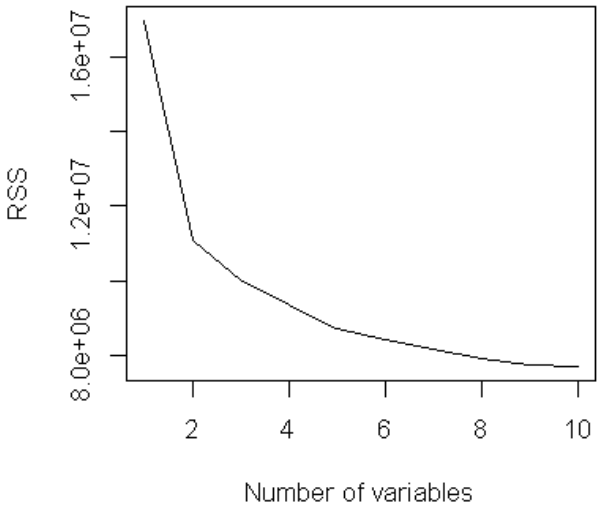
(Intercept)
2.47235315365667
mean_fie
0.0228447569810958
wtd_range_atomic_radius
-0.124641182864429
std_atomic_radius
0.340726603933943
wtd_gmean_ElectronAffinity
-0.23631893234114
std_FusionHeat
-0.503176535939115
mean_ThermalConductivity
0.247618629751339
gmean_ThermalConductivity
-0.298322624415931
entropy_ThermalConductivity
17.9535362321785
wtd_range_ThermalConductivity
0.148115708694414
wtd_std_Valence
-16.495102792937

In [30]:

```
par(mfrow = c(2, 2))
plot(reg.summary$cp, xlab = "Number of variables", ylab = "C_p", type ="l")
points(which.min(reg.summary$cp), reg.summary$cp[which.min(reg.summary$cp)], col = "red", cex = 2,
pch = 20)
plot(reg.summary$bic, xlab = "Number of variables", ylab = "BIC", type ="l")
points(which.min(reg.summary$bic), reg.summary$bic[which.min(reg.summary$bic)], col = "red", cex =
2,pch = 20)
plot(reg.summary$adjr2, xlab = "Number of variables", ylab = "Adjusted R^2", type = "l")
points(which.max(reg.summary$adjr2), reg.summary$adjr2[which.max(reg.summary$adjr2)], col = "red",
cex =2, pch = 20)
plot(reg.summary$rss, xlab = "Number of variables", ylab = "RSS", type ="l")
mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise

selection", side = 3, line = -2, outer = TRUE)
```







The abve plot signifies the following measures,

- Mallow's CP :- This metric helps us to know how precise our model is in the prediciton. Lower the value, more precise the prediction.
- Bayesian information criterion(BIC) :- It is a criterion for model selection among a finite set of models; the model with the lowest BIC is preferred. It is based, in part, on the likelihood function and it is closely related to the Akaike information criterion (AIC).When fitting models, it is possible to increase the likelihood by adding parameters, but doing so may result in overfitting. BIC attempts to resolve this problem by introducing a penalty term for the number of parameters in the model; the penalty term is larger in BIC than in AIC.
- Adjusted R-squared :- The adjusted r-square is a standardized indicator of r-square, adjusting for the number of predictor variables. This shows the standardized variance of the independent variables on the dependent variable in regression analysis. The adjusted r-square includes the degrees of freedom for the statistical model, which is the total number of variables minus one.
- Residual sum of squares(RSS) :- The sum of the squares of residuals (deviations predicted from actual empirical values of data). It is a measure of the discrepancy between the data and an estimation model. A small RSS indicates a tight fit of the model to the data. It is used as an optimality criterion in parameter selection and model selection.

As we can see from the above plots, when the number of variables is 10, RSS, BIC and CP are optimally minimum and ajusted r squared is maximum. Thus we can justify that our choice of 10 features is right. Below, we subset the data based on the features selected from regsubsets and pass it to a linear model later on.

In [31]:

```
features <-
c('mean_fie','wtd_range_atomic_radius','std_atomic_radius','wtd_gmean_ElectronAffinity','std_Fusion
Heat','mean_ThermalConductivity','gmean_ThermalConductivity','entropy_ThermalConductivity','wtd_rar
ge_ThermalConductivity','wtd_std_Valence','critical_temp')
data_train_linearmodel <- data_train_split[,(colnames(data_train_split) %in% features)]
```

In our second approach, we will build a random forest classifier and use these features to build a XGboost model.

Random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. Permutation Importance or percentage increase in MSE is assessed for each feature by removing the association between that feature and the target. This is achieved by randomly permuting the values of the feature and measuring the resulting increase in error. The influence of the correlated features is also removed. If the percentage increase in MSE is high, removing that feature from the model will result in a higher increase of MSE which is not what we want. (This will take some time to run)

In [32]:

```
rf <- randomForest(critical_temp ~ ., data=data_train_split, importance=TRUE,
proximity=TRUE)
```

In [33]:

```
print(rf)
```

```
Call:
 randomForest(formula = critical_temp ~ ., data = data_train_split,      importance = TRUE, proximi
ty = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 9

          Mean of squared residuals: 88.90356
                    % Var explained: 92.41
```

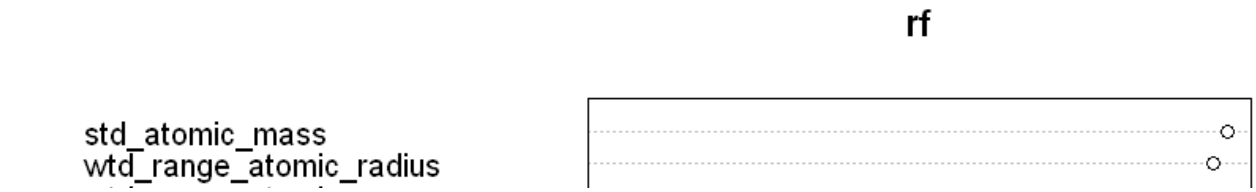
In [34]:

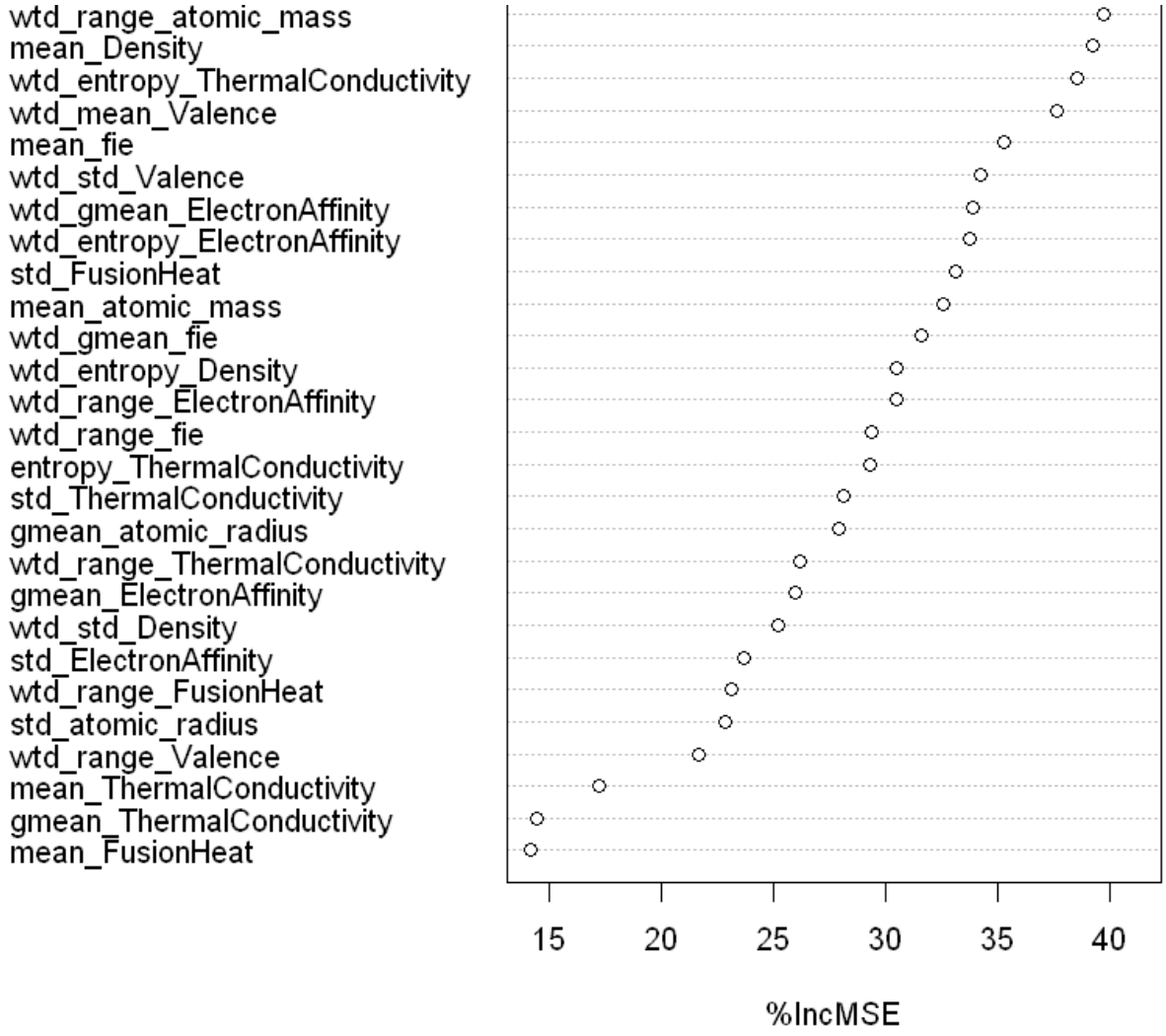
```
importance(rf,type=1)
```

	%IncMSE
mean_atomic_mass	32.58546
wtd_range_atomic_mass	39.71077
std_atomic_mass	41.21903
mean_fie	35.27856
wtd_gmean_fie	31.62645
wtd_range_fie	29.37492
gmean_atomic_radius	27.91675
wtd_range_atomic_radius	40.61313
std_atomic_radius	22.84707
mean_Density	39.25698
wtd_entropy_Density	30.52375
wtd_std_Density	25.18261
gmean_ElectronAffinity	25.96128
wtd_gmean_ElectronAffinity	33.89692
wtd_entropy_ElectronAffinity	33.72171
wtd_range_ElectronAffinity	30.50241
std_ElectronAffinity	23.72070
mean_FusionHeat	14.18440
wtd_range_FusionHeat	23.14258
std_FusionHeat	33.11174
mean_ThermalConductivity	17.25299
gmean_ThermalConductivity	14.46788
entropy_ThermalConductivity	29.30666
wtd_entropy_ThermalConductivity	38.54180
wtd_range_ThermalConductivity	26.19346
std_ThermalConductivity	28.12605
wtd_mean_Valence	37.61984
wtd_range_Valence	21.66787
wtd_std_Valence	34.22607

In [35]:

```
varImpPlot(rf,type=1)
```





As per the above explanation, as the %IncMSE increases, more important is the variable in the model. Thus as a threshold, we will keep 20 as a threshold and remove the features with a %IncMSE of less than 20.

In [36]:

```
features <- c('mean_FusionHeat','mean_ThermalConductivity','gmean_ThermalConductivity')
```

In [37]:

```
data_randomforest <- data_train_split[, !(colnames(data_train_split) %in% features)]
```

4. Model Development

In this section of the assignment, we develop or train our models using the training data and test their performance on the test data. In essence we will be building 4 models,

- XGBoost
- Multiple linear regression
- Lasso regression
- Ridge regression

We first begin with building an XGBoost model. XGBoost is an ensemble learning method. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.The models that form the ensemble, also known as base learners, could be either from the same learning algorithm or different learning algorithms. We use predominantly decision trees as the base learner, followed by boosting. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of

the residuals.

Boosting consists of three simple steps: An initial model F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0) A new model h1 is fit to the residuals from the previous step Now, F0 and h1 are combined to give F1, the boosted version of F0. The mean squared error from F1 will be lower than that from F0: $$ F_{1}(x) = F_{0}(x) + h_{1}(x)$$ To improve the performance of F1, we could model after the residuals of F1 and create a new model F2: $$ F_{2}(x) = F_{1}(x) + h_{2}(x)$$

The exciting part of XGBoost which results in a high accuracy and consistency for majority of tasks i s in the optimization of the objective function. XGBoost uses gradient descent to optimize its loss function. As seen above, $h_{m}(x)$ represents the mean residual at each terminal node of the tree. In gradient boosting, the average gradient component would be computed.

The following steps are involved in gradient boosting:

- F0(x) – with which we initialize the boosting algorithm
- The gradient of the loss function is computed iteratively
- Each hm(x) is fit on the gradient obtained at each step
- The multiplicative factor γm for each terminal node is derived and the boosted model Fm(x) is defined

This is what makes XGBoost so accurate and consistent. For faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. Data is sorted and stored in in-memory units called blocks. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again. This feature also serves useful for steps like split finding and column sub-sampling.

Below, we implement the XGBoost algorithm on the subset generated from the random forest. We require to convert the train and test data to a xgb.DMatrix as it is recommended from the xgboost library.

In [38]:

```
feature_names <- names(data_randomforest)
```

In [39]:

```
colnames(data_randomforest)
```

'mean_atomic_mass' 'wtd_range_atomic_mass' 'std_atomic_mass' 'mean_fie' 'wtd_gmean_fie' 'wtd_range_fie' 'gmean_atomic_radius' 'wtd_range_atomic_radius' 'std_atomic_radius' 'mean_Density' 'wtd_entropy_Density' 'wtd_std_Density' 'gmean_ElectronAffinity' 'wtd_gmean_ElectronAffinity' 'wtd_entropy_ElectronAffinity' 'wtd_range_ElectronAffinity' 'std_ElectronAffinity' 'wtd_range_FusionHeat' 'std_FusionHeat' 'entropy_ThermalConductivity' 'wtd_entropy_ThermalConductivity' 'wtd_range_ThermalConductivity' 'std_ThermalConductivity' 'wtd_mean_Valence' 'wtd_range_Valence' 'wtd_std_Valence' 'critical_temp'

In [40]:

```
set.seed(123)
test_data_clean <- test_data[,(colnames(test_data) %in% colnames(data_randomforest))]
```

In [41]:

```
set.seed(123)
dtrain <- xgb.DMatrix(data=as.matrix(data_randomforest),label=data_randomforest$critical_temp)
dtest <- xgb.DMatrix(data=as.matrix( test_data_clean), label=test_data_clean$critical_temp)
```

Below, we initialize a list of parameters which we will pass to the XGBoost function. These consists of :-

- Booster which is linear
- Ojective which is linear regresison
- number of sub samples
- maximum depth of trees
- Number of samples by trees
- eta which is the learning rate for gradient descent
- eval metric which is a measure of performance; in this case we choose rmse
- minimum child weight

In [42]:

```
param <- list(booster = "gblinear"
              , objective = "reg:linear"
              , subsample = 0.5
              , max_depth = 16
              , colsample_bytree = 0.8
              , eta = 0.02
              , eval_metric = 'rmse'
              , min_child_weight = 1)
```

Below, we will cross validate the model on the train data set to get our optimum number of rounds for which the xgboost performs the best.

In [43]:

```
xgb_cv <- xgb.cv(data=dtrain,
                 params=param,
                nrounds=100,
                prediction=TRUE,
                maximize=FALSE,
                nfold=10,
                early_stopping_rounds = 30,
                print_every_n = 5
)
```

```
[1] train-rmse:39.446626+0.052997 test-rmse:39.443740+0.505365
Multiple eval metrics are present. Will use test_rmse for early stopping.
Will train until test_rmse hasn't improved in 30 rounds.

[6] train-rmse:29.553819+0.037648 test-rmse:29.553096+0.355522
[11] train-rmse:26.526614+0.031931 test-rmse:26.526803+0.317921
[16] train-rmse:24.115647+0.027572 test-rmse:24.116138+0.295720
[21] train-rmse:22.155151+0.024911 test-rmse:22.156043+0.275684
[26] train-rmse:20.557287+0.022717 test-rmse:20.558515+0.257819
[31] train-rmse:19.248179+0.021390 test-rmse:19.249719+0.241547
[36] train-rmse:18.166839+0.020399 test-rmse:18.168588+0.227672
[41] train-rmse:17.264384+0.019277 test-rmse:17.266402+0.216453
[46] train-rmse:16.501634+0.018774 test-rmse:16.503846+0.206884
[51] train-rmse:15.847588+0.018129 test-rmse:15.849993+0.199515
[56] train-rmse:15.278636+0.017721 test-rmse:15.281219+0.193703
[61] train-rmse:14.776235+0.017366 test-rmse:14.779015+0.189059
[66] train-rmse:14.326460+0.017079 test-rmse:14.329365+0.185489
[71] train-rmse:13.918646+0.016876 test-rmse:13.921718+0.182523
[76] train-rmse:13.544691+0.016734 test-rmse:13.547859+0.180130
[81] train-rmse:13.198437+0.016658 test-rmse:13.201725+0.178060
[86] train-rmse:12.875186+0.016588 test-rmse:12.878550+0.176324
[91] train-rmse:12.571464+0.016544 test-rmse:12.574927+0.174700
[96] train-rmse:12.284394+0.016489 test-rmse:12.287955+0.173285
[100] train-rmse:12.065256+0.016439 test-rmse:12.068881+0.172189
```

In [44]:

```
nrounds <- xgb_cv$best_iteration
```

We now train the model on training dataset with the best parameters.

In [45]:

```
xgb <- xgb.train(params = param
                 , data = dtrain
                 , nrounds = nrounds
                 , verbose = 1
                 , print_every_n = 5
                )
```

We now predict on the test data set.

In [46]:

```
preds_test <- predict(xgb,slice(dtest,1:4253))
```

In [47]:

```
test_data_clean$predicted <- preds_test
```

Below we calculate the RMSE between the actual and predicted values for critical temperature for model comparison.

In [48]:

```
rmse_xgboost <- RMSE(test_data_clean$critical_temp,test_data_clean$predicted)
```

We calculate r-squared below for model comparison.

In [49]:

```
r_sq_xgboost <- (cor(test_data_clean$predicted, test_data_clean$critical_temp))^2
```

Multiple linear regression

We now build a multiple linear regression model from the subset generated from the first section of feature selection.

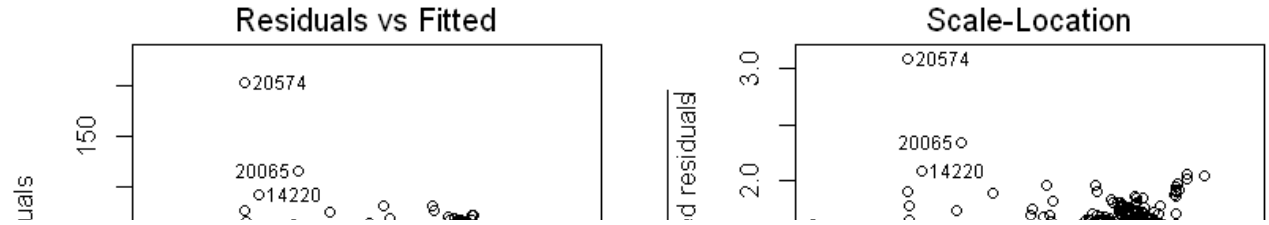
In [50]:

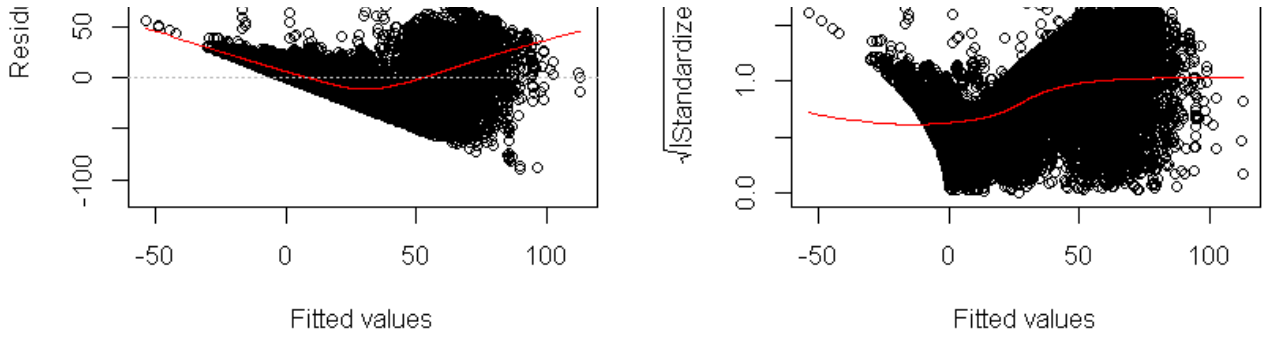
```
linear_model <- lm(critical_temp ~ .,data=data_train_linearmodel)
summary(linear_model)
```

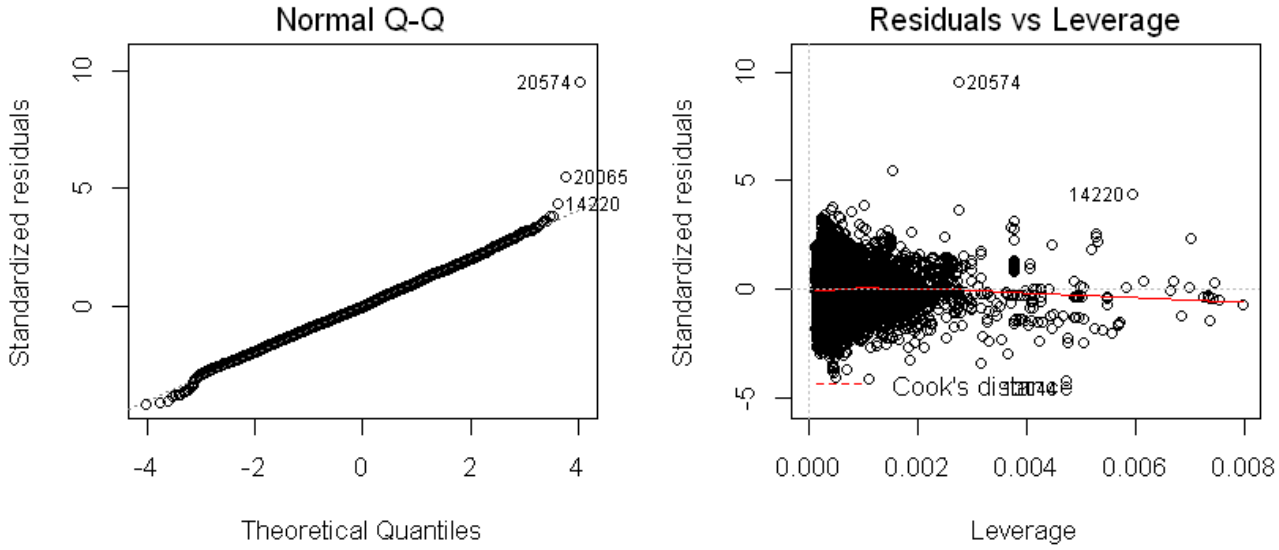
```
Call:
lm(formula = critical_temp ~ ., data = data_train_linearmodel)

Residuals:
    Min      1Q  Median      3Q     Max 
-89.573 -14.498  -0.941  14.532 202.389 

Coefficients:
                               Estimate Std. Error t value Pr(>|t|)    
(Intercept)                    2.472353   1.849189   1.337    0.181    
mean_fie                       0.022845   0.002147  10.641   <2e-16 ***
wtd_range_atomic_radius       -0.124641   0.005523 -22.568   <2e-16 ***
std_atomic_radius              0.340727   0.011676  29.181   <2e-16 ***
wtd_gmean_ElectronAffinity    -0.236319   0.005534 -42.703   <2e-16 ***
std_FusionHeat                -0.503177   0.022912 -21.961   <2e-16 ***
mean_ThermalConductivity       0.247619   0.006577  37.651   <2e-16 ***
gmean_ThermalConductivity     -0.298323   0.008348 -35.734   <2e-16 ***
entropy_ThermalConductivity   17.953536   0.575451  31.199   <2e-16 ***
wtd_range_ThermalConductivity  0.148116   0.005063  29.253   <2e-16 ***
wtd_std_Valence              -16.495103   0.401626 -41.071   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.27 on 16999 degrees of freedom
Multiple R-squared:  0.6141, Adjusted R-squared:  0.6139 
F-statistic:  2705 on 10 and 16999 DF,  p-value: < 2.2e-16
```

In [51]:

```
par(mfcol=c(2,2))
plot(linear_model)
```







From the above plot, we can make the following interpretations,

- The residual vs fitted plot tells us about the linear assumption. It shows if residuals have non-linear patterns. If you find equally spread residuals around a horizontal line without distinct patterns, that is a good indication you have linear relationships. However, if the relationship between predictors and an response variable is non-linear, an obvious pattern could show up in this plot if the model cannot capture the non-linearity. However, in this case, we can see that the residuals are not scattered evenly and this can lead to a non-linear relationship between critical temperature and our predictors.
- The Q-Q plot (i.e., quantile-quantile plot) is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a Normal. In the case of linear regression analysis, we assume that residual is normally distributed with constant variance and mean equal to zero. We can see that most residuals are normally distributed except for a few data points.
- The scale location plot is used to check for the assumption of equal variance by showing if residuals are spread equally along the ranges of predictors. It is good if we can see a horizontal line with equally (randomly) spread points. The scale-location plot shows that the residuals appear randomly spread.It can be seen from the plot that the scale location plot is quite linear althought not being perfectly linear, thus giving a good reason for us to believe that the residuals are spread equally.
- The residual leverage plot helps us identify influential data samples. we look for outlying values at the upper right corner or at the lower right corner. Samples located in those places can be influential against a regression line.We use Cook's distance for indicating whether a sample is influential or not. When samples are influential, they are outside Cook's distance. However, in the plot, most of the points seem to be within this range and thus there are no influential cases observed.

In [52]:

```
predictions <- predict(linear_model,test_data)
```

In [53]:

```
rmse_linearmodel <- RMSE(test_data$critical_temp,predictions)
```

In [54]:

```
r_sq_linearmodel <- summary(linear_model)$adj.r.squared
```

Ridge regression and Lasso regression

We now build a ridge regression to perform ordinary least squares regression. The main difference between OLS and ridge and lasso is that ridge and lasso adds a penalty term which shrinks the estimate of the coefficients towards zero for ridgewhile in lasso it may set some coefficients to 0. Below, we begin by building a ridge regression model followed by a lasso regression model.

In [55]:

```
train.mat <- model.matrix(critical_temp ~ ., data = data_train_split)[,-1]
test.mat <- model.matrix(critical_temp ~ ., data = test_data_split)[,-1]
```

We create a sequence of values for lambda so that we can select the optimal value by cross validation on the training data set.

In [56]:

```
grid <- 10^seq(4, -2, length = 100)
```

We now fit a ridge regression model on the data, cross validate to get the best lambda values and predict on the test data set. This is done by passing the training data to glmnet function and setting alpha to 1.

In [57]:

```
fit.ridge <- glmnet(train.mat, train_data$critical_temp, alpha = 1, lambda = grid)
cv.ridge <- cv.glmnet(train.mat, train_data$critical_temp, alpha = 1, lambda = grid)
bestlam.ridge <- cv.ridge$lambda.min
bestlam.ridge
```

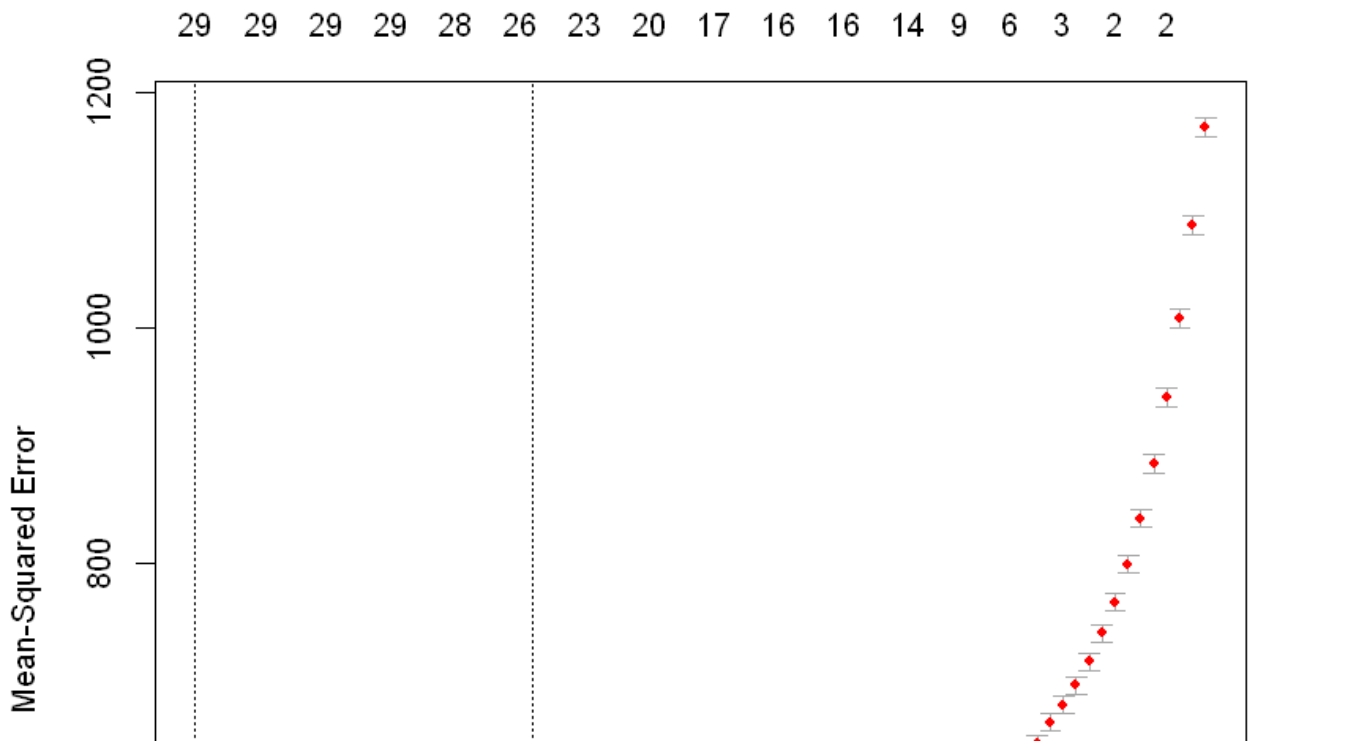
0.0114975699539774

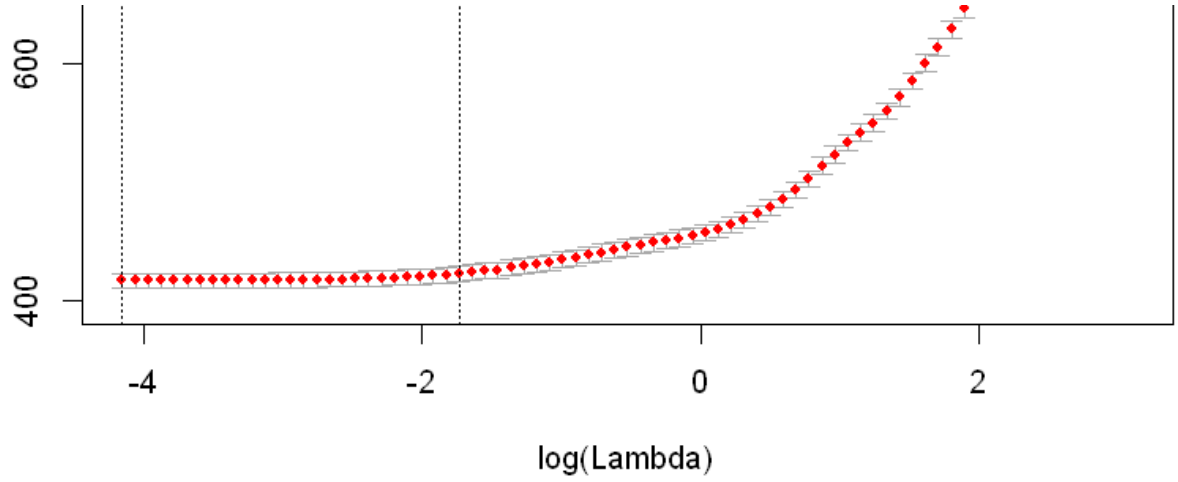
In [58]:

```
cv.ridge <- cv.glmnet(train.mat, train_data$critical_temp , alpha = 1)
```

In [59]:

```
plot(cv.ridge)
```





We now pass the test data to the predict function in ridge regression and set lambda to the optimal value of 0.00114975699539774.

In [60]:

```
predict(fit.ridge, s = bestlam.ridge, type = "coefficients")
```

```
30 x 1 sparse Matrix of class "dgCMatrix"
                                        1
(Intercept)                  -4.106479e+01
mean_atomic_mass              1.598502e-01
wtd_range_atomic_mass        -7.582688e-02
std_atomic_mass               2.220922e-01
mean_fie                      4.296083e-02
wtd_gmean_fie                 1.755980e-02
wtd_range_fie                -1.934546e-02
gmean_atomic_radius           9.829875e-02
wtd_range_atomic_radius      -1.270657e-01
std_atomic_radius             3.157543e-01
mean_Density                 -2.352291e-03
wtd_entropy_Density           4.724817e-01
wtd_std_Density              -9.759045e-04
gmean_ElectronAffinity        9.742414e-02
wtd_gmean_ElectronAffinity   -2.358783e-01
wtd_entropy_ElectronAffinity -2.271874e+01
wtd_range_ElectronAffinity   -1.121972e-01
std_ElectronAffinity          1.130978e-01
mean_FusionHeat               2.414484e-01
wtd_range_FusionHeat         -1.206500e-01
std_FusionHeat               -6.080062e-01
mean_ThermalConductivity      1.533369e-01
gmean_ThermalConductivity    -2.420343e-01
entropy_ThermalConductivity   2.190596e+01
wtd_entropy_ThermalConductivity 8.284612e+00
wtd_range_ThermalConductivity 1.655530e-01
std_ThermalConductivity       1.151365e-01
wtd_mean_Valence             -1.133003e+00
wtd_range_Valence             6.038129e+00
wtd_std_Valence              -1.773346e+01
```

In [61]:

```
# prediction using ridge model for original test data
pred.ridge <- predict(fit.ridge, s = bestlam.ridge, newx = test.mat)
rmse_ridge <- RMSE(pred.ridge,test_data$critical_temp)
r_sq_ridge <- (cor(pred.ridge, test_data$critical_temp))^2
```

We now build a lasso regression model which is done by setting the value of alpha to 0.

In [62]:

```
fit.lasso <- glmnet(train.mat, train_data$critical_temp, alpha = 0, lambda = grid)
cv.lasso <- cv.glmnet(train.mat, train_data$critical_temp, alpha = 0, lambda = grid)
```

```
bestlam.lasso <- cv.lasso$lambda.min
bestlam.lasso
```

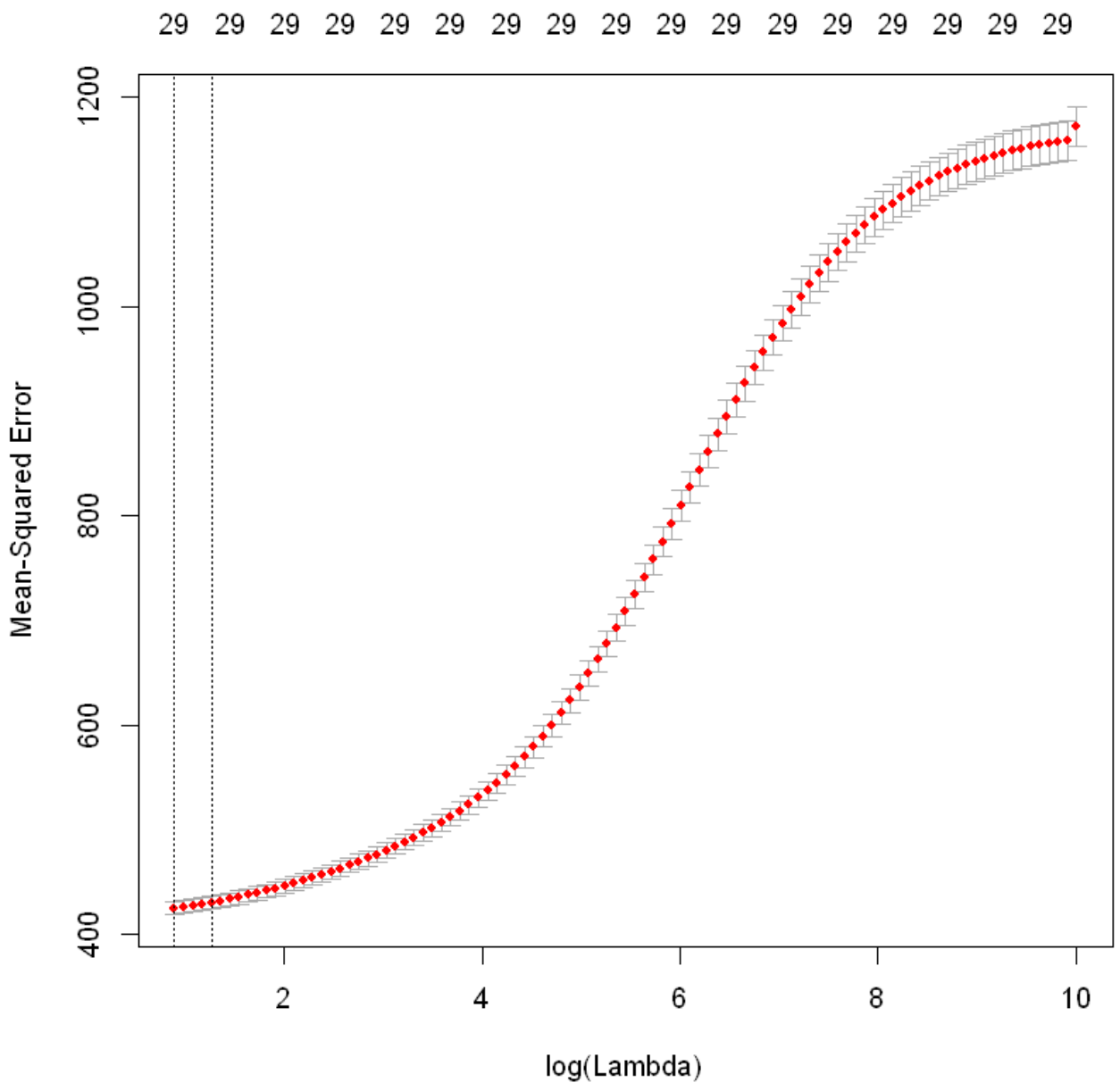
0.0265608778294668

In [63]:

```
cv.lasso <- cv.glmnet(train.mat, train_data$critical_temp , alpha = 0)
```

In [64]:

```
plot(cv.lasso)
```



In [65]:

```
predict(fit.lasso, s = bestlam.lasso, type = "coefficients")
```

```
30 x 1 sparse Matrix of class "dgCMatrix"
                                       1
(Intercept)                -4.258620e+01
mean_atomic_mass            1.618141e-01
wtd_range_atomic_mass      -7.662373e-02
std_atomic_mass             2.236742e-01
```

```
std_atomic_mass                 2.250742e-01
mean_fie                        4.301589e-02
wtd_gmean_fie                   1.904295e-02
wtd_range_fie                  -2.007858e-02
gmean_atomic_radius             1.036252e-01
wtd_range_atomic_radius        -1.270026e-01
std_atomic_radius               3.163050e-01
mean_Density                   -2.383991e-03
wtd_entropy_Density             7.298661e-01
wtd_std_Density                -9.869418e-04
gmean_ElectronAffinity          9.921545e-02
wtd_gmean_ElectronAffinity     -2.351682e-01
wtd_entropy_ElectronAffinity   -2.301404e+01
wtd_range_ElectronAffinity     -1.164011e-01
std_ElectronAffinity            1.151812e-01
mean_FusionHeat                 2.499184e-01
wtd_range_FusionHeat           -1.265067e-01
std_FusionHeat                 -6.087976e-01
mean_ThermalConductivity        1.598712e-01
gmean_ThermalConductivity      -2.479418e-01
entropy_ThermalConductivity     2.167542e+01
wtd_entropy_ThermalConductivity 8.452602e+00
wtd_range_ThermalConductivity   1.668783e-01
std_ThermalConductivity         1.103346e-01
wtd_mean_Valence               -1.309341e+00
wtd_range_Valence               6.296109e+00
wtd_std_Valence                -1.775474e+01
```

In [66]:

```
# prediction using ridge model for original test data
pred.lasso <- predict(fit.lasso, s = bestlam.lasso, newx = test.mat)
rmse_lasso <- RMSE(pred.lasso,test_data$critical_temp)
r_sq_lasso <- (cor(pred.lasso, test_data$critical_temp))^2
```

5. Model Comparison

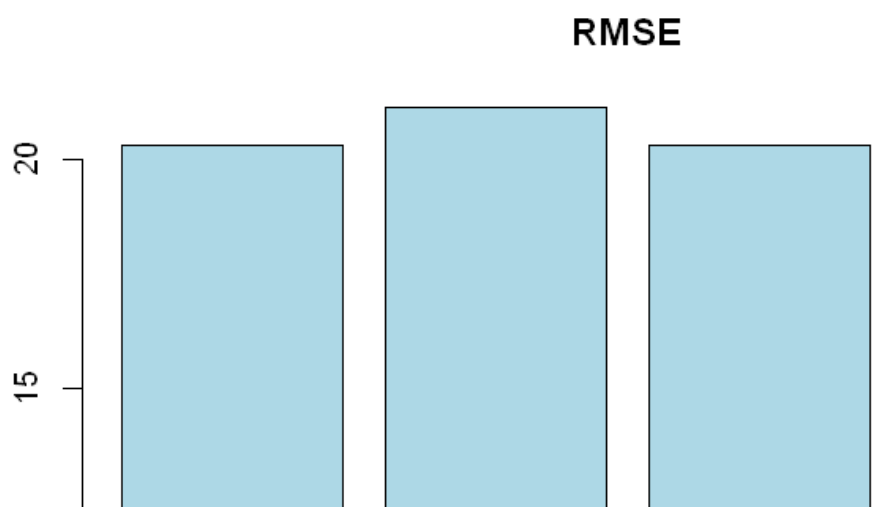
We will now compare various error metrics of our model and derive which works best for our prediction. I have created three visualizations for RMSE, MSE and R-squared for the four models as shown below.

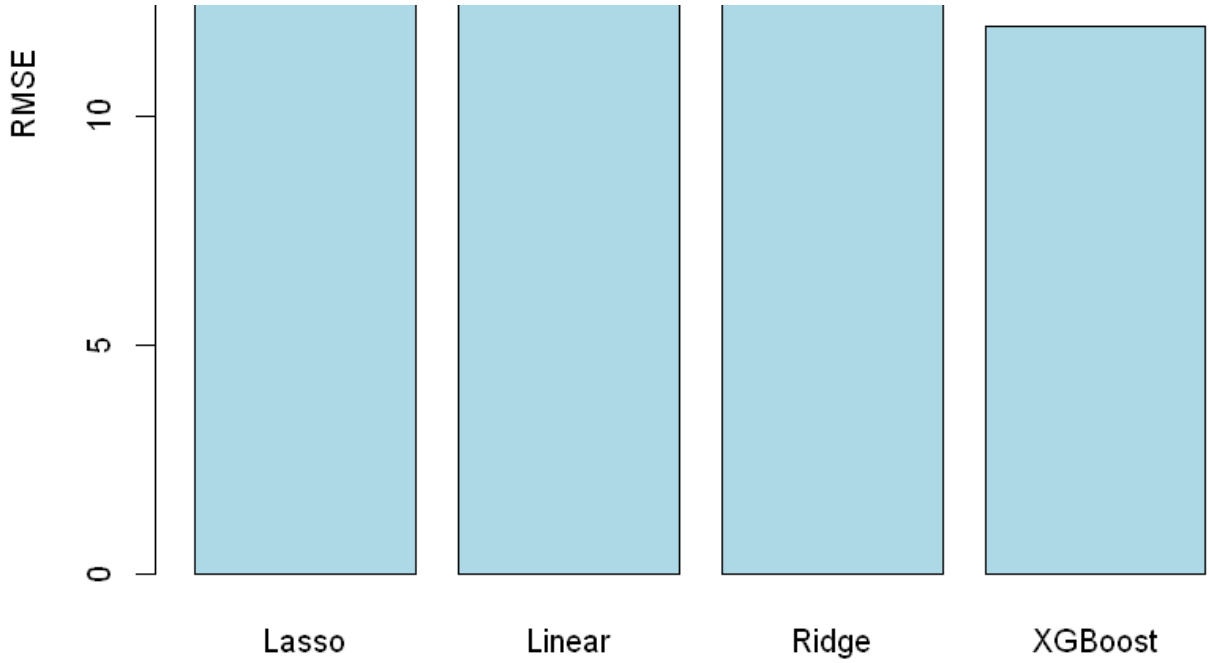
In [67]:

```
rmse <- c(rmse_lasso,rmse_linearmodel,rmse_ridge,rmse_xgboost)
r_squared <- c(r_sq_lasso,r_sq_linearmodel,r_sq_ridge,r_sq_xgboost)
mse <- c(rmse_lasso^2,rmse_linearmodel^2,rmse_ridge^2,rmse_xgboost^2)
```

In [68]:

```
barplot(rmse,main="RMSE",
   names.arg=c("Lasso", "Linear", "Ridge","XGBoost"),ylab='RMSE',col='lightblue')
```

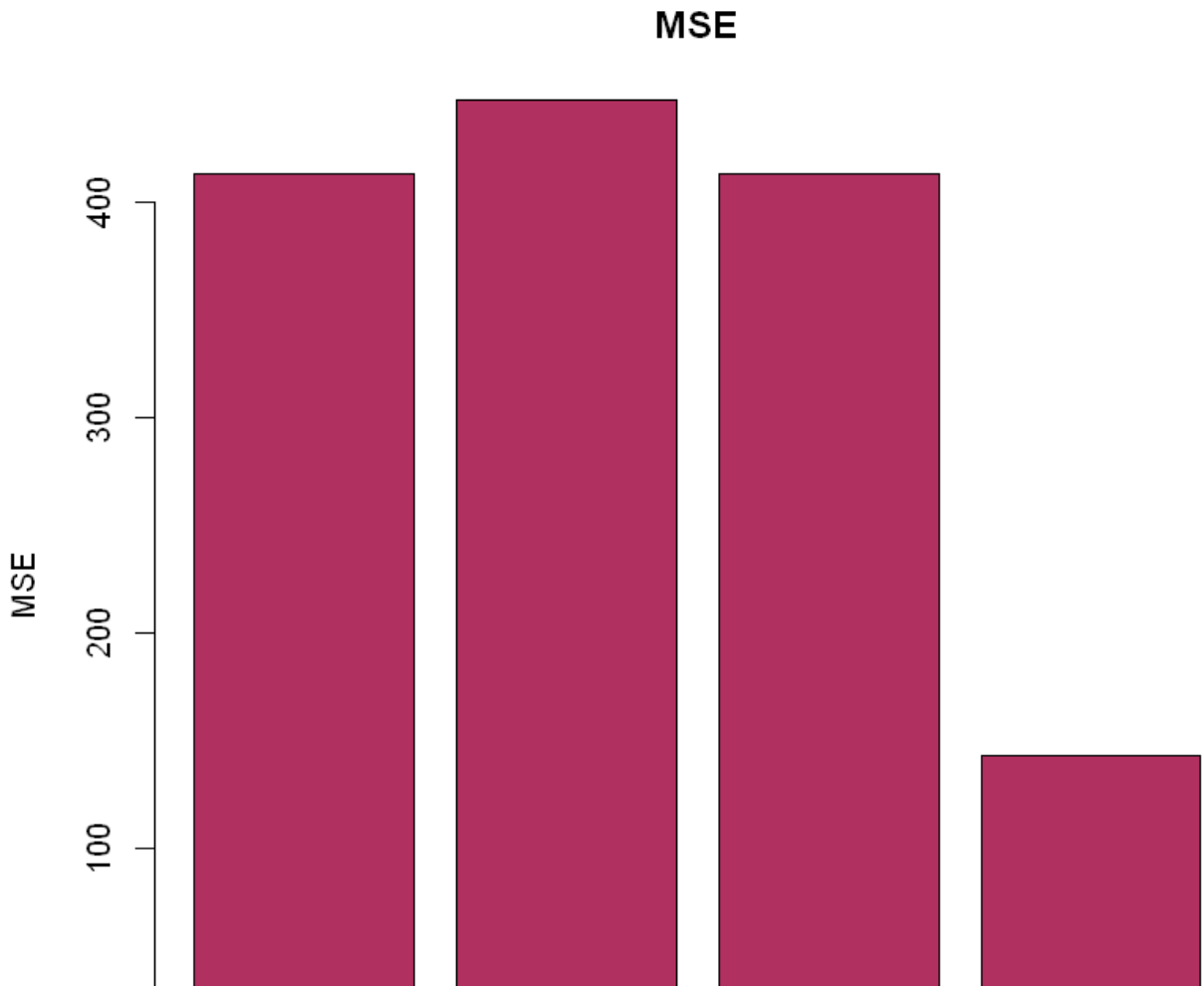


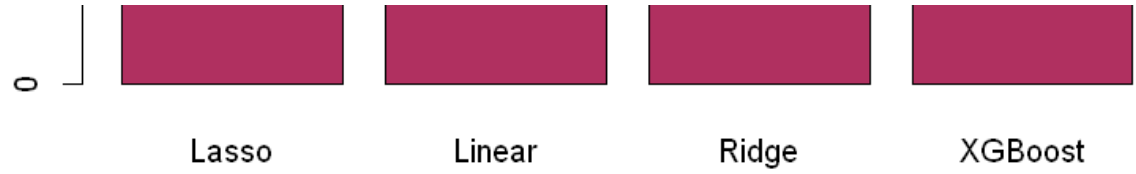


We can see that RMSE is the least for XGBoost followed by Ridge, Lasso and multiple linear model.

In [69]:

```
barplot(mse,main="MSE",
   names.arg=c("Lasso", "Linear", "Ridge","XGBoost"),ylab='MSE',col='maroon')
```

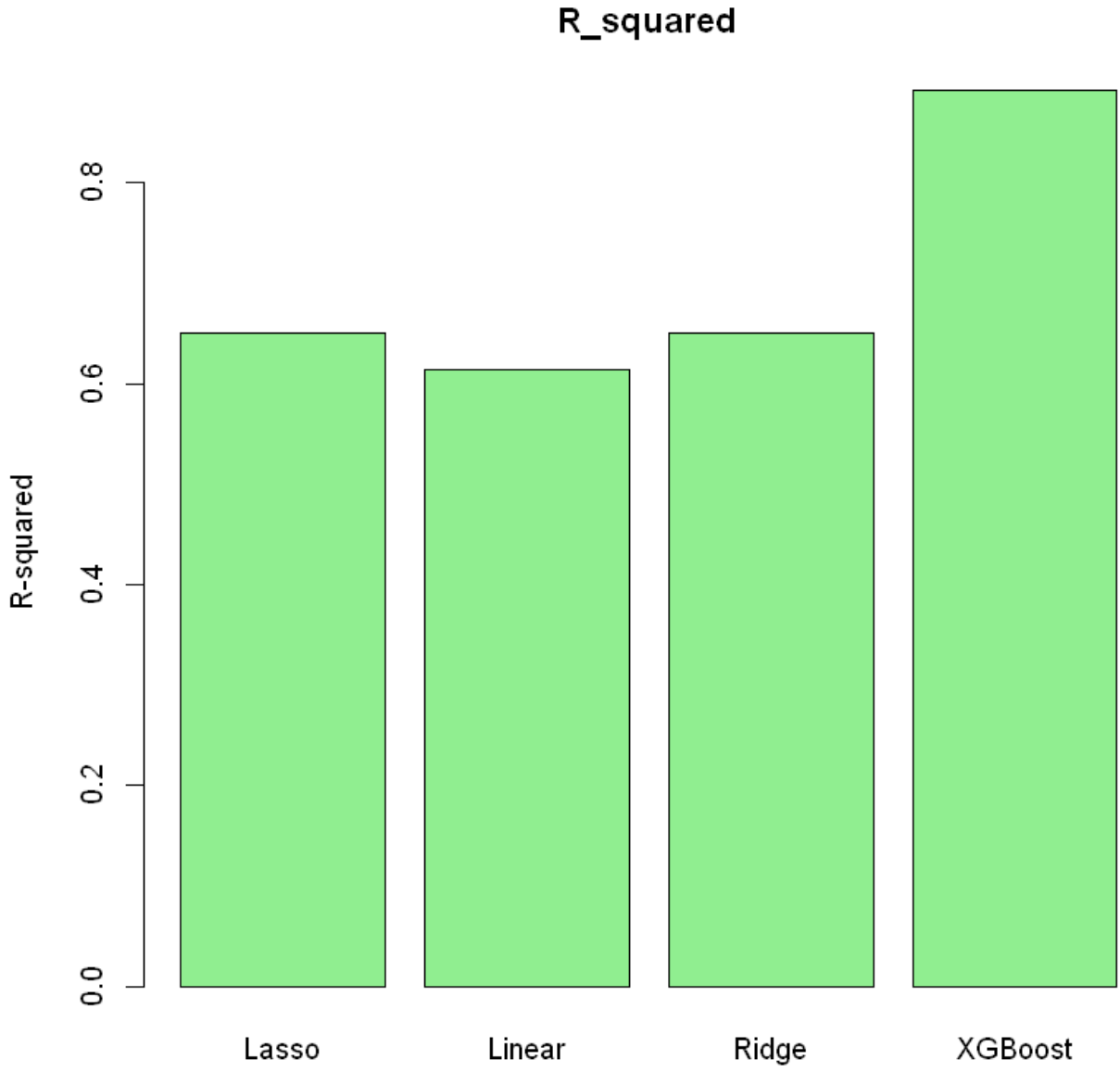




The same trend follows for MSE as well.

In [70]:

```
barplot(r_squared,main="R_squared",
   names.arg=c("Lasso", "Linear", "Ridge","XGBoost"),ylab='R-squared',col='lightgreen')
```



XGBoost seems to have the highest R_squared(around 0.9), followed by ridge and lasso and linear regression models. Thus summarizing our model comparison phase, we can say that

- XGBoost performs the best in comparison with the other models.
- Lasso and Ridge regression have almost the same performance with Ridge performing better than Lasso.

- Multiple linear model performs the worst in comparison with the other 3 models.

6. Conclusion

In concusion, we can confirm that the foundings in the paper were verified ,i.e. performance of XGBoost for predicting critical temperature is much better when compared to other models. However, it seems that lasso and ridge regression mayhave been better model candidates when compared to the linear regression model as they perform much better when compared to linear model.

Thus by selecting the below features,

- mean_atomic_mass
- wtd_range_atomic_mass
- std_atomic_mass
- mean_fie
- wtd_gmean_fie
- wtd_range_fie
- gmean_atomic_radius
- wtd_range_atomic_radius
- std_atomic_radius
- mean_Density
- wtd_entropy_Density
- wtd_std_Density
- gmean_ElectronAffinity
- wtd_gmean_ElectronAffinity
- wtd_entropy_ElectronAffinity
- wtd_range_ElectronAffinity
- std_ElectronAffinity
- wtd_range_FusionHeat
- std_FusionHeat
- entropy_ThermalConductivity
- wtd_entropy_ThermalConductivity
- wtd_range_ThermalConductivity
- std_ThermalConductivity
- wtd_mean_Valence
- wtd_range_Valence
- wtd_std_Valence

and passing them to a XGBoost model, would be our best candidate to predict critical temperature of a superconductor.

7. References

https://www.r-bloggers.com/ridge-regression-and-the-lasso/
https://www.r-bloggers.com/variable-importance-plot-and-variable-selection/
https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

In [ ]: