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

Student information

- Family Name: Vydyula
- Given Name: Shyama Malhari
- Student ID: 29860644
- Student email: svyd0001@student.monash.edu

Programming Language: R 3.5.1 in Jupyter Notebook

R Libraries used:

- psych
- ggplot2
- purrr
- tidyr
- ggcorrplot
- stringr
- dplyr
- data.table
- xgboost
- readr
- stringr
- caret
- usdm
- fmsb
- VIF
- ggpubr

Table of Contents

In [1]:

```
#install.packages('caret')
#install.packages('ggplot2')
#install.packages('usdm')
#install.packages('fmsb')
#install.packages('VIF')
#install.packages('psych')
#install.packages('ggpubr')
#install.packages('leaps')
```

In [96]:

```
library(tidyverse)
library(caret)
library(leaps)
library(psych)
library(ggplot2)
library(purrr)
library(tidyr)
library(ggcorrplot)
library(stringr)
library(dplyr)
library(data.table)
library(xgboost)
library(readr)
library(stringr)
library(caret)
library(usdm)
library(fmsb)
library(VIF)
library(ggpubr)
```

1. Introduction

`According to Wikipedia:`

"**Superconductivity** is the set of physical properties observed in certain materials, wherein electrical resistance vanishes and from which magnetic flux fields are expelled. Any material exhibiting these properties is a superconductor. Unlike an ordinary metallic conductor, whose resistance decreases gradually as its temperature is lowered even down to near absolute zero, a superconductor has a characteristic critical temperature below which the resistance drops abruptly to zero. An electric current through a loop of superconducting wire can persist indefinitely with no power source"

Superconductors have immense applications in the current world such as:

- Used in Magnetic Resonanc Imaging or MRI, which is used in the healthcare industry for physical examination of the human body.
- Used in Particle Acelerators, machines that use EM field to propel charges, at CERN.
- Used in Superconducting Quantum Interference Devices or (SQUIDs)

Since the conductance of a superconductor depends almost entirely on the ability to identify its 'Critical Temperature', or the temperature at which its resistance drops to zero, the need to identify and measure this temperature using the various chemical properties has become a topic of interest for physicians around the world.

In this project, we will be conducting a statistical analysis to predict the 'Critical Temperature' of a superconductor, given some of its chemical properties.

The project is entirely based on : ***"A Data Driven Statistical Model for Predicting the Critical Temperature of a Superconductor at https://arxiv (https://arxiv). org/pdf/1803.10260.pdf"***

2. Data Exploration

"However the model builder's main problem is often not how to fit a model but rather how to formulate it in the first place".

Chris Chatfield has rightly said in his publication, 'Exploratory Data Analysis' that is wise to begin a statistical analysis with an informal, exploratory examination of the data called exploratory data analysis (abbreviated EDA). The two main objectives are: data description and model-formulation. It is suggested that it is important to see EDA as an integral part of statistical inference. (Chatfield, 1986)

Loading the dataset:

In [12]:

```
#Reading the Dataset using "read" function in R.
super <- read.csv('train.csv',header = TRUE, sep = ',')
```

In [13]:

```
#Dimensions of the Data
dim(super)
```

21263   82

It can be seen from the output that there are <font color = "green">82 columns </font> or variables in this dataset and 21,263 observations.

In [14]:

```
#Structure of the Data
str(super)
```

```
'data.frame':	21263 obs. of  82 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 2
05 171 171 171 ...
 $ 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 2.2
5 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 3
6 31 ...
```

From the above output, it is clear that all the columns that are in the dataset are numerical in type. The first column tells us how many elements are present in the super conductor and the last columns is the critical temperature of that super conductor. The rest of the 80 columns are derived from the 8 basic properties of the super conductor as mentioned in Hamidieh, K. (2018), the foundation reference for this Assignment.

The 8 properties can be described as follows:

Data Description:

1. Atomic Mass : It is the total proton and neutron rest masses measured in Atomic Mass Units (AMU).

2. First Ionization Energy : It is the energy required to remove a valence electron from an element. Measured in kilo-Joules per mole (kJ/mol).

3. Atomic Radius : It is the calculated atomic radius of the element. Measured in picometer (pm).

4. Density : It is the density at standard temperature and pressure. Measured in kilograms per meters cubed (kg/m3).

5. Electron Affinity : The energy required to add an electron to a neutral atom. Measured in kilo-Joules per mole (kJ/mol).

6. Fusion Heat : The energy to change from solid to liquid state without temperature change. Measured in kilo-Joules per mole (kJ/mol).

7. Thermal Conductivity : The thermal conductivity coefficient measured in watts per meter-Kelvin (W/(m x K)).

8. Valence : The number of chemical bonds formed by the elements. It is a numerical value and doesn't have a unit of measurement.

In [15]:

```
#Setting the kernel to allow wide datasets to be displayed without breaks
options(repr.matrix.max.cols=1000, repr.matrix.max.rows=2000)
```

Renaming the columns for lucidity:

In [16]:

```
super1 <- super
```

In [17]:

```
super1 <- super1 %>% rename_at(vars(starts_with("mean_")), funs(str_replace(., 
"mean_", "m_"))) #mean
super1 <- super1 %>% rename_at(vars(starts_with("wtd_mean_")), funs(str_replac
e(., "wtd_mean_", "wm_"))) #weighted mean
super1 <- super1 %>% rename_at(vars(starts_with("gmean_")), funs(str_replace(.
, "gmean_", "gm_"))) #geometric mean
super1 <- super1 %>% rename_at(vars(starts_with("wtd_gmean_")), funs(str_repla
ce(., "wtd_gmean_", "wgm_"))) #weighted geometric mean
super1 <- super1 %>% rename_at(vars(starts_with("entropy_")), funs(str_replace
(., "entropy_", "e_"))) #entropy
super1 <- super1 %>% rename_at(vars(starts_with("wtd_entropy_")), funs(str_rep
lace(., "wtd_entropy_", "we_"))) #weighted entropy
super1 <- super1 %>% rename_at(vars(starts_with("range_")), funs(str_replace(.
, "range_", "r_"))) #range
super1 <- super1 %>% rename_at(vars(starts_with("wtd_range_")), funs(str_repla
ce(., "wtd_range_", "wr_"))) #weighted range
super1 <- super1 %>% rename_at(vars(starts_with("wtd_std_")), funs(str_replace
(., "wtd_std_", "wstd_"))) #weighted standard deviation
```

```
Warning message:
“`is_lang()` is deprecated as of rlang 0.2.0.
Please use `is_call()` instead.
This warning is displayed once per session.”Warning message:
“`lang_modify()` is deprecated as of rlang 0.2.0.
Please use `call_modify()` instead.
This warning is displayed once per session.”
```

In [18]:

```
super1 <- super1 %>% rename_at(vars(contains("atomic_mass")), funs(str_replace
(., "atomic_mass", "am"))) #Atomic Mass
super1 <- super1 %>% rename_at(vars(contains("atomic_radius")), funs(str_repla
ce(., "atomic_radius", "ar"))) #Atomic Radius
super1 <- super1 %>% rename_at(vars(contains("Density")), funs(str_replace(., 
"Density", "d"))) #Density
super1 <- super1 %>% rename_at(vars(contains("ElectronAffinity")), funs(str_re
place(., "ElectronAffinity", "ea"))) #Electron affinity
super1 <- super1 %>% rename_at(vars(contains("FusionHeat")), funs(str_replace(
., "FusionHeat", "fh"))) #Fusion Heat
super1 <- super1 %>% rename_at(vars(contains("ThermalConductivity")), funs(str
_replace(., "ThermalConductivity", "tc"))) #Thermal Conductivity
super1 <- super1 %>% rename_at(vars(contains("Valence")), funs(str_replace(., 
"Valence", "v"))) #Valence
```

Lets check out the first and the last few rows of the dataset using the <font color = "blue"> **head( )** </font> and <font color = "blue">**tail( )**</font> functions.

In [19]:

```
head(super1)
```

number_of_elements	m_am	wm_am	gm_am	wgm_am	e_am	we_am	r_an
4	88.94447	57.86269	66.36159	36.11661	1.181795	1.0623955	122.906
5	92.72921	58.51842	73.13279	36.39660	1.449309	1.0577551	122.906
4	88.94447	57.88524	66.36159	36.12251	1.181795	0.9759805	122.906
4	88.94447	57.87397	66.36159	36.11956	1.181795	1.0222909	122.906
4	88.94447	57.84014	66.36159	36.11072	1.181795	1.1292237	122.906
4	88.94447	57.79504	66.36159	36.09893	1.181795	1.2252028	122.906

In [20]:

```
tail(super1)
```

	number_of_elements	m_am	wm_am	gm_am	wgm_am	e_am	we_am
21258	3	89.38983	89.38983	63.69471	63.69471	0.7825737	0.7825737
21259	4	106.95788	53.09577	82.51538	43.13556	1.1771448	1.2541187
21260	5	92.26674	49.02137	64.81266	32.86775	1.3232866	1.5716301
21261	2	99.66319	95.60910	99.43388	95.46432	0.6908472	0.5301975
21262	2	99.66319	97.09560	99.43388	96.90108	0.6908472	0.6408830
21263	3	87.46833	86.85850	82.55576	80.45872	1.0412701	0.8952292

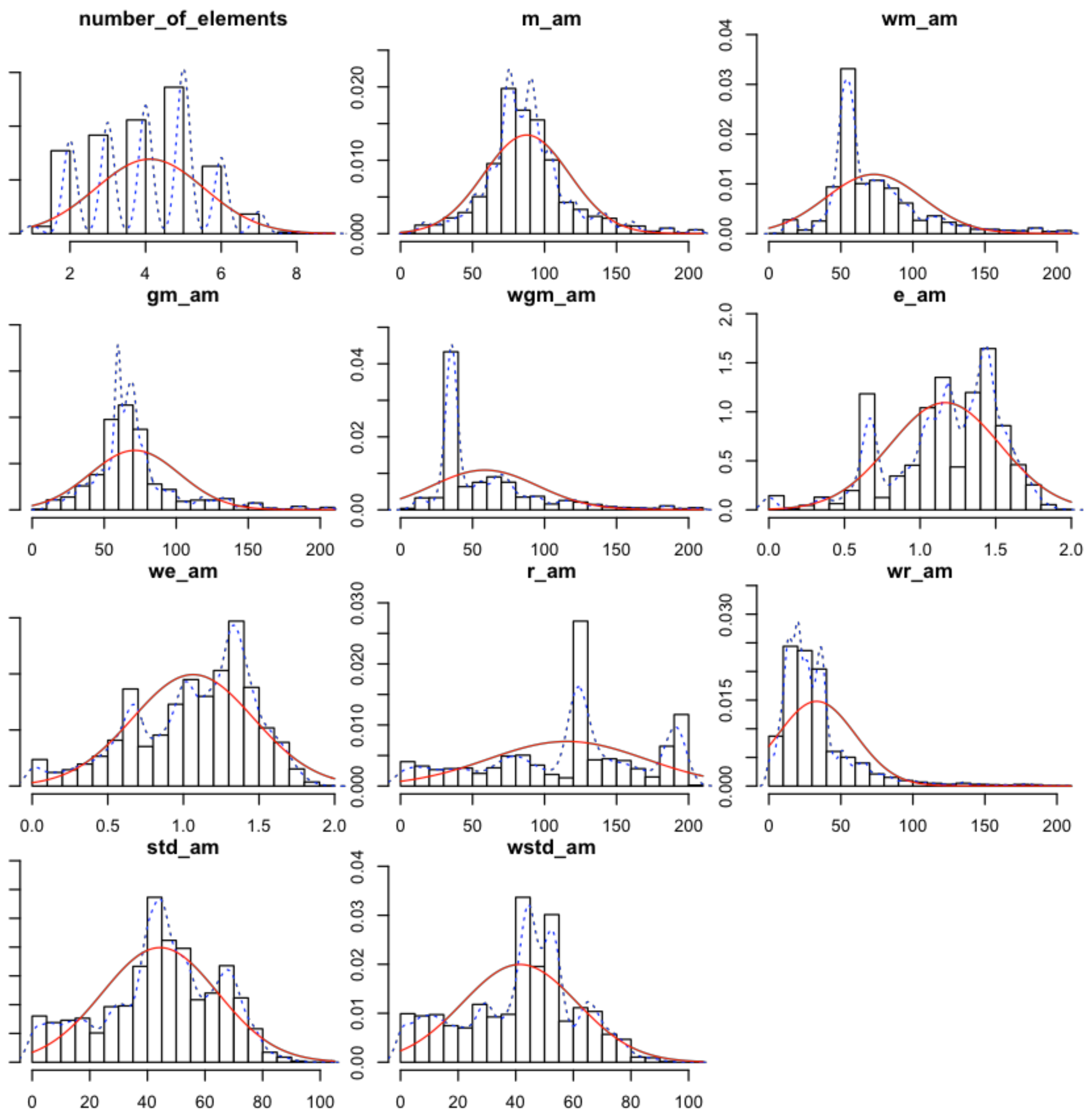
Summary Statistics:

Since the given dataset is an extremely wide dataset, we will compute the summaries of the variables block by block. i.e., Atomic Mass, Entropy, Atomic Radius and so on. This will help us see the output clearly and deduce some information from the same. Let us begin with the first block, which is **"Atomic Mass"** and also inclues **Number of Elements**:

In [21]:

```
summary(super1[1:11])
multi.hist(super1[1:11],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

```
 number_of_elements      m_am             wm_am             gm_am       
 Min.   :1.000      Min.   :  6.941   Min.   :  6.423   Min.   :  
5.321  
 1st Qu.:3.000      1st Qu.: 72.458   1st Qu.: 52.144   1st Qu.: 5
8.041  
 Median :4.000      Median : 84.923   Median : 60.697   Median : 6
6.362  
 Mean   :4.115      Mean   : 87.558   Mean   : 72.988   Mean   : 7
1.291  
 3rd Qu.:5.000      3rd Qu.:100.404   3rd Qu.: 86.104   3rd Qu.: 7
8.117  
 Max.   :9.000      Max.   :208.980   Max.   :208.980   Max.   :20
8.980  
     wgm_am             e_am             we_am             r_am       
 Min.   :  1.961   Min.   :0.0000   Min.   :0.0000   Min.   :  0.0
0  
 1st Qu.: 35.249   1st Qu.:0.9667   1st Qu.:0.7754   1st Qu.: 78.5
1  
 Median : 39.918   Median :1.1995   Median :1.1468   Median :122.9
1  
 Mean   : 58.540   Mean   :1.1656   Mean   :1.0639   Mean   :115.6
0  
 3rd Qu.: 73.113   3rd Qu.:1.4445   3rd Qu.:1.3594   3rd Qu.:154.1
2  
 Max.   :208.980   Max.   :1.9838   Max.   :1.9582   Max.   :207.9
7  
     wr_am            std_am           wstd_am      
 Min.   :  0.00   Min.   :  0.00   Min.   :  0.00  
 1st Qu.: 16.82   1st Qu.: 32.89   1st Qu.: 28.54  
 Median : 26.64   Median : 45.12   Median : 44.29  
 Mean   : 33.23   Mean   : 44.39   Mean   : 41.45  
 3rd Qu.: 38.36   3rd Qu.: 59.32   3rd Qu.: 53.63  
 Max.   :205.59   Max.   :101.02   Max.   :101.02  
```



In [22]:

```
table(super[1])
```

```
  1    2    3    4    5    6    7    8    9 
285 3280 3895 4496 5792 2666  774   61   14 
```

- From the summary of "number_of_elements" we can see that the minimum number of elements that a superconductor contains is 1 and the maximum number of elements that a superconductor contains is 9. On an average, a superconductor contains 4 elements in its chemical composition. And this statement can be backed up in the next point.
- We can see from the output of running the **table( )** function on this column that the count of the superconductors which have 5 elements are the highest. The next highest count belongs to super conductors with 4 number of elements. While superconductors with 2 or 3 elements have close numbers. Superconductors with 1 or 7 or 8 or 9 elements are highly variable, the least being superconductors with 9 elements with a count of just 14.

The variables extracted for the "Atomic Mass" of a superconductor are observed to have the following properties:

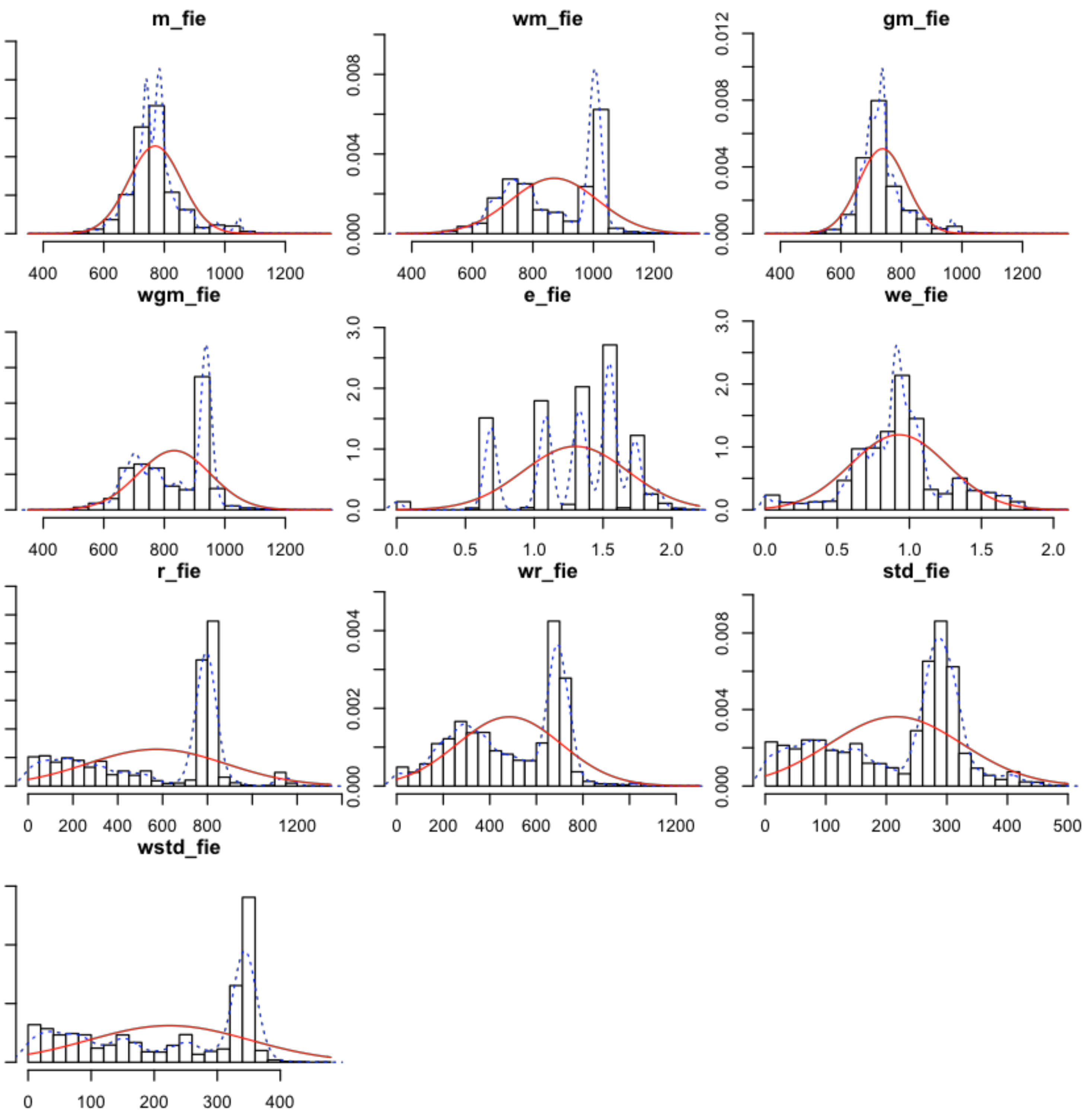
- The mean, weighted mean, geometric mean and weighted geometric mean of atomic masses have the same upper limit or max value of 208.980 AMU.
- While the minimum values of mean, weighted mean and geometric mean lie close to each other; the minimum value of the weighted geometric mean is far lesser than its counterparts.
- Since entropy is a logarithmic value, the range of that value is much lesser than the rest of the derivatives.
- The range variables are pretty much self explanatory as they are the difference between the atomic masses of the elements.
- Where as the standard deviation and weighted standard deviation of the atomic mass has a mean value between 41-45 AMU. Almost half as much as the mean of mean atomic mass of the superconductor.
- We can also see from the histograms that mean `atomic mass seems to have a somewhat normal distribution`. the `geometric mean seems to be skewed to the right`.
- The `entropy and range` based derived functions also seem to have a `very variable distribution`.
- The highly variant data could help the model perform better.

The next block of stats is **"First Ionization Energy"**:

In [23]:

```
summary(super1[12:21])
multi.hist(super1[12:21],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

```
     m_fie            wm_fie           gm_fie           wgm_fie      
 Min.   : 375.5   Min.   : 375.5   Min.   : 375.5   Min.   : 375.5  
 1st Qu.: 723.7   1st Qu.: 738.9   1st Qu.: 692.5   1st Qu.: 720.1  
 Median : 764.9   Median : 890.0   Median : 728.0   Median : 856.2  
 Mean   : 769.6   Mean   : 870.4   Mean   : 737.5   Mean   : 832.8  
 3rd Qu.: 796.3   3rd Qu.:1004.1   3rd Qu.: 765.7   3rd Qu.: 937.6  
 Max.   :1313.1   Max.   :1348.0   Max.   :1313.1   Max.   :1327.6  
     e_fie           we_fie            r_fie            wr_fie      
 Min.   :0.000   Min.   :0.0000   Min.   :   0.0   Min.   :   0.0  
 1st Qu.:1.086   1st Qu.:0.7538   1st Qu.: 262.4   1st Qu.: 291.1  
 Median :1.356   Median :0.9168   Median : 764.1   Median : 510.4  
 Mean   :1.299   Mean   :0.9267   Mean   : 572.2   Mean   : 483.5  
 3rd Qu.:1.551   3rd Qu.:1.0618   3rd Qu.: 810.6   3rd Qu.: 690.7  
 Max.   :2.158   Max.   :2.0386   Max.   :1304.5   Max.   :1251.9  
    std_fie          wstd_fie     
 Min.   :  0.0   Min.   :  0.00  
 1st Qu.:114.1   1st Qu.: 92.99  
 Median :266.4   Median :258.45  
 Mean   :215.6   Mean   :224.05  
 3rd Qu.:297.7   3rd Qu.:342.66  
 Max.   :499.7   Max.   :479.16  
```



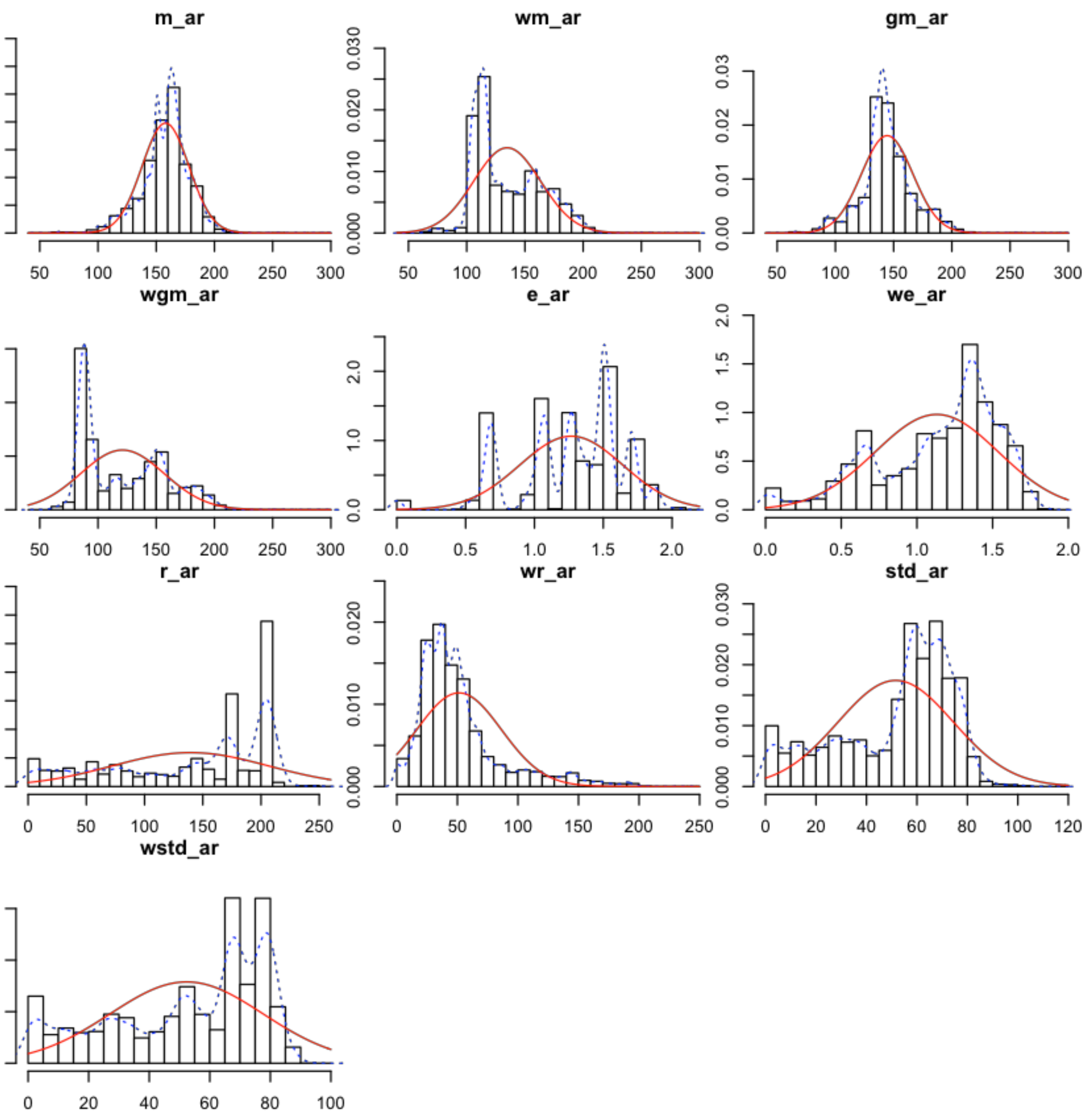
- Since FIE refers to the ionization energy which is measured in Kilo Joules per mole, the values in general, are high.
- While the mean, weighted mean, geometric mean, weighted geometric mean, range and weighted range all lie on a similar plane with values > 1300 or close to 1300 kJ/mole.
- But the mean of all these variables lies between 730 to 870 kJ/mole.
- Entropy variables, are again logarithmic and hence they have a small range of 0-2.1 kJ/mole.
- The mean standard deviation lies around 215 kJ/mole.
- While the mean FIE histogram does seem like a decent data distribution, it is striking to see the how the variability of the other extracted features is like.
- For example, the spread of `we_fie is possibly ideal`. Where as the distribution of `entropy_fie does not look promising at all.` Meaning that, there are several redundant values which might or might not be useful for us during model development.

The next block of stats is **"Atomic Radius"**:

In [24]:

```
summary(super1[22:31])
multi.hist(super1[22:31],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

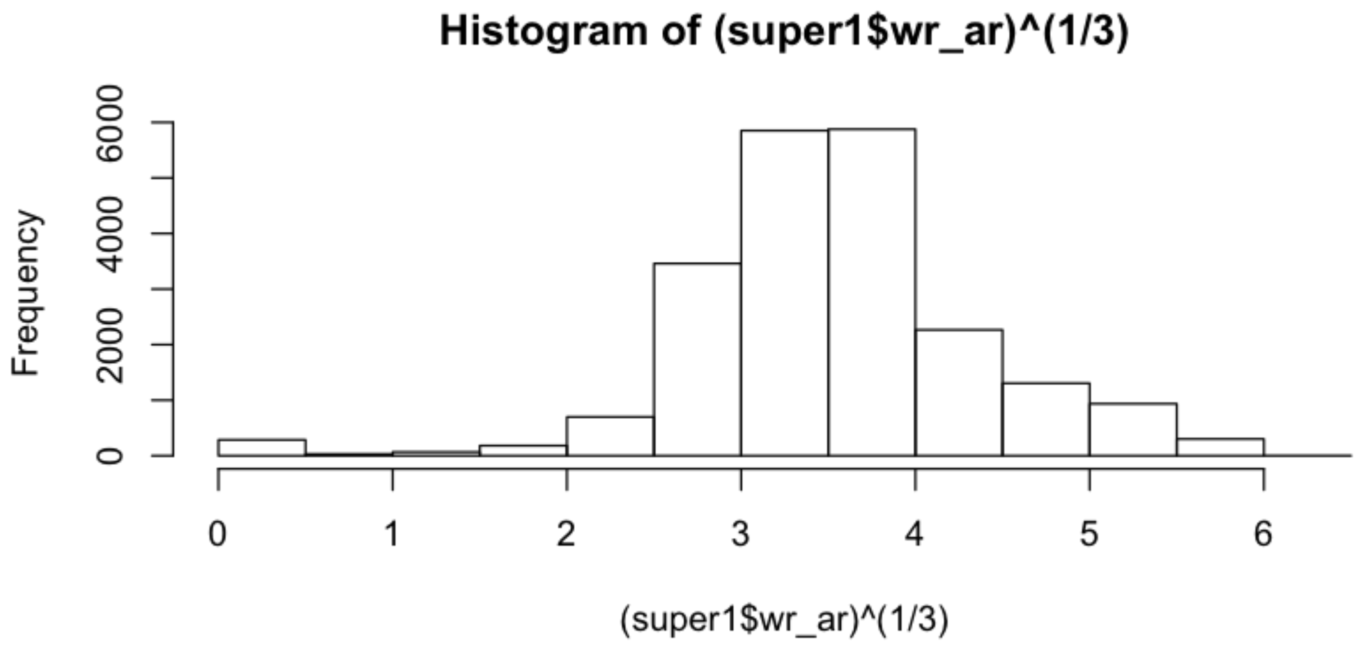
```
     m_ar              wm_ar              gm_ar              wgm_ar
 Min.   : 48.0    Min.   : 48.0    Min.   : 48.0    Min.   : 48.00
 1st Qu.:149.3    1st Qu.:112.1    1st Qu.:133.5    1st Qu.: 89.21
 Median :160.2    Median :126.0    Median :142.8    Median :113.18
 Mean   :158.0    Mean   :134.7    Mean   :144.4    Mean   :120.99
 3rd Qu.:169.9    3rd Qu.:158.3    3rd Qu.:155.9    3rd Qu.:150.99
 Max.   :298.0    Max.   :298.0    Max.   :298.0    Max.   :298.00
     e_ar              we_ar              r_ar               wr_ar
 Min.   :0.000    Min.   :0.0000    Min.   :  0.0    Min.   :  0.00
 1st Qu.:1.066    1st Qu.:0.8522    1st Qu.: 80.0    1st Qu.: 28.60
 Median :1.331    Median :1.2429    Median :171.0    Median : 43.00
 Mean   :1.268    Mean   :1.1311    Mean   :139.3    Mean   : 51.37
 3rd Qu.:1.512    3rd Qu.:1.4257    3rd Qu.:205.0    3rd Qu.: 60.22
 Max.   :2.142    Max.   :1.9037    Max.   :256.0    Max.   :240.16
     std_ar            wstd_ar
 Min.   :  0.00    Min.   : 0.00
 1st Qu.: 35.11    1st Qu.:32.02
 Median : 58.66    Median :59.93
 Mean   : 51.60    Mean   :52.34
 3rd Qu.: 69.42    3rd Qu.:73.78
 Max.   :115.50    Max.   :97.14
```

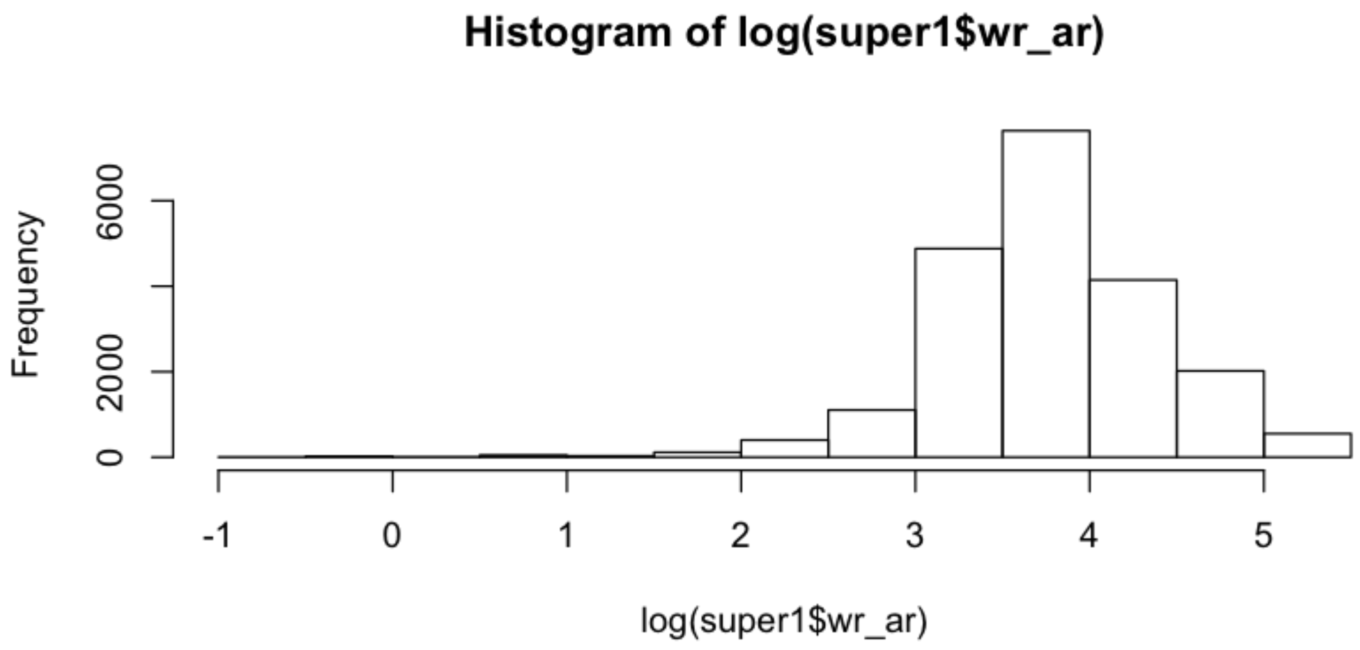


- The average atomic radius of the superconductors is around 160 picometers.
- The minimum and maximum of mean, weighted mean, geometric mean and weighted geometric mean atomic radii all lie between 48 - 298 picometers.
- The standard deviation the atomic radii has a mean of 50~ picometers.
- The distribution of the variables themselves looks extremely promising. The `mean atomic radius` has an `extremely normal` distribution.
- The weighted range atomic radius is extremely right skewed.

In [25]:

```
#Checking the distribution of weighted range atomic radius after cube root tra
nsformation
par(mfrow =c(2,1))
hist((super1$wr_ar)^(1/3))
hist(log(super1$wr_ar))
```





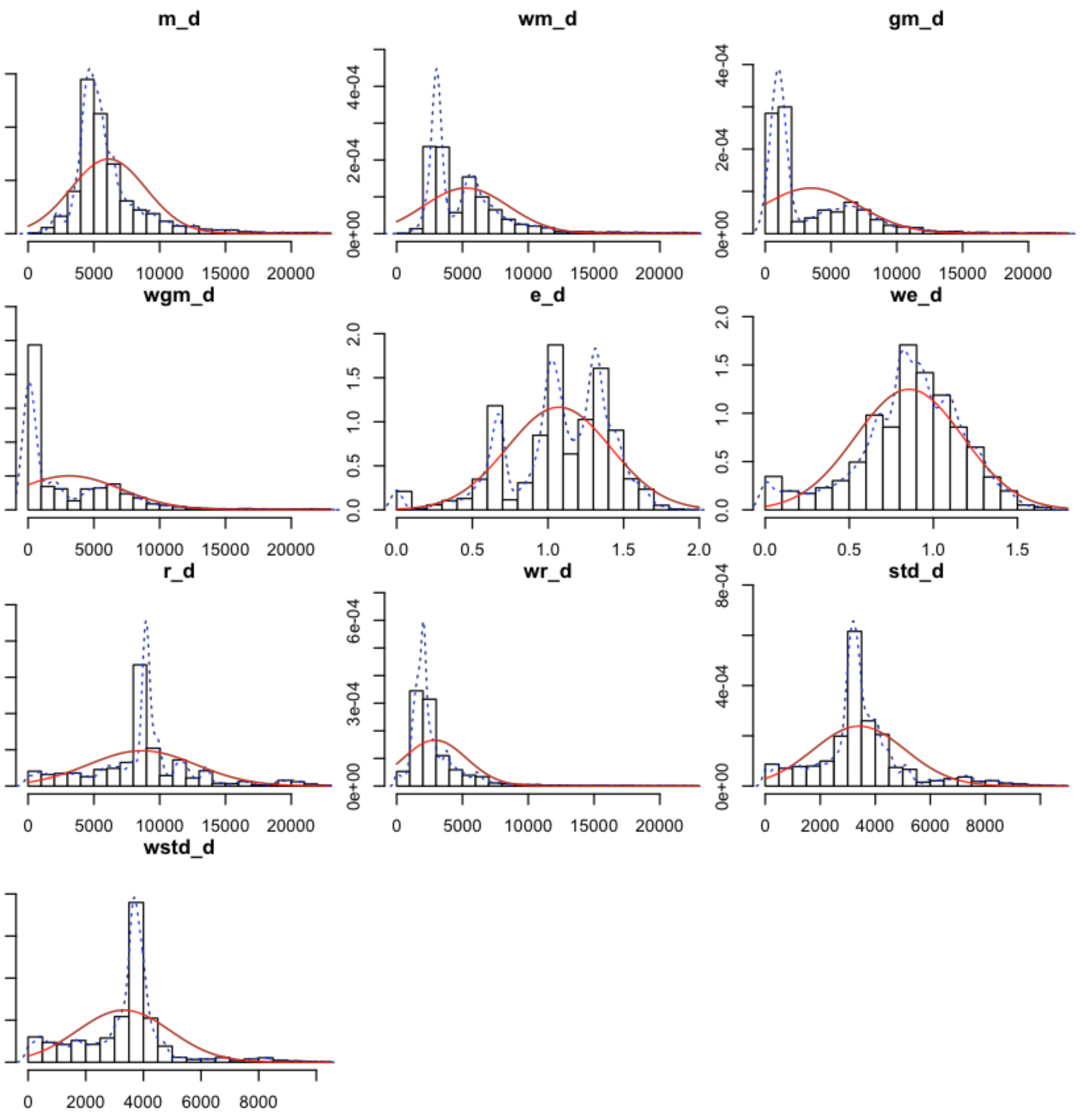
```
 It can be seen from above that, the log transformation of the variable isn't
as effective here as much as the cube root transformation.
```

The next block we will be looking into is **"Density"**:

```
In [26]:
```

```
summary(super1[32:41])
multi.hist(super1[32:41],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

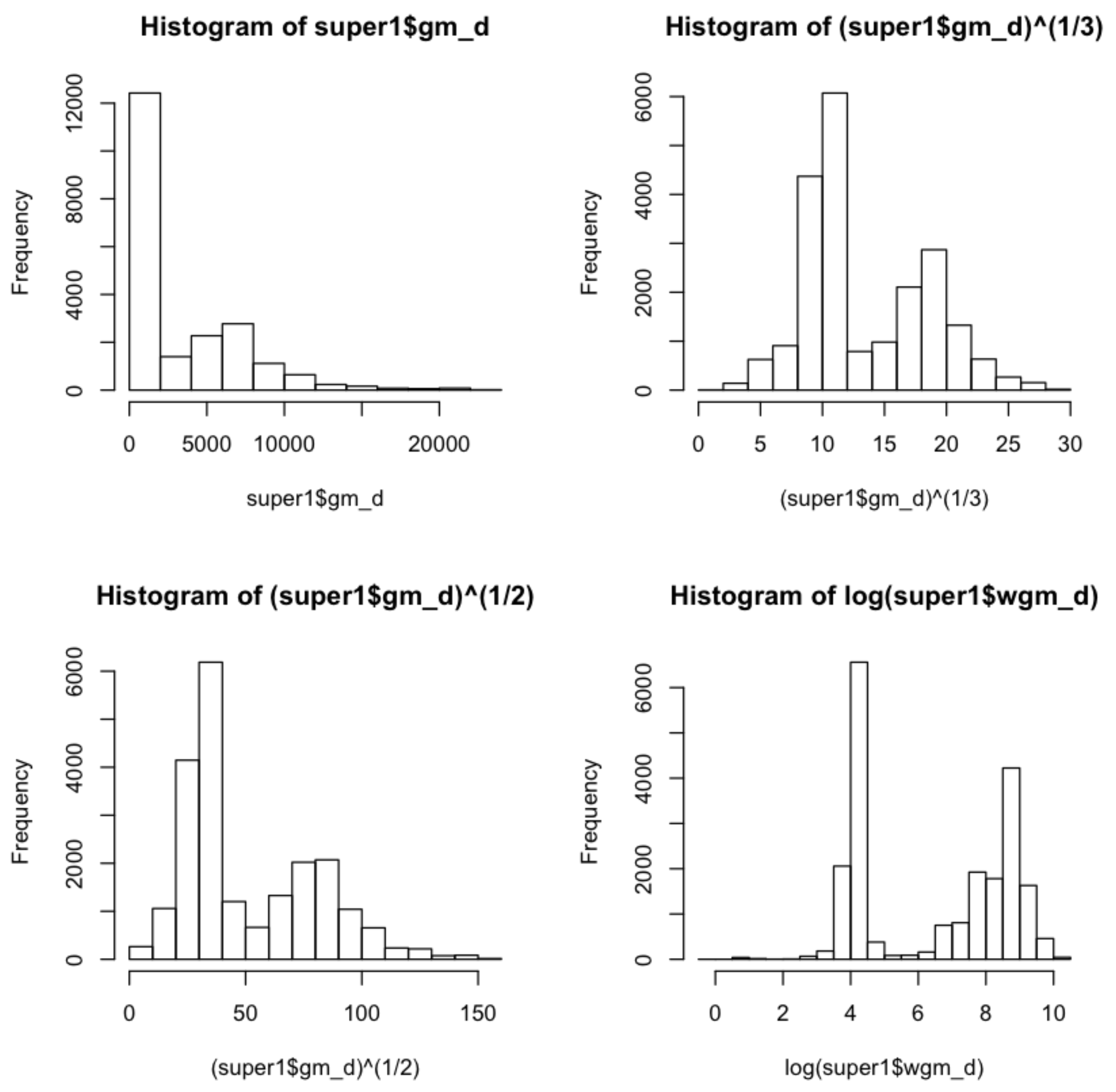
```
      m_d                 wm_d                 gm_d          
 Min.    :    1.429   Min.    :    1.429   Min.    :    1.429  
 1st Qu.: 4513.500   1st Qu.: 2999.158   1st Qu.:  883.117  
 Median : 5329.086   Median : 4303.422   Median : 1339.975  
 Mean   : 6111.465   Mean   : 5267.189   Mean   : 3460.692  
 3rd Qu.: 6728.000   3rd Qu.: 6416.333   3rd Qu.: 5794.965  
 Max.   :22590.000   Max.   :22590.000   Max.   :22590.000  
     wgm_d                e_d              we_d              r_d       
 Min.    :    0.686   Min.   :0.000   Min.   :0.0000   Min.   :    
0  
 1st Qu.:   66.747   1st Qu.:0.914   1st Qu.:0.6887   1st Qu.: 664
8  
 Median : 1515.365   Median :1.091   Median :0.8827   Median : 895
9  
 Mean   : 3117.241   Mean   :1.072   Mean   :0.8560   Mean   : 866
5  
 3rd Qu.: 5766.015   3rd Qu.:1.324   3rd Qu.:1.0809   3rd Qu.: 977
9  
 Max.   :22590.000   Max.   :1.954   Max.   :1.7034   Max.   :2258
9  
      wr_d            std_d            wstd_d     
 Min.   :    0   Min.   :    0   Min.   :    0  
 1st Qu.: 1657   1st Qu.: 2819   1st Qu.: 2564  
 Median : 2083   Median : 3302   Median : 3626  
 Mean   : 2903   Mean   : 3417   Mean   : 3319  
 3rd Qu.: 3409   3rd Qu.: 4004   3rd Qu.: 3959  
 Max.   :22434   Max.   :10724   Max.   :10411  
```



- Since density is measured per cubed meter, again, the values will be large.
- The mean density of a superconductor being around 6000~ kg/m3.
- The most dense element has a density of 22590 kg/m3.
- From the plots, it seems to appear that most of the features extracted for density seem to be normal except for geometric mean, which has a steep skew.
- We can try to apply some transformation to see if it help normalize it.

In [27]:

```
#Checking the distribution of weighted geometric mean density after transforma
tion
par(mfrow =c(2,2))
hist(super1$gm_d) #actual data
hist((super1$gm_d)^(1/3)) #cube root transformation
hist((super1$gm_d)^(1/2)) #square root transformation
hist(log(super1$wgm_d)) #logarithmic transformation
```



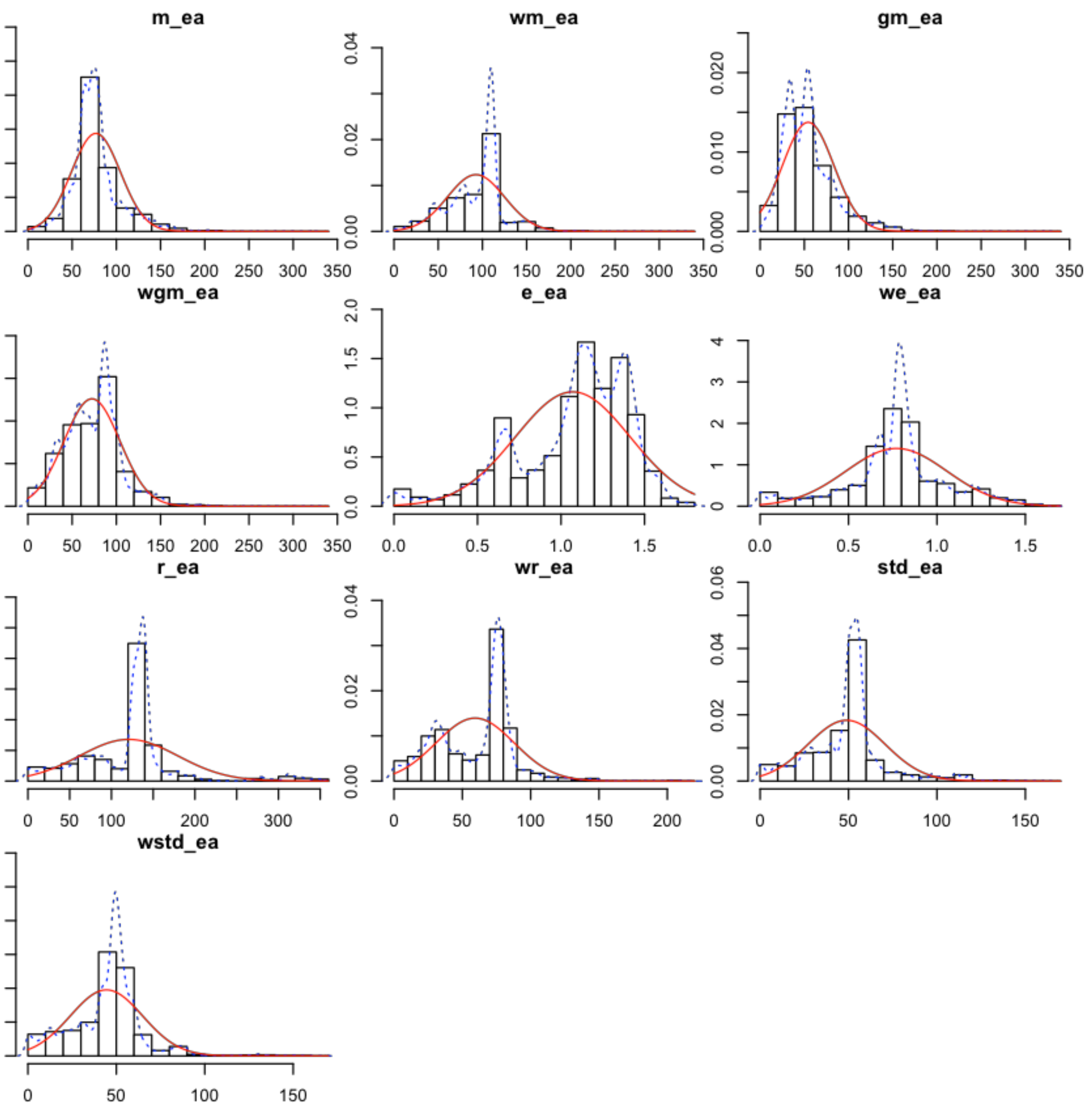
From the above histograms, we can see that the `transformations do not` seem to have any `useful outcome`.

The next block we will be looking into is **"Electron Affinity"**:

In [28]:

```
summary(super1[42:51])
multi.hist(super1[42:51],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

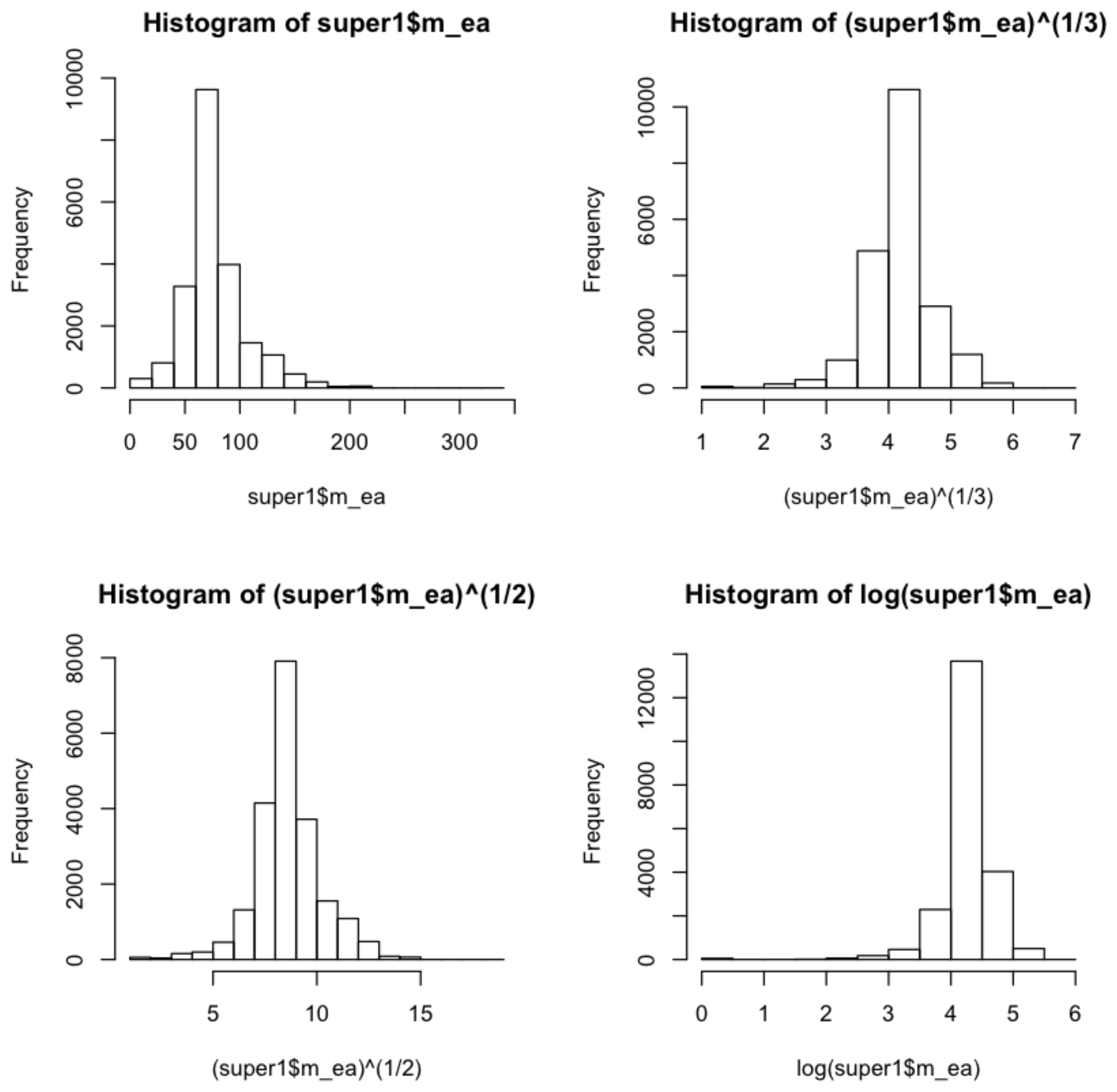
```
      m_ea             wm_ea              gm_ea             wgm_ea
 Min.   :  1.50   Min.   :  1.50   Min.   :  1.50   Min.   :  1.50
 1st Qu.: 62.09   1st Qu.: 73.35   1st Qu.: 33.70   1st Qu.: 50.77
 Median : 73.10   Median :102.86   Median : 51.47   Median : 73.17
 Mean   : 76.88   Mean   : 92.72   Mean   : 54.36   Mean   : 72.42
 3rd Qu.: 85.50   3rd Qu.:110.74   3rd Qu.: 67.51   3rd Qu.: 89.98
 Max.   :326.10   Max.   :326.10   Max.   :326.10   Max.   :326.10
      e_ea             we_ea              r_ea             wr_ea
 Min.   :0.0000   Min.   :0.0000   Min.   :  0.0   Min.   :  0.00
 1st Qu.:0.8906   1st Qu.:0.6607   1st Qu.: 86.7   1st Qu.: 34.04
 Median :1.1383   Median :0.7812   Median :127.0   Median : 71.16
 Mean   :1.0702   Mean   :0.7708   Mean   :120.7   Mean   : 59.33
 3rd Qu.:1.3459   3rd Qu.:0.8775   3rd Qu.:138.6   3rd Qu.: 76.71
 Max.   :1.7677   Max.   :1.6754   Max.   :349.0   Max.   :218.70
     std_ea            wstd_ea
 Min.   :  0.00   Min.   :  0.00
 1st Qu.: 38.37   1st Qu.: 33.44
 Median : 51.13   Median : 48.03
 Mean   : 48.91   Mean   : 44.41
 3rd Qu.: 56.22   3rd Qu.: 53.32
 Max.   :162.90   Max.   :169.08
```



- Electron affinity seemes to be lying between 1.50 to 326.10 kJ/mole.
- While the mean of mean, weighted mean, geometric mean and weighted geometric mean columns are highly variant.
- Most of the data is highly positively skewed. Thus, we can try to fix positive skew by applying root transformations on the data.
- Let us explore the possibilities below:

In [29]:

```
#Checking the distribution of weighted geometric mean density after transforma
tion
par(mfrow =c(2,2))
hist(super1$m_ea) #actual data
hist((super1$m_ea)^(1/3)) #cube root transformation
hist((super1$m_ea)^(1/2)) #square root transformation
hist(log(super1$m_ea)) #logarithmic transformation
```



At LAST, we were able to transform a variable, in this case using square root, and cause a significant impact on it. `<insert emoji>`

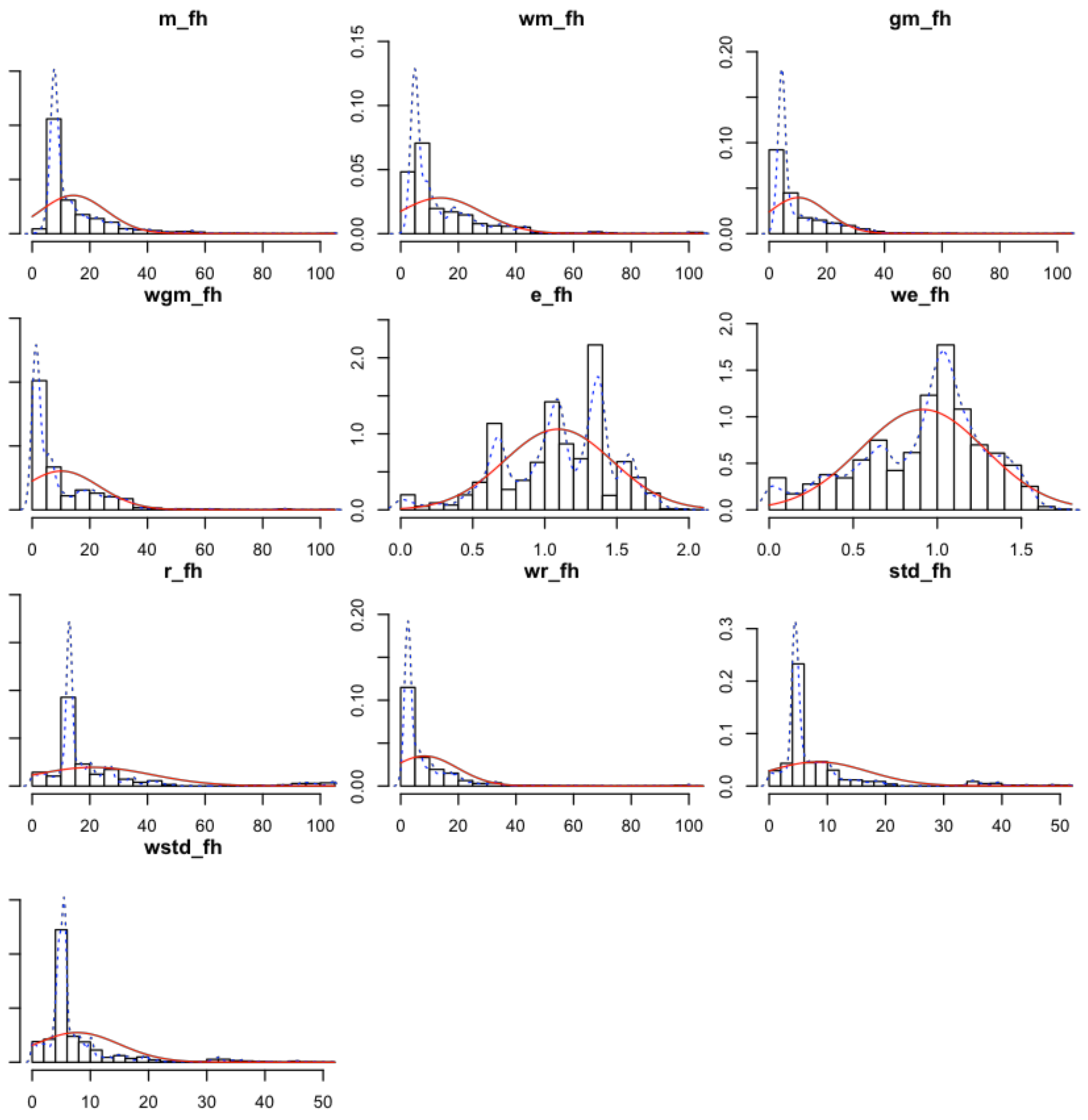
If this variable, 'm_ea' does happen to be an important predictor, we can try to check the accuracy of the model using the true value and compare it with using the transformed value.

The next block we will be looking into is **"Fusion Heat"**:

In [30]:

```
summary(super1[52:61])
multi.hist(super1[52:61],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

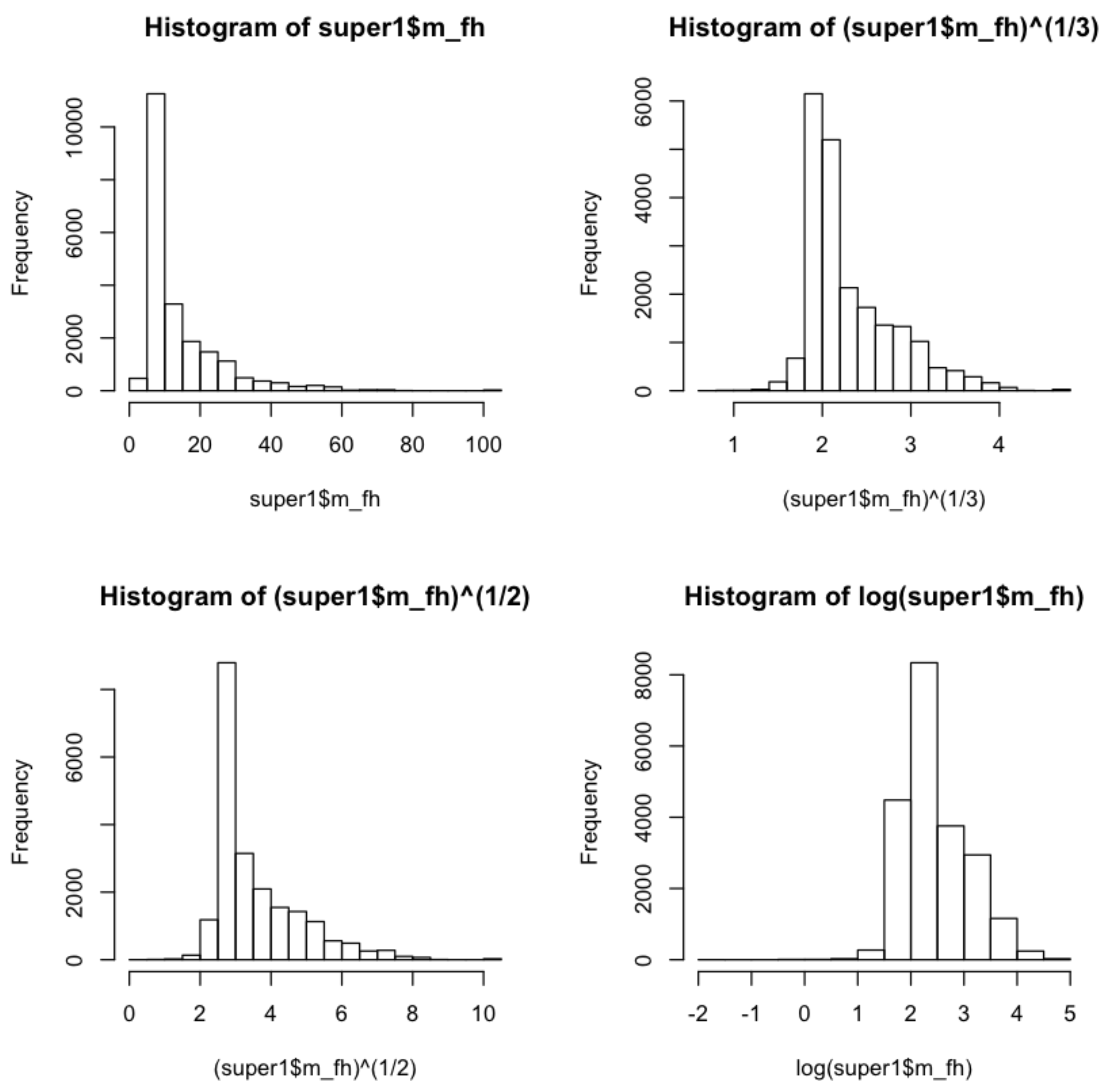
```
      m_fh              wm_fh              gm_fh              wgm_fh     
 Min.   :  0.222   Min.   :  0.222   Min.   :  0.222   Min.   :  0
.222  
 1st Qu.:  7.589   1st Qu.:  5.033   1st Qu.:  4.110   1st Qu.:  1
.322  
 Median :  9.304   Median :  8.331   Median :  5.253   Median :  4
.930  
 Mean   : 14.296   Mean   : 13.848   Mean   : 10.137   Mean   : 10
.141  
 3rd Qu.: 17.114   3rd Qu.: 18.514   3rd Qu.: 13.600   3rd Qu.: 16
.429  
 Max.   :105.000   Max.   :105.000   Max.   :105.000   Max.   :105
.000  
      e_fh              we_fh              r_fh              wr_fh       
 Min.   :0.0000   Min.   :0.0000   Min.   :  0.00   Min.   :  0.00
0  
 1st Qu.:0.8333   1st Qu.:0.6727   1st Qu.: 12.88   1st Qu.:  2.32
9  
 Median :1.1121   Median :0.9950   Median : 12.88   Median :  3.43
6  
 Mean   :1.0933   Mean   :0.9141   Mean   : 21.14   Mean   :  8.21
9  
 3rd Qu.:1.3781   3rd Qu.:1.1574   3rd Qu.: 23.20   3rd Qu.: 10.49
9  
 Max.   :2.0344   Max.   :1.7472   Max.   :104.78   Max.   :102.67
5  
     std_fh            wstd_fh      
 Min.   : 0.000   Min.   : 0.000  
 1st Qu.: 4.261   1st Qu.: 4.603  
 Median : 4.948   Median : 5.501  
 Mean   : 8.323   Mean   : 7.718  
 3rd Qu.: 9.041   3rd Qu.: 8.018  
 Max.   :51.635   Max.   :51.680  
```



- The fusion heat values also do not seem to be large.
- They have a very low mean lurking between 10-15 kJ/mole.
- While the maximum range is around 104 kJ/mole.
- The standard deviation and weighted standard deviation also seem to be extremely identical in their values.
- From the plots, its very evident that most of the fusion heat variables also seem to be extremely skewed to the right.
- We can check if we are able to eliminate this by using transformation.

In [31]:

```
#Checking the distribution of mean fusion heat after transformation
par(mfrow =c(2,2))
hist(super1$m_fh) #actual data
hist((super1$m_fh)^(1/3)) #cube root transformation
hist((super1$m_fh)^(1/2)) #square root transformation
hist(log(super1$m_fh)) #logarithmic transformation
```



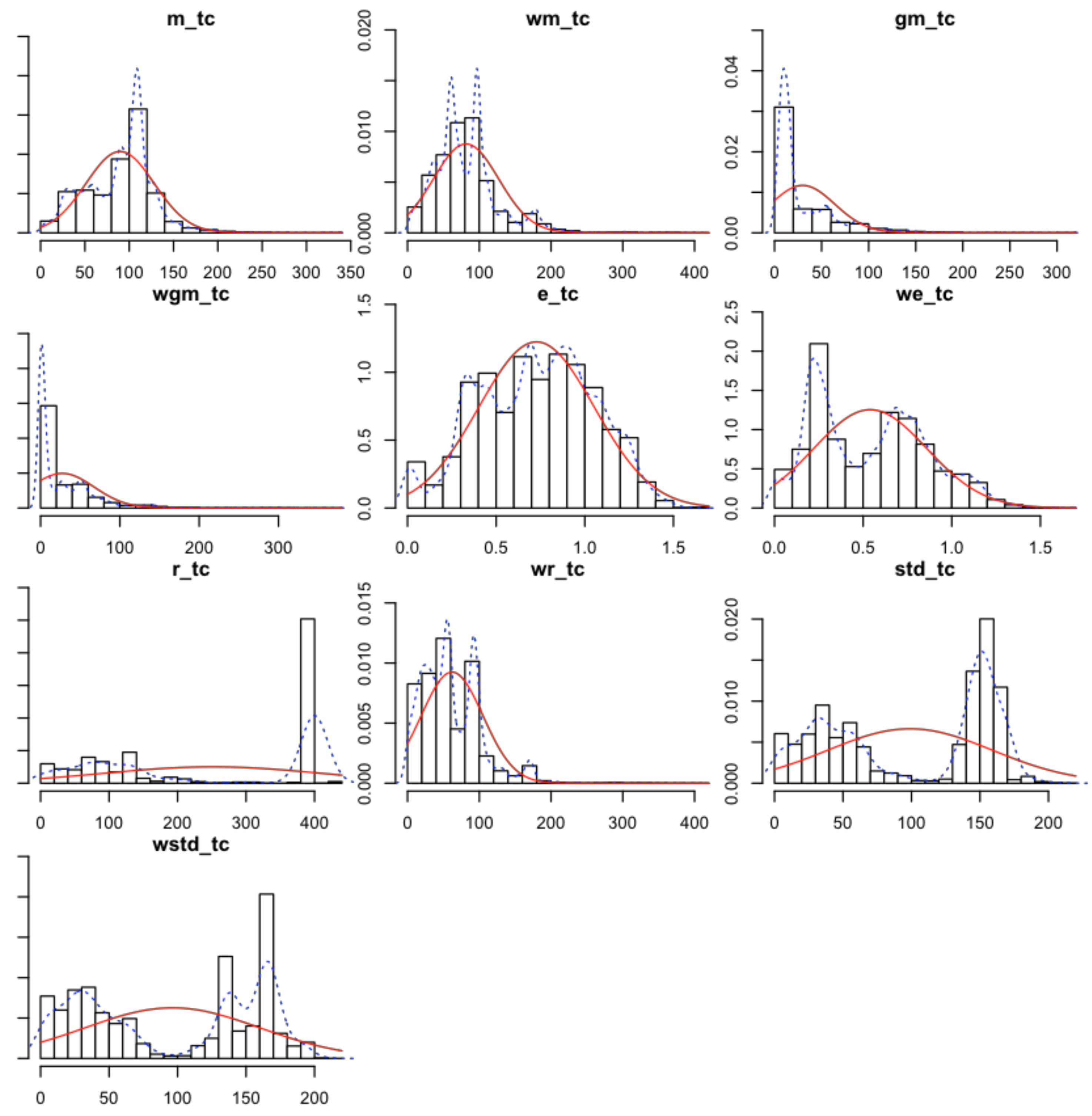
Out of all the transformations applied, the log transformation seems to be of a *close to normal* fit than any other. Thus, if this variable is also considered to be of significant importance from later exploration, we can apply transformations as necessary.

Next block we would be looking into is **"Thermal Conductivity"**:

In [32]:

```
summary(super1[62:71])
multi.hist(super1[62:71],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

```
      m_tc               wm_tc              gm_tc              wgm_
tc
 Min.   :  0.0266   Min.   :  0.0266   Min.   :  0.0266   Min.   :  
0.023
 1st Qu.: 61.0000   1st Qu.: 54.1810   1st Qu.:  8.3398   1st Qu.:  
1.087
 Median : 96.5044   Median : 73.3333   Median : 14.2876   Median :  
6.096
 Mean   : 89.7069   Mean   : 81.5491   Mean   : 29.8417   Mean   : 
27.308
 3rd Qu.:111.0053   3rd Qu.: 99.0629   3rd Qu.: 42.3713   3rd Qu.: 
47.308
 Max.   :332.5000   Max.   :406.9600   Max.   :317.8836   Max.   :
376.033
      e_tc              we_tc              r_tc              wr_tc       
 Min.   :0.0000   Min.   :0.0000   Min.   :  0.00   Min.   :  0.00  
 1st Qu.:0.4578   1st Qu.:0.2507   1st Qu.: 86.38   1st Qu.: 29.35  
 Median :0.7387   Median :0.5458   Median :399.80   Median : 56.56  
 Mean   :0.7276   Mean   :0.5400   Mean   :250.89   Mean   : 62.03  
 3rd Qu.:0.9622   3rd Qu.:0.7774   3rd Qu.:399.97   3rd Qu.: 91.87  
 Max.   :1.6340   Max.   :1.6130   Max.   :429.97   Max.   :401.44  
     std_tc            wstd_tc      
 Min.   :  0.00   Min.   :  0.00  
 1st Qu.: 37.93   1st Qu.: 31.99  
 Median :135.76   Median :113.56  
 Mean   : 98.94   Mean   : 96.23  
 3rd Qu.:153.81   3rd Qu.:162.71  
 Max.   :214.99   Max.   :213.30  
```



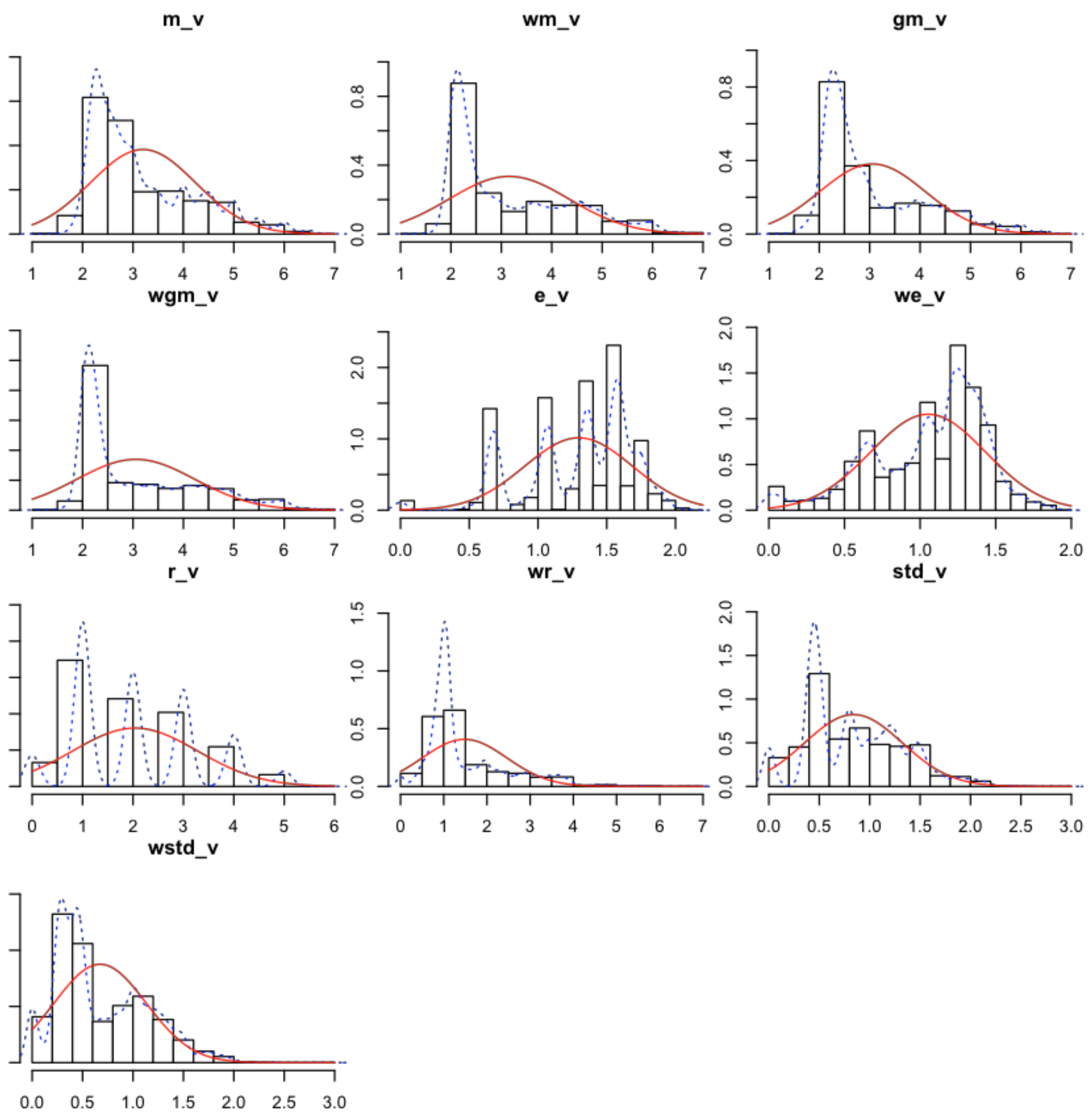
- Thermal conductivity also has low values with 75% of its mean values lying below 111 watts per meter-K.
- The lowest value being 0.0266 watts per meter-K and the high values of 332.50 and 406.96 watts per meter-K for the mean and weighted mean thermal conductivities respectively.
- The distributions of thermal conductivity based features are extremely varied.
- For example, the **std_tc** and **wstd_tc** variables seem to be **bimodal** in nature.
- Each of these individual features needs to be checked/transformed.
- We will not be going into it now as it can get extremely complex. But we can always come back to this for insights after estimating their relevance to the target variable.

And the final block we would be looking into is **"Valence"**:

In [33]:

```
summary(super1[72:81])
multi.hist(super1[72:81],dcol= c("blue","red"),dlty=c("dotted", "solid"))
```

```
      m_v               wm_v              gm_v              wgm_v
 Min.   :1.000    Min.   :1.000    Min.   :1.000    Min.   :1.000
 1st Qu.:2.333    1st Qu.:2.117    1st Qu.:2.280    1st Qu.:2.091
 Median :2.833    Median :2.618    Median :2.615    Median :2.434
 Mean   :3.198    Mean   :3.153    Mean   :3.057    Mean   :3.056
 3rd Qu.:4.000    3rd Qu.:4.026    3rd Qu.:3.728    3rd Qu.:3.915
 Max.   :7.000    Max.   :7.000    Max.   :7.000    Max.   :7.000
      e_v               we_v              r_v               wr_v
 Min.   :0.000    Min.   :0.0000   Min.   :0.000    Min.   :0.0000
 1st Qu.:1.061    1st Qu.:0.7757   1st Qu.:1.000    1st Qu.:0.9215
 Median :1.369    Median :1.1665   Median :2.000    Median :1.0631
 Mean   :1.296    Mean   :1.0528   Mean   :2.041    Mean   :1.4830
 3rd Qu.:1.589    3rd Qu.:1.3308   3rd Qu.:3.000    3rd Qu.:1.9184
 Max.   :2.142    Max.   :1.9497   Max.   :6.000    Max.   :6.9922
     std_v              wstd_v
 Min.   :0.0000    Min.   :0.0000
 1st Qu.:0.4518    1st Qu.:0.3069
 Median :0.8000    Median :0.5000
 Mean   :0.8393    Mean   :0.6740
 3rd Qu.:1.2000    3rd Qu.:1.0204
 Max.   :3.0000    Max.   :3.0000
```



- Valence is just the number of chemical bonds made by the element.
- Although the values appear to be continuous, they could be discrete, lying within a range. As an element cannot form bonds in decimal values.
- The minimum and maximum number of bonds formed by a superconductor are 1 and 7 respectively.

Unique Materials:

In [34]:

```
unique <- read.csv('unique_m.csv')
```

In [35]:

```
head(unique)
```

H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si	P	S	Cl	Ar	K	Ca	Sc	Ti	V	Cr
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	0	0	4	0	0	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	0	0	4	0	0	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	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In [36]:

```
#Converting the dataset into a better format for analysis and visualisation.
un <- unique
un[ un > 0 ] <- 1
un$critic_temp <- unique$critical_temp
un$critical_temp <- NULL
```

```
Warning message in Ops.factor(left, right):
“‘>’ not meaningful for factors”
```

In [37]:

```
#Converting the values into factor format
un[1:86] <- lapply(un[1:86], factor)
un1 <- data.frame(apply(un[1:86], 2, table))
```

In [38]:

```
#Renaming the columns
un2<- un1[,grepl("*Freq",names(un1))]
un2 <- un2 %>% rename_at(vars(contains(".Freq")), funs(str_replace(., ".Freq",
""))) 
un2 <- un2[c(-1),]
rownames(un2) <- NULL
un2
```

H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si	P	S
299	21263	311	96	1205	1274	306	11964	669	21263	322	522	731	725	355	694

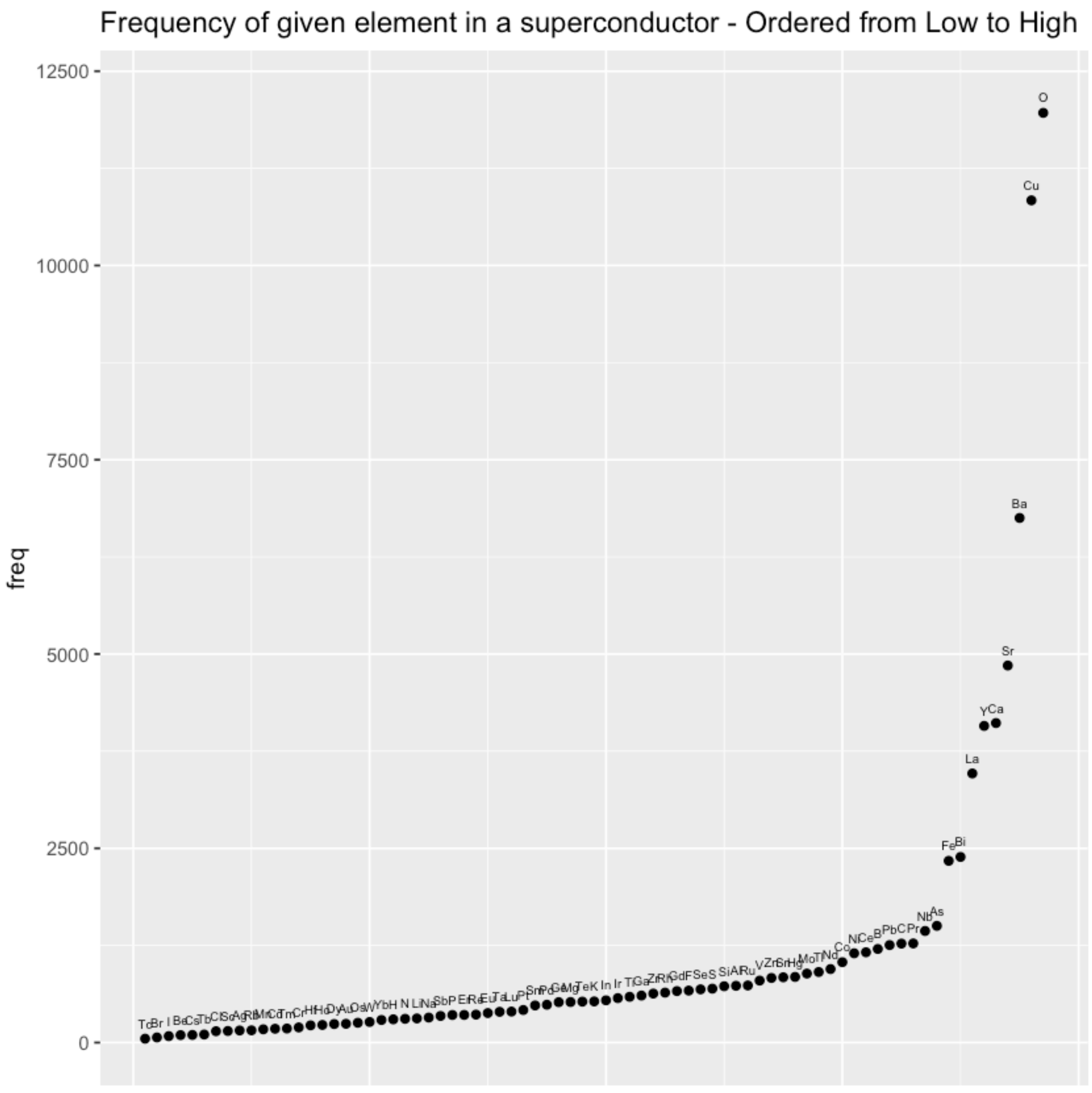
In [39]:

```
un3 <- data.frame(t(un2))
un4 <- tibble::rownames_to_column(un3, "VALUE")
colnames(un4)[1] <- "element"
colnames(un4)[2] <- "freq"
un4 <- un4[!grepl(21263, un4$freq),]
rownames(un4) <- NULL
#Ordering the dataframe by frequency of the material (ascending)
un5 <- un4[order(un4$freq),]
rownames(un5) <- NULL
un6<-un5
un6<- cbind(1:77,un5)
names(un6)[names(un6) == "1:77"] <- "sno"
```

```
Warning message:
“`list_len()` is deprecated as of rlang 0.2.0.
Please use `new_list()` instead.
This warning is displayed once per session.”
```

In [40]:

```
#Finally, plottting the dataframe
ggplot(un6, aes(sno,freq)) + geom_point() + geom_text(aes(label=element), posi
tion = position_nudge(y = 200),size=2) + theme(axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) + ggtitle("Frequency of given element in
a superconductor - Ordered from Low to High")
```



- We can see that Oxygen containing superconductors are the largest in number.
- Followed by Copper and Barium.

Scatter Plots between each Predictor and Target:

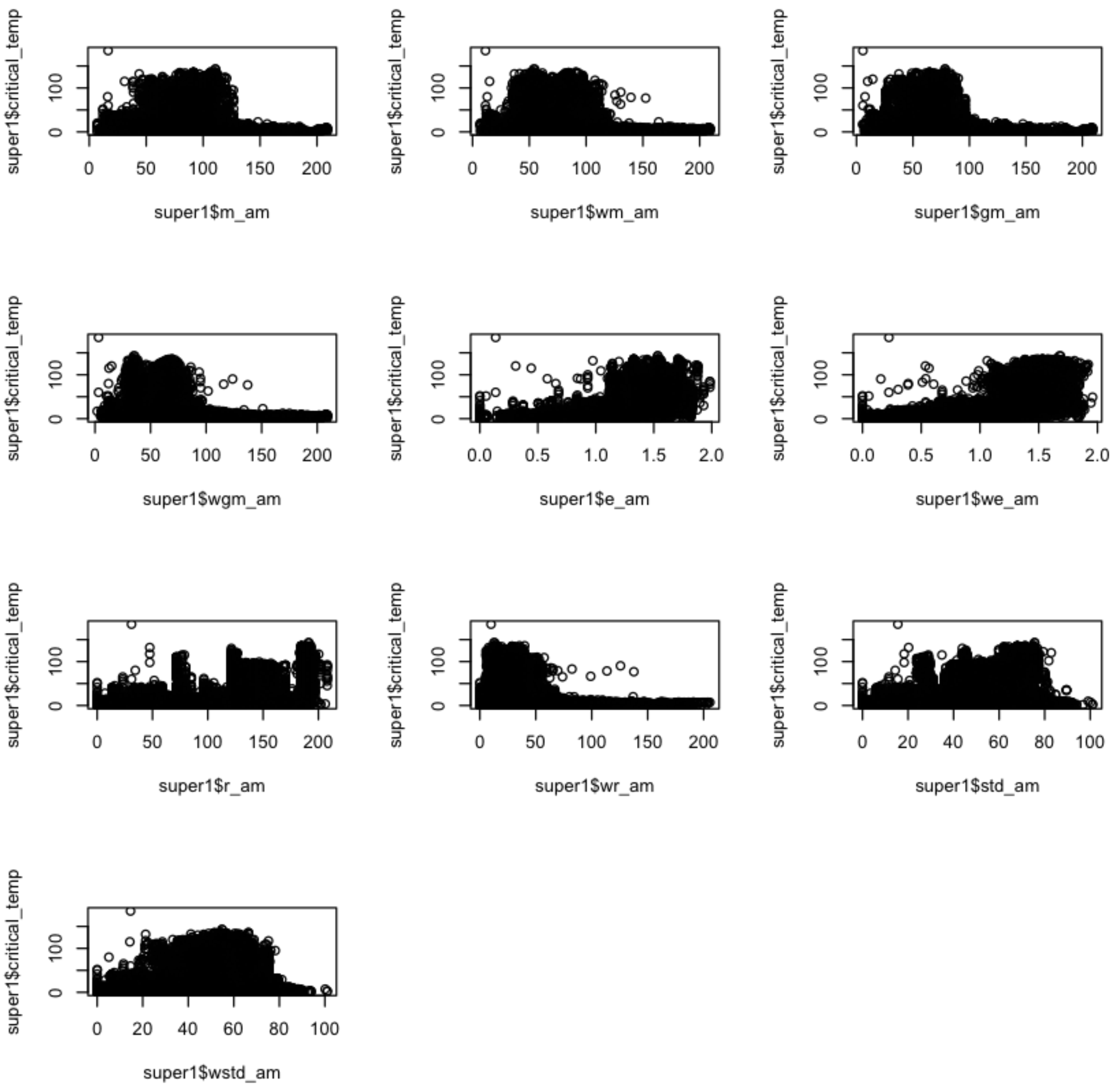
Scatter plots are a very good way to understand relationship between two variables. The intensity, strength, direction and disctribution of these plots gives a good evidence is drawing conclusions which could be useful for the model development stage.

We will be plotting all the variables with the target. Albeit this is tedious with such a large number of variables, it a good practice to do.

1. Atomic Mass vs Critical Temp Plots

In [42]:

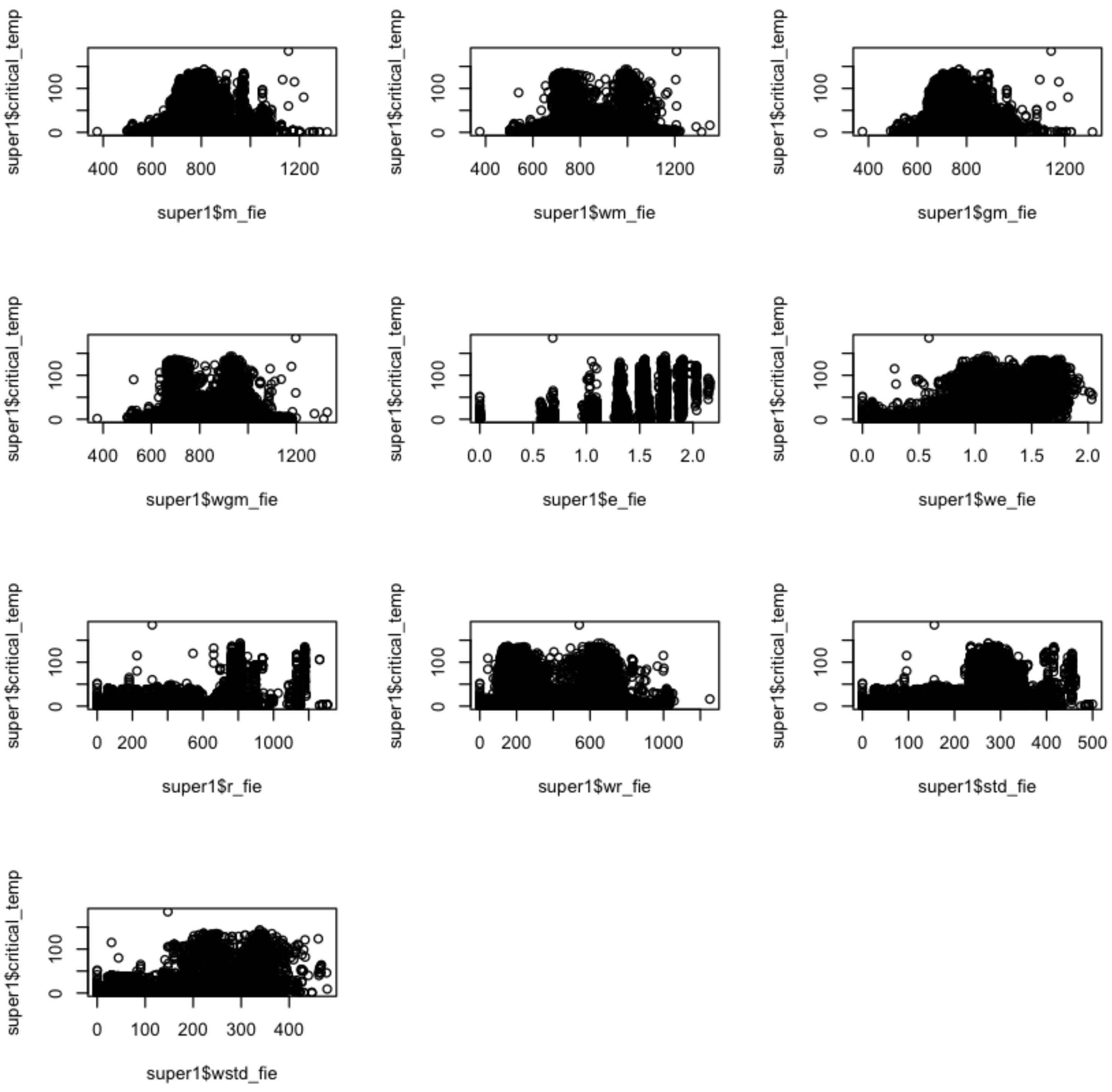
```
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_am, super1$critical_temp)
plot(super1$wm_am, super1$critical_temp)
plot(super1$gm_am, super1$critical_temp)
plot(super1$wgm_am, super1$critical_temp)
plot(super1$e_am, super1$critical_temp)
plot(super1$we_am, super1$critical_temp)
plot(super1$r_am, super1$critical_temp)
plot(super1$wr_am, super1$critical_temp)
plot(super1$std_am, super1$critical_temp)
plot(super1$wstd_am, super1$critical_temp)
```



First Ionization Energy vs Critical Temp plots

In [43]:

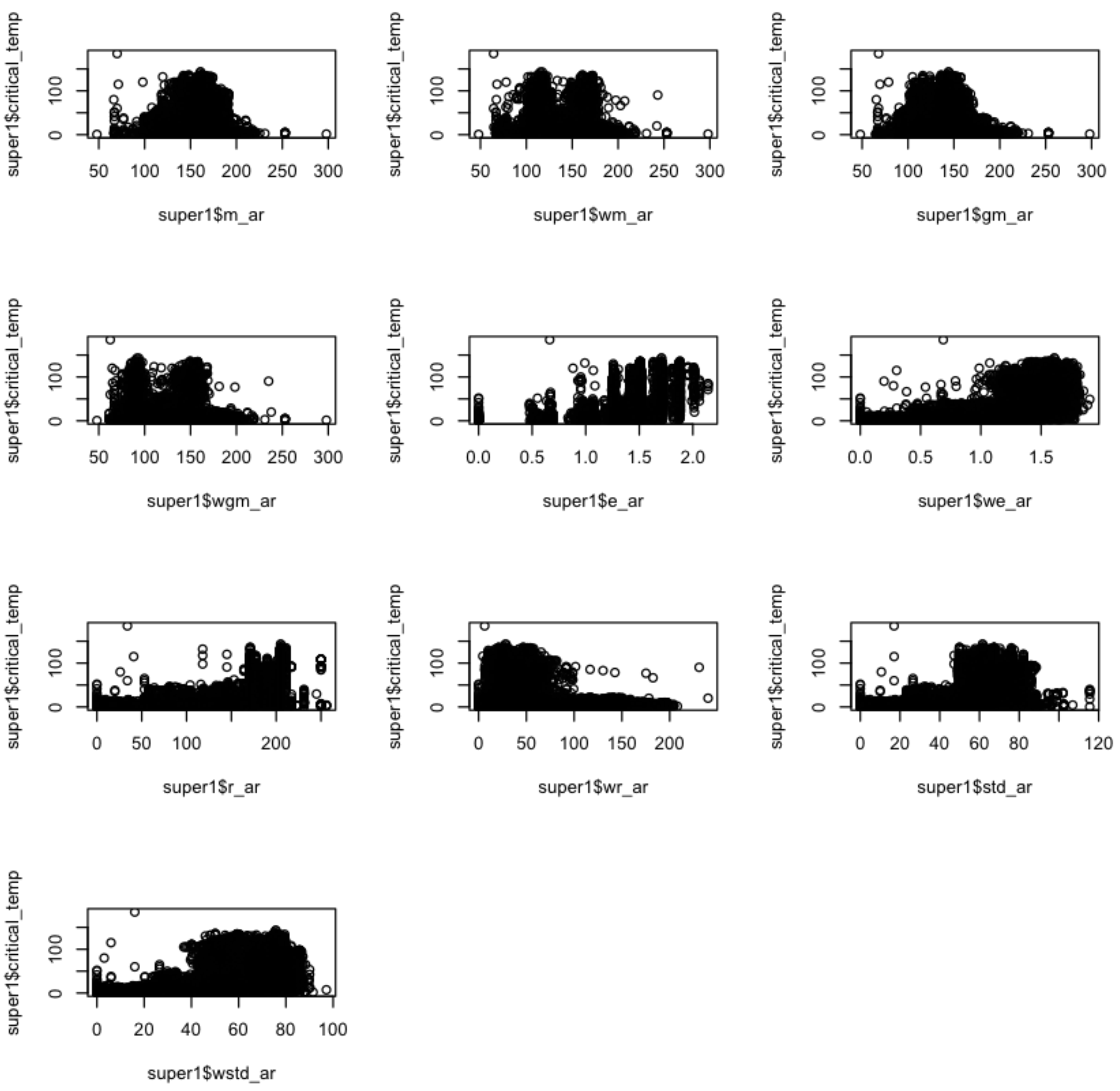
```
#Scatter plots of First Ionization Energy based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_fie, super1$critical_temp)
plot(super1$wm_fie, super1$critical_temp)
plot(super1$gm_fie, super1$critical_temp)
plot(super1$wgm_fie, super1$critical_temp)
plot(super1$e_fie, super1$critical_temp)
plot(super1$we_fie, super1$critical_temp)
plot(super1$r_fie, super1$critical_temp)
plot(super1$wr_fie, super1$critical_temp)
plot(super1$std_fie, super1$critical_temp)
plot(super1$wstd_fie, super1$critical_temp)
```



Atomic Radius vs Critical Temp Plots:

In [44]:

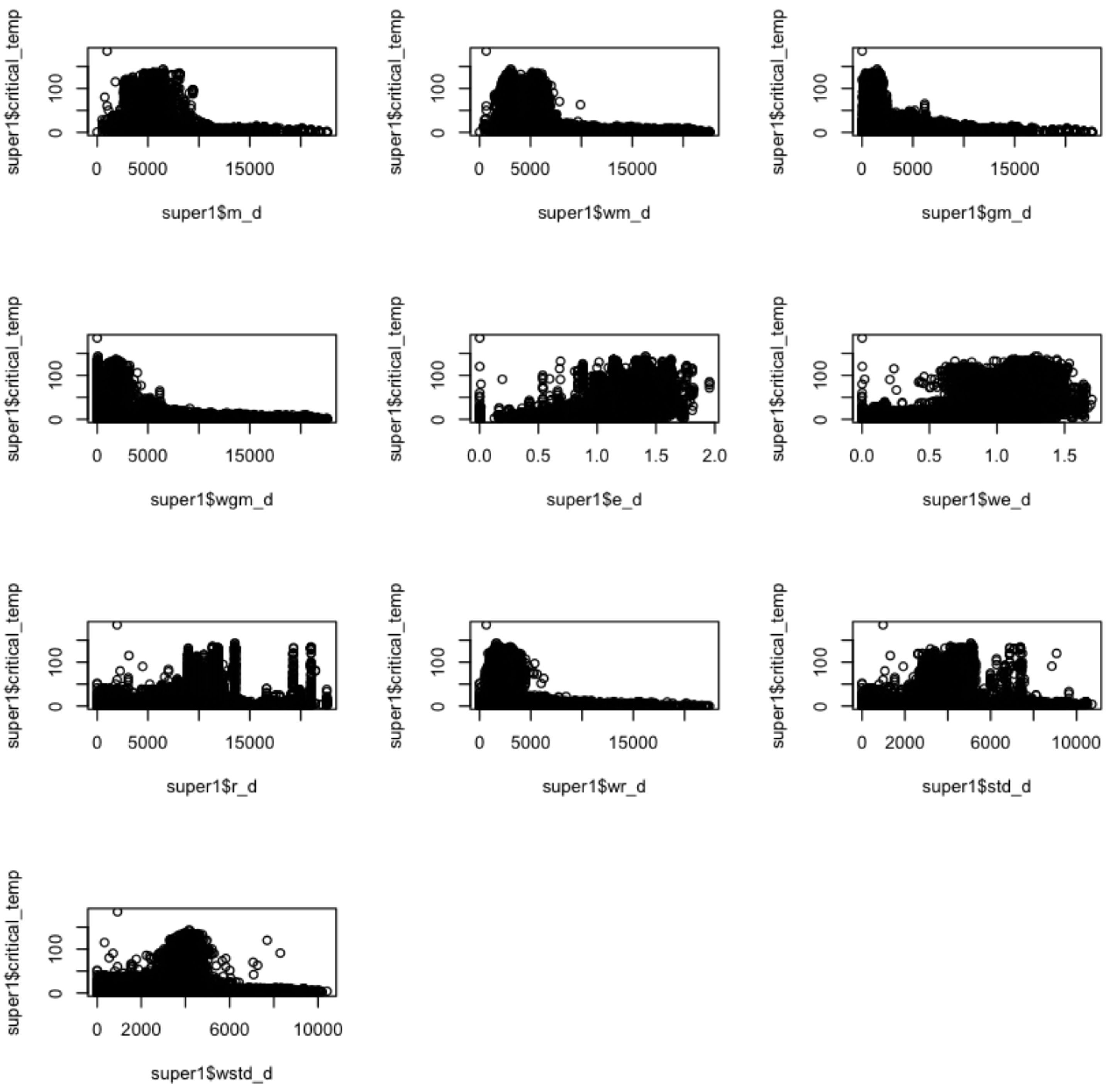
```
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_ar, super1$critical_temp)
plot(super1$wm_ar, super1$critical_temp)
plot(super1$gm_ar, super1$critical_temp)
plot(super1$wgm_ar, super1$critical_temp)
plot(super1$e_ar, super1$critical_temp)
plot(super1$we_ar, super1$critical_temp)
plot(super1$r_ar, super1$critical_temp)
plot(super1$wr_ar, super1$critical_temp)
plot(super1$std_ar, super1$critical_temp)
plot(super1$wstd_ar, super1$critical_temp)
```



Density vs Critical Temp Plots:

In [45]:

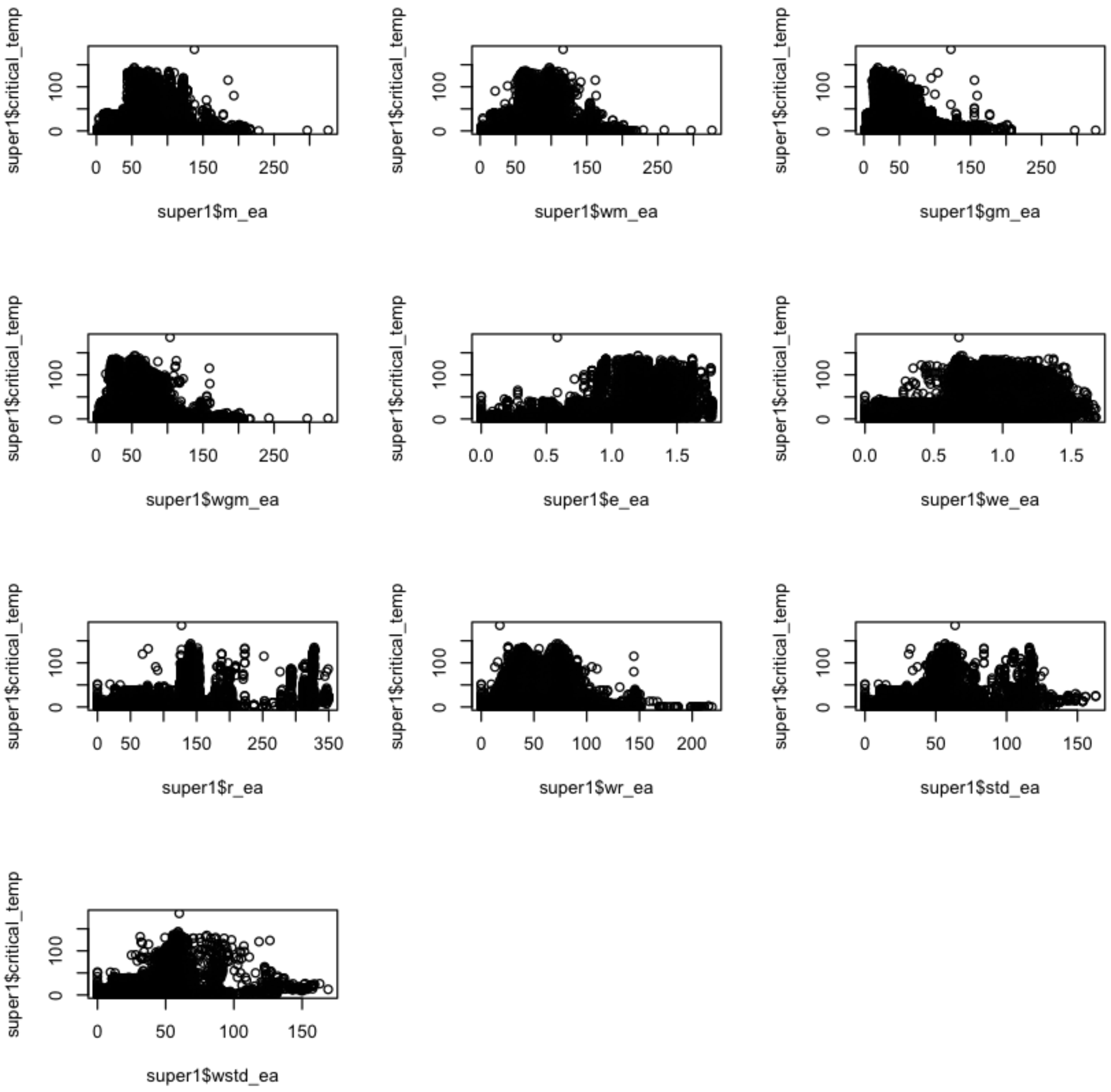
```r
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_d, super1$critical_temp)
plot(super1$wm_d, super1$critical_temp)
plot(super1$gm_d, super1$critical_temp)
plot(super1$wgm_d, super1$critical_temp)
plot(super1$e_d, super1$critical_temp)
plot(super1$we_d, super1$critical_temp)
plot(super1$r_d, super1$critical_temp)
plot(super1$wr_d, super1$critical_temp)
plot(super1$std_d, super1$critical_temp)
plot(super1$wstd_d, super1$critical_temp)
```



Electron Affinity vs Critical Temp Plots:

In [46]:

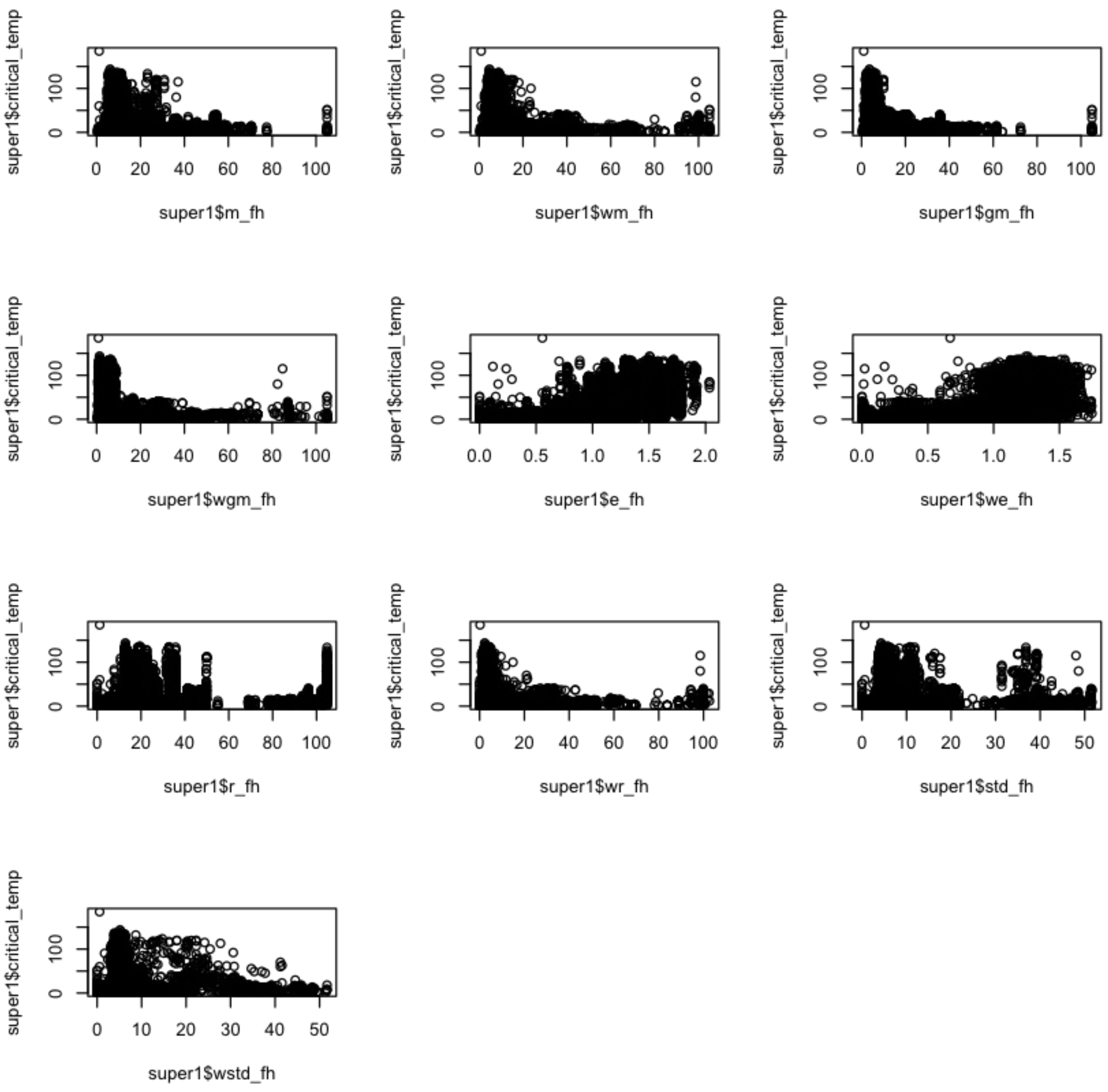
```r
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_ea, super1$critical_temp)
plot(super1$wm_ea, super1$critical_temp)
plot(super1$gm_ea, super1$critical_temp)
plot(super1$wgm_ea, super1$critical_temp)
plot(super1$e_ea, super1$critical_temp)
plot(super1$we_ea, super1$critical_temp)
plot(super1$r_ea, super1$critical_temp)
plot(super1$wr_ea, super1$critical_temp)
plot(super1$std_ea, super1$critical_temp)
plot(super1$wstd_ea, super1$critical_temp)
```



Fusion Heat vs Critical Temp Plots:

In [47]:

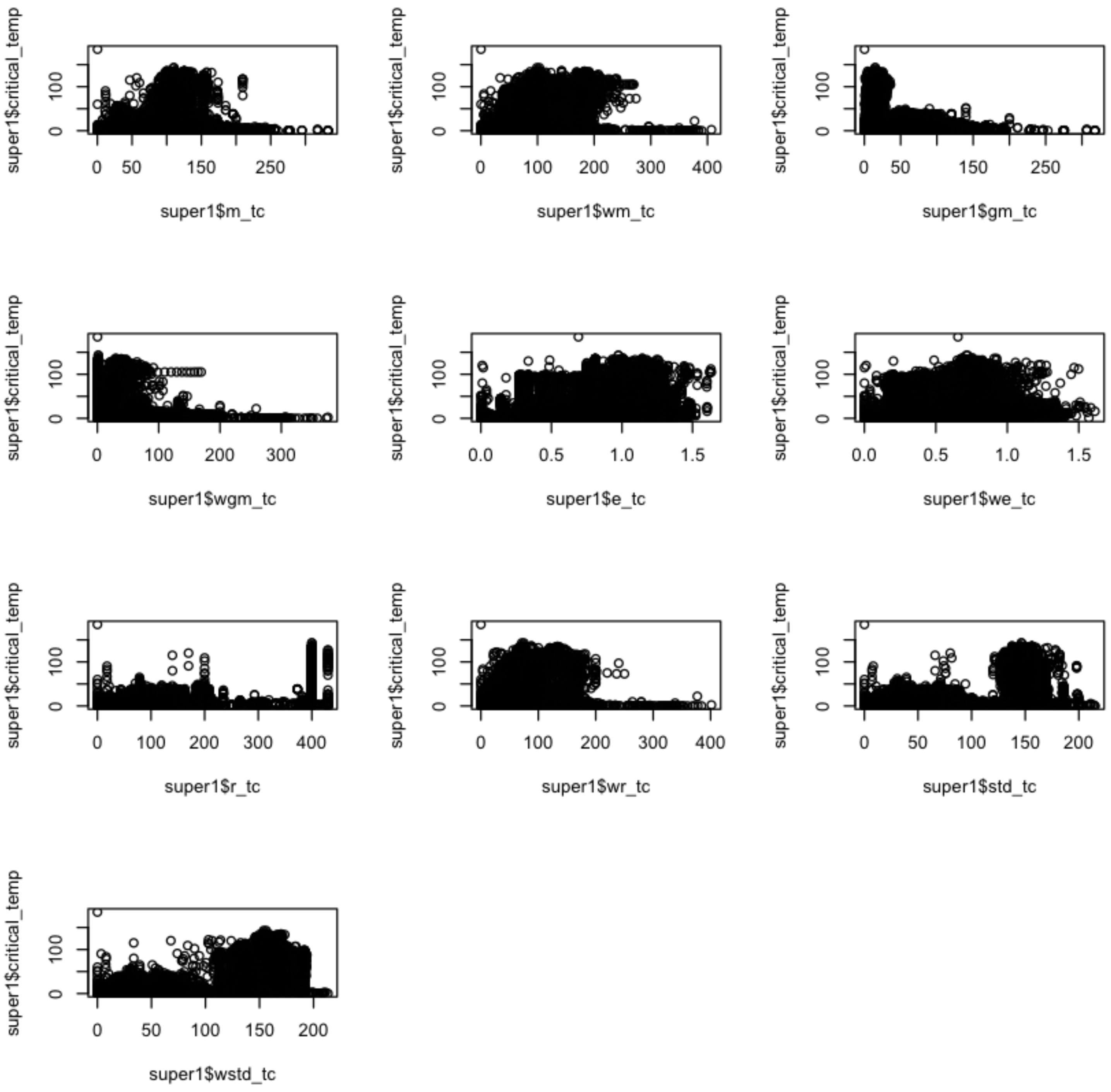
```
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_fh, super1$critical_temp)
plot(super1$wm_fh, super1$critical_temp)
plot(super1$gm_fh, super1$critical_temp)
plot(super1$wgm_fh, super1$critical_temp)
plot(super1$e_fh, super1$critical_temp)
plot(super1$we_fh, super1$critical_temp)
plot(super1$r_fh, super1$critical_temp)
plot(super1$wr_fh, super1$critical_temp)
plot(super1$std_fh, super1$critical_temp)
plot(super1$wstd_fh, super1$critical_temp)
```



Thermal Conductivity vs Critical Temp Plots:

In [48]:

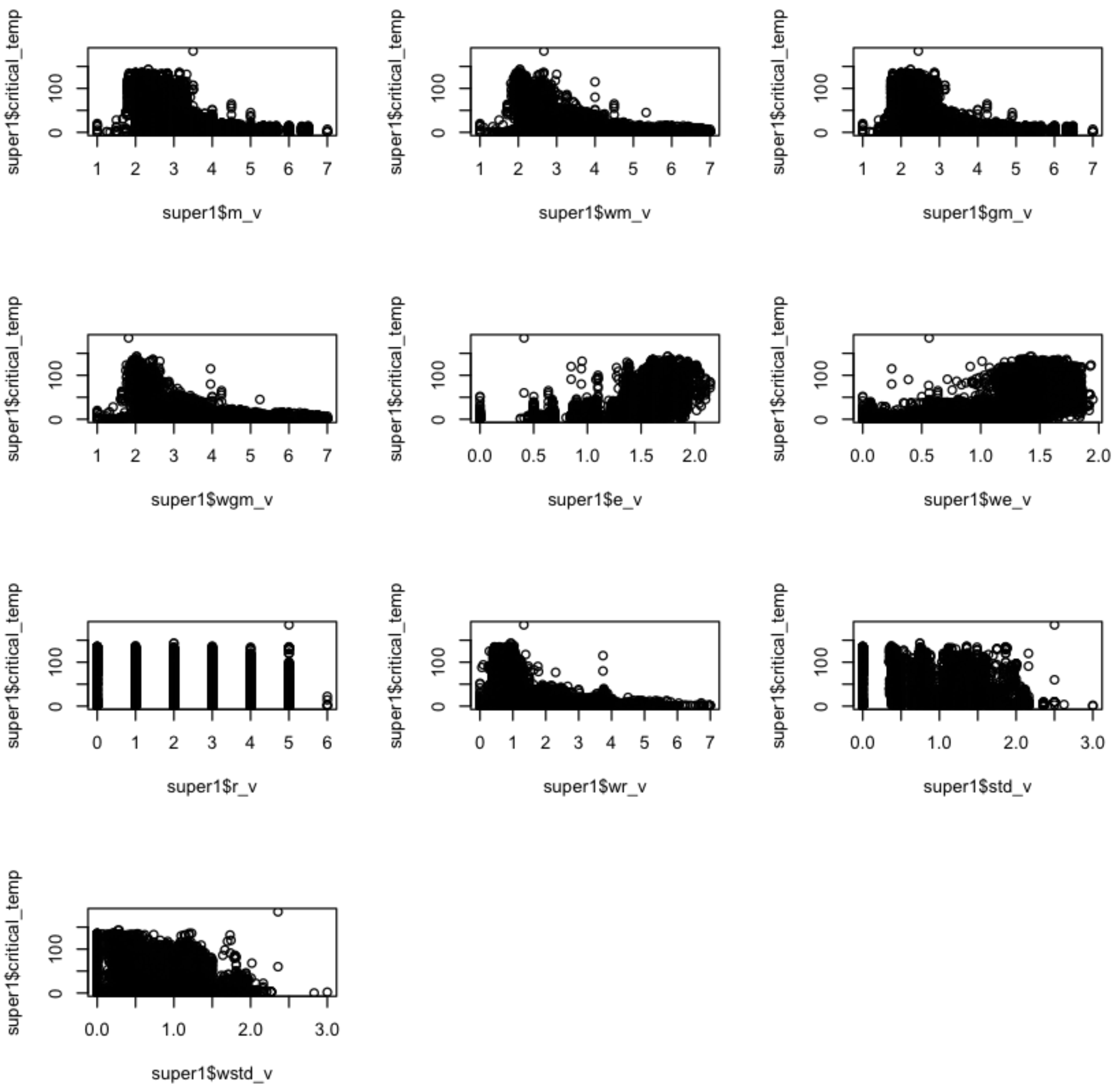
```r
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_tc, super1$critical_temp)
plot(super1$wm_tc, super1$critical_temp)
plot(super1$gm_tc, super1$critical_temp)
plot(super1$wgm_tc, super1$critical_temp)
plot(super1$e_tc, super1$critical_temp)
plot(super1$we_tc, super1$critical_temp)
plot(super1$r_tc, super1$critical_temp)
plot(super1$wr_tc, super1$critical_temp)
plot(super1$std_tc, super1$critical_temp)
plot(super1$wstd_tc, super1$critical_temp)
```



Valence vs Critical Temp Plots:

In [49]:

```r
#Scatter plots of Atomic Mass based variables:
par(mfrow = c(4,3))
#dev.new(width=5, height=4)
plot(super1$m_v, super1$critical_temp)
plot(super1$wm_v, super1$critical_temp)
plot(super1$gm_v, super1$critical_temp)
plot(super1$wgm_v, super1$critical_temp)
plot(super1$e_v, super1$critical_temp)
plot(super1$we_v, super1$critical_temp)
plot(super1$r_v, super1$critical_temp)
plot(super1$wr_v, super1$critical_temp)
plot(super1$std_v, super1$critical_temp)
plot(super1$wstd_v, super1$critical_temp)
```



range_valence or r_v -> an intriguing variable. Might have very low variance or zero variance.

In broad outline, EDA includes checks on data quality, calculation of summary statistics, plotting appropriate graphs as and when necessary and using complicated data-analytic techniques to derive information from the data. (Chatfield, 1986)

Since we have already seen the summary/descriptive statistics, plotted scatter plots between each variable and predictor, let us delve into the actual relationship between these predictors that can be explained by correlation between them.

Correlation between Variables:

In [50]:

```
#Computing the correlation between all the variables in the dataset
super.cor <- cor(super1)
super.cor
```

	number_of_elements	**m_am**	**wm_am**	**gm_am**	**w**
number_of_elements	1.000000000	-0.141922797	-0.353064435	-0.292968819	-0.454
m_am	-0.141922797	1.000000000	0.815977034	0.940298163	0.745
wm_am	-0.353064435	0.815977034	1.000000000	0.848241529	0.964
gm_am	-0.292968819	0.940298163	0.848241529	1.000000000	0.856
wgm_am	-0.454525138	0.745840579	0.964085105	0.856975448	1.000
e_am	0.939304058	-0.104000152	-0.308046011	-0.190213789	-0.370
we_am	0.881845150	-0.097609229	-0.412665507	-0.232183150	-0.484
r_am	0.682777066	0.125658599	-0.144029437	-0.175860637	-0.352
wr_am	-0.320293446	0.446224904	0.716623230	0.458473082	0.673
std_am	0.513998153	0.196460491	-0.060739153	-0.121707976	-0.274
wstd_am	0.546391203	0.130675110	-0.089470766	-0.166042011	-0.331
m_fie	0.167450898	-0.285781955	-0.209296217	-0.367690186	-0.276
wm_fie	0.484444731	-0.222097406	-0.522595267	-0.354664335	-0.612
gm_fie	0.024228530	-0.240565224	-0.109490435	-0.286844286	-0.154
wgm_fie	0.424152224	-0.219381273	-0.508108815	-0.341584501	-0.588
e_fie	0.973195335	-0.166934898	-0.369772508	-0.316670438	-0.471
we_fie	0.719209167	-0.163564893	-0.129779058	-0.287701018	-0.227
r_fie	0.781226913	-0.255627735	-0.452303149	-0.431689462	-0.575

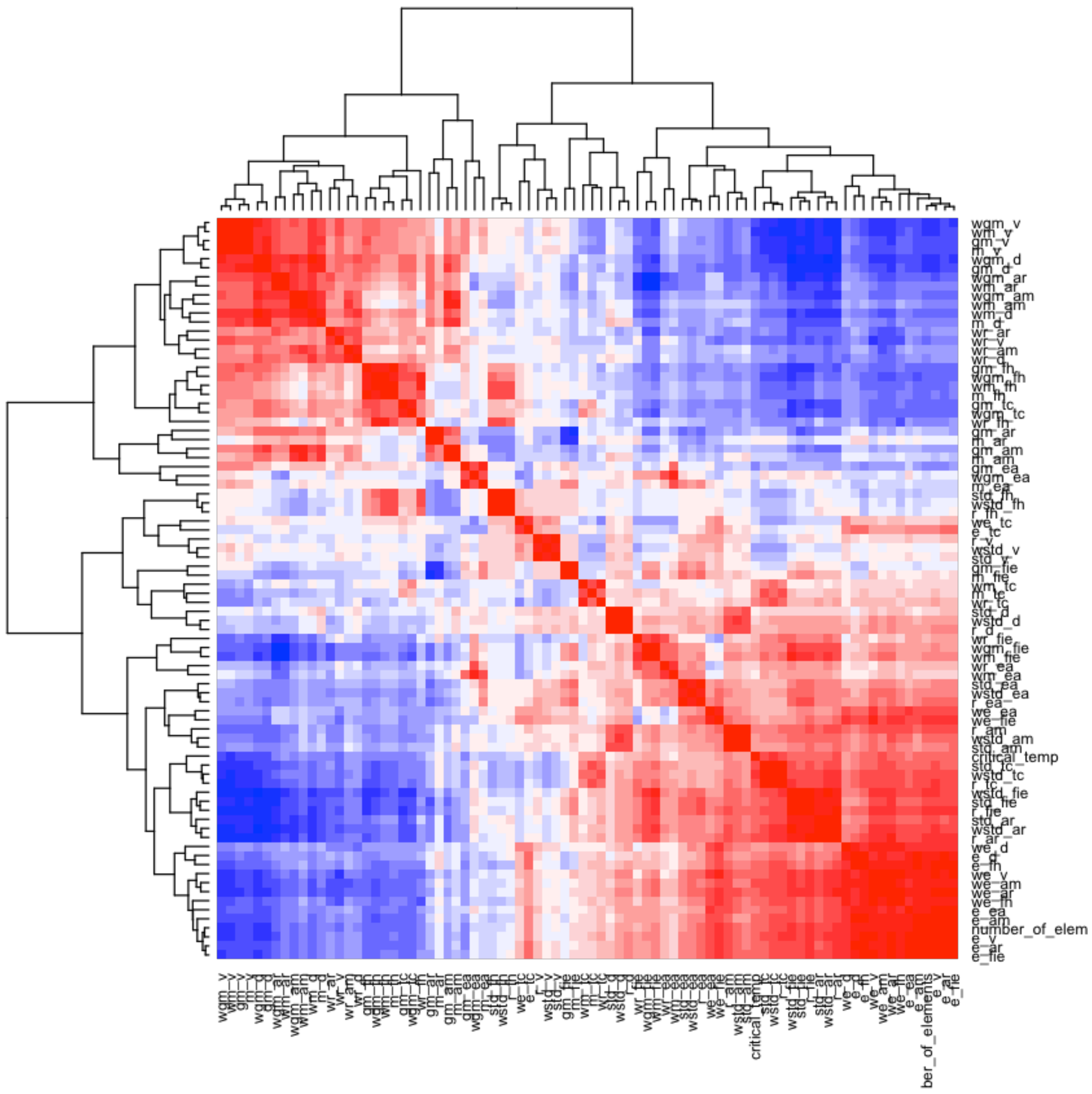
wr_fie	0.329623733	-0.080544826	-0.420457052	-0.155439062	-0.451
std_fie	0.674004751	-0.276560574	-0.459323462	-0.450045014	-0.578
wstd_fie	0.717830737	-0.222811916	-0.492250297	-0.390842997	-0.617
m_ar	-0.001388822	0.497663631	0.288451485	0.510867018	0.301
wm_ar	-0.422144163	0.376759801	0.660011051	0.488821849	0.720
gm_ar	-0.240444347	0.561060699	0.468457460	0.647559721	0.527
wgm_ar	-0.518255859	0.359894450	0.667112083	0.496461190	0.749
e_ar	0.972245245	-0.140034439	-0.345070536	-0.282047842	-0.441
we_ar	0.904120682	-0.147603805	-0.400482727	-0.311701386	-0.514
r_ar	0.768060022	-0.270694807	-0.524860727	-0.460197290	-0.645
wr_ar	-0.371349583	0.141100325	0.363882464	0.240295641	0.432
std_ar	0.624810298	-0.326402906	-0.551140623	-0.512841330	-0.665
wstd_ar	0.695088577	-0.280439751	-0.554820067	-0.462396711	-0.681
m_d	-0.418674700	0.756861118	0.749260525	0.779757034	0.740
wm_d	-0.507895375	0.608934913	0.842664701	0.677131018	0.852
gm_d	-0.630504092	0.596484643	0.712815440	0.728477184	0.789
wgm_d	-0.649882311	0.525588122	0.767010936	0.663642015	0.843
e_d	0.871831749	-0.043415596	-0.246377142	-0.125672127	-0.300
we_d	0.767078421	0.026324752	-0.195893891	-0.093880853	-0.273
r_d	0.413485633	0.198067091	-0.002868148	-0.024974646	-0.163
wr_d	-0.355389487	0.342390738	0.585686944	0.368142682	0.576
std_d	0.210723871	0.245042403	0.103156510	0.037866071	-0.048
wstd_d	0.334072073	0.180942527	0.009921269	-0.037299230	-0.174
m_ea	-0.119302572	0.088229536	0.147303238	0.079376432	0.119
wm_ea	0.195607767	0.061103034	-0.096426785	-0.006352581	-0.158
gm_ea	-0.356067248	0.189282435	0.272260819	0.219650894	0.274
wgm_ea	-0.052883973	0.134382340	0.021876579	0.111858038	-0.011
e_ea	0.877304020	-0.091539414	-0.290219664	-0.238002465	-0.395

we_ea	0.625798404	-0.107650648	-0.093796393	-0.224756609	-0.194
r_ea	0.531540406	-0.187068587	-0.225890340	-0.284097947	-0.300
wr_ea	0.241411233	0.010235114	-0.204480413	-0.055316460	-0.246
std_ea	0.423738261	-0.164960192	-0.197728679	-0.249926458	-0.262
wstd_ea	0.480812544	-0.133100550	-0.210756959	-0.238027624	-0.291
m_fh	-0.437624455	-0.137668902	0.006730077	-0.092243615	0.054
wm_fh	-0.449271665	-0.135428565	0.014681004	-0.089138720	0.072
gm_fh	-0.514251754	0.014818495	0.164239431	0.086599422	0.219
wgm_fh	-0.519109340	-0.043002734	0.120043990	0.024199132	0.189
e_fh	0.900759345	-0.008498640	-0.225286701	-0.126798354	-0.313
we_fh	0.860478819	-0.028540944	-0.237218403	-0.171927681	-0.349
r_fh	0.005734172	-0.347581882	-0.283419738	-0.384838353	-0.279
wr_fh	-0.371787677	-0.167527922	-0.070411491	-0.128757736	-0.006
std_fh	-0.113361228	-0.337968801	-0.253910567	-0.361244227	-0.239
wstd_fh	-0.074796377	-0.335778336	-0.272805875	-0.372665857	-0.277
m_tc	0.227655893	-0.158265641	-0.236418136	-0.190936638	-0.248
wm_tc	0.206068511	-0.065988791	-0.058075067	-0.104939744	-0.056
gm_tc	-0.485323530	0.006004074	0.184990043	0.110768515	0.271
wgm_tc	-0.469206128	0.056394180	0.250226215	0.131642343	0.322
e_tc	0.501871343	-0.100076682	-0.076936360	-0.116455185	-0.076
we_tc	0.207065208	-0.098220524	0.025638296	-0.104643666	0.020
r_tc	0.696059989	-0.114537871	-0.376573091	-0.243464920	-0.464
wr_tc	0.316771509	-0.027789696	-0.108511530	-0.095661466	-0.129
std_tc	0.602018066	-0.110657807	-0.362511650	-0.233587261	-0.447
wstd_tc	0.665579856	-0.110856410	-0.350993285	-0.232079144	-0.431
m_v	-0.609412358	0.374098807	0.534450406	0.487020743	0.599

```
         wm_v     -0.648550851   0.304683244   0.545587036   0.427960618   0.614

         gm_v     -0.618512449   0.392152739   0.539780205   0.511507826   0.608

        wgm_v     -0.659267738   0.321398940   0.548980987   0.450356645   0.623

          e_v      0.967832451  -0.156786430  -0.375718236  -0.306246222  -0.477

         we_v      0.892559475  -0.145610386  -0.331024627  -0.307662306  -0.448
          r_v      0.231873976  -0.107449727  -0.039155397  -0.165010264  -0.078

         wr_v     -0.447769825   0.168632512   0.330903813   0.272302621   0.409

        std_v      0.105365152  -0.080278563  -0.003681475  -0.124626855  -0.033
       wstd_v      0.035216288  -0.081252677   0.077322868  -0.117335886   0.030

critical_temp      0.601068571  -0.113523246  -0.312272020  -0.230345375  -0.369
```

In [51]:

```r
palette = colorRampPalette(c("blue", "white", "red")) (20)
heatmap(x = super.cor, col = palette, symm = TRUE)

#Red -> high positive correlation
#Blue -> high negative correlation
#White -> little to now correlation
```



In the above heat map, <font color = 'red'> RED </font> indicates high positive correlation, BLUE indicates high negative correlation and WHITE indicates no correlation between the predictors themselves.

Now that we have performed significant exploration of the data, we can move to the model development stage where we will be looking at a few regression methods along with appropriate feature selection methodologies.

3. Model Development

"*The aim of regression models is to model the variation of a quantitative response variable y in terms of the variation of one or several explanatory variables (x1,. . .,xp)*" (Härdle W.K., Simar L., 2012)

In the steps that follow, we will be formulating a few models which could help us predict the Critical Temperature of a superconductor. Variable subset selection will be done independently for each model based on the type of model being used .

SPLITTING THE DATA:

- Randomly setting the seed to a number so that a random split is generated everytime this code is run. This ensures that the data is fully shuffled everytime for better prediction purposes.
- Then, <font color = 'green'> sample( ) </font> function is used to sample the data, in this case 1 sample containing all the data, but in a shuffled manner to ensure a good standard of practise.
- Finally, the shuffled data is assigned to **shuf_super** which will now be split into training and testing sets.

In [52]:

```
set.seed(62)
row <- sample(nrow(super1))
shuf_super <- super1[row,]
```

"At the end of the training process, the final model should predict correct outputs for the input samples from T, but it should also be able to generalize well to previously unseen data. Poor generalization can be characterized by over-training. If the model over-trains, it just memorizes the training examples and it is not able to give correct outputs also for patterns that were not in the training dataset." (Reitermanova, 2010)

Thus, splitting a dataset into 2, one for training and one for testing is extremely important to avoid the errors and reduce the `"Bias Variance Tradeoff"`.

- We will split the data in a ratio of 4:1. i.e. **80% Training Data** and **20% Testing Data.**
- For this purpose, the <font color = 'green'>createDataPartition( )</font> function is used.

In [53]:

```
#Determining the row to split the data on
split <- createDataPartition(shuf_super$critical_temp, p=0.8, list=FALSE)

#Creating the train dataset
train <- shuf_super[split,]

#Creating the test dataset
test <- shuf_super[-split,]
```

- Let us also divide the train and test data into a subsest of the variables (all 81 of them) and the target variable (1 of them)

In [54]:

```
train.predictors <- train[,-82]
train.target <- train[,82]
test.predictors <- test[,-82]
test.target <- test[,82]
```

FEATURE SELECTION:

"Generally, features are characterized as:

1. Relevant: These are features which have an influence on the output and their role can not be assumed by the rest
2. Irrelevant: Irrelevant features are defined as those features not having any influence on the output, and whose values are generated at random for each example.
3. Redundant: A redundancy exists whenever a feature can take the role of another (perhaps the simplest way to model redundancy)." (Karagiannopoulos et. al, no date)

Thus, we will be using several methods while developing the models below which will help us select the best features for implementation. Feature selection can be done by: **Filter, Wrapper or Embedded** methods.

MODEL 1: MULTIPLE LINEAR REGRESSION WITH FILTER METHODS

"Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y."

Variable selection can be done for model development in many methods. For this linear regression model, we will be looking into some filter methods which will help us rank the predictors in order of relevance and helps us measure a particular feature's influence on the target, eg: correlation between variables and target. Irrelevancy can be eliminated by discarding the features that would have little to no influence on the model such as the ones with extremely low variance, as low variability affects regression models. Finally, redundancy can be eliminated by targeting multicollinearity within the features. We will look into this further:

Correlation Criteria (Relevance)

- **Correlation** is the measure which indicates the relationship between two variables. In our dataset, since we have a large number of predictor variables, it helps to identify the ones which are highly correlated to try to identify the type of relationship between these variables for better model development purposes.

- This will be checked by using the <font color= 'green'> cor( )</font> function which returns the Pearson's correlation coefficient.

In [55]:

```
co <- data.frame(cor(train[1:81],train[82]))
co <- setDT(co, keep.rownames = TRUE)[]
new <- co[order(abs(critical_temp))]
#Displaying the top 30% of the features with the highest correlation
tail(new,24)
```

rn	critical_temp
e_am	0.5435378
std_fie	0.5438187
gm_d	-0.5439631
e_fh	0.5523997
e_ar	0.5580286
we_fh	0.5612418
std_ar	0.5623458
e_fie	0.5669584
gm_v	-0.5751778
wstd_fie	0.5824024
we_v	0.5903004
e_v	0.5978685
wstd_ar	0.6004421
number_of_elements	0.6005712
m_v	-0.6018994
r_fie	0.6021512
we_ar	0.6026045
wgm_v	-0.6178228
we_am	0.6267691
wm_v	-0.6343775
std_tc	0.6528679
r_ar	0.6556092
r_tc	0.6866783
wstd_tc	0.7228292

Here, we have the list of the top <font color = 'blue'> 30 % </font> of the variables ordered by their correlation with the target variable. For the development of the initial model, we can try to go ahead with these features and perform more analysis as we go ahead:

In [56]:

```
vars<- tail(new$rn,24)
vars[25] = 'critical_temp'
#Function to extract the specific columns from the dataframe
extract_columns <- function(data,var) {
    extracted_data <- data %>%
        select_(.dots = var)
    return(extracted_data)
}

#Passing the 'train' dataframe with all variables and retrieving a 'train1' da
taframe with selected 30 vars
train1 <- extract_columns(train,vars)
```

In [57]:

```
#head(train1)
dim(train1)
```

```
17011   25
```

Variance Criteria (Irrelevance)

- **Variance** of a predictor is one of the important factors to consider during the process of feature selection. Ideally, you would like all of your predictor values to be variant to provide the best and stable outcome for your prediction, but sometimes that is not the case.
- "*These so-called “near zero-variance predictors” can cause numerical problems during resampling for some models, such as linear regression*" (Kuhn, 2008). Since the first model we are computing is a multiple-linear regression model, it is safer to eliminate these predictors, if any.
- This will be done by using the nearZeroVar( ) function from the <font color= 'green'>caret </font> package.

In [58]:

```
nzv <- nearZeroVar(train1, saveMetrics= TRUE)
nzv[nzv$nzv,][1:5,]
```

	freqRatio	percentUnique	zeroVar	nzv
r_tc	24.33333	2.580683	FALSE	TRUE
NA	NA	NA	NA	NA
NA.1	NA	NA	NA	NA
NA.2	NA	NA	NA	NA
NA.3	NA	NA	NA	NA

We can see from the above output that there is only one predictor, r_tc (range_ThermalConductivity), which has a near zero variance. Let us remove this variable from our dataset and continue with the analysis.

In [59]:

```
nzv <- nearZeroVar(train1)
trains <- train1[, -nzv]
dim(trains)
```

```
17011  24
```

Redundancy Criteria (Multicollinearity)

In [60]:

```
cor(trains)
```

	e_am	std_fie	gm_d	e_fh	e_ar	we_fh
e_am	1.0000000	0.6031344	-0.5478821	0.9288372	0.9721652	0.8451658
std_fie	0.6031344	1.0000000	-0.8089333	0.4775156	0.6418863	0.5589844
gm_d	-0.5478821	-0.8089333	1.0000000	-0.4493800	-0.5954943	-0.5028169
e_fh	0.9288372	0.4775156	-0.4493800	1.0000000	0.9312397	0.8819167
e_ar	0.9721652	0.6418863	-0.5954943	0.9312397	1.0000000	0.8679841
we_fh	0.8451658	0.5589844	-0.5028169	0.8819167	0.8679841	1.0000000
std_ar	0.5692775	0.8760483	-0.8031595	0.4852203	0.6127398	0.5350553
e_fie	0.9644950	0.6733612	-0.6247972	0.9176415	0.9977341	0.8647613
gm_v	-0.5573122	-0.7402440	0.8260959	-0.4951590	-0.5931249	-0.5066060
wstd_fie	0.6611082	0.9343778	-0.7951378	0.5753409	0.6855607	0.6525401
we_v	0.8616050	0.6929071	-0.6229998	0.8246655	0.8996565	0.9083106
e_v	0.9633610	0.6918555	-0.6353083	0.9221954	0.9895608	0.8664223
wstd_ar	0.6481837	0.8590990	-0.7979871	0.5799674	0.6804241	0.6327714
number_of_elements	0.9389885	0.6750714	-0.6311550	0.9016500	0.9720279	0.8606433
m_v	-0.5538916	-0.7215544	0.8115204	-0.5046794	-0.5815610	-0.5070151
r_fie	0.7045358	0.9818244	-0.8157000	0.5961189	0.7412939	0.6578987
we_ar	0.8799557	0.7218116	-0.6538538	0.8415550	0.9142093	0.9077977
wgm_v	-0.5953249	-0.7138098	0.7953745	-0.5633628	-0.6256183	-0.6003254
we_am	0.8894220	0.6626746	-0.5965650	0.8350306	0.8902726	0.8733723
wm_v	-0.5888600	-0.6973042	0.7786380	-0.5612617	-0.6122889	-0.5906187
std_tc	0.5989179	0.6114040	-0.5868615	0.5990330	0.5961358	0.5963150
r_ar	0.7098016	0.8819359	-0.8155271	0.6432901	0.7461311	0.6689321
wstd_tc	0.6519329	0.6257707	-0.6041828	0.6536395	0.6456540	0.6392261
critical_temp	0.5435378	0.5438187	-0.5439631	0.5523997	0.5580286	0.5612418

We can see from the above correlation matrix that the variables are highly correlated with each other which could be a cause of concern as multicollinearity is not a property you would like to have in your dataset. Let us check what we can do in order to explain or repair this outcome.

```
In [61]:
```

```
#Setting an extremely high arbitrary threshold value to check the vif all vari
ables.
vifstep(trains[1:24],th=2000)
```

```
No variable from the 24 input variables has collinearity problem.

The linear correlation coefficients ranges between:
min correlation ( e_fh ~ gm_d ):  -0.4467544
max correlation ( e_fie ~ e_ar ):  0.9977541

---------- VIFs of the remained variables --------
            Variables         VIF
1                e_am   32.251656
2             std_fie  163.056012
3                gm_d    5.367897
4                e_fh   22.568991
5                e_ar 1150.609920
6               we_fh   12.860893
7              std_ar  100.893969
8               e_fie 1691.536609
9                gm_v  925.503303
10           wstd_fie   53.674596
11               we_v   28.372194
12                e_v  986.960003
13            wstd_ar   48.371481
14 number_of_elements   41.017526
15                m_v  912.935437
16              r_fie  173.297078
17              we_ar   62.611072
18              wgm_v  611.933738
19              we_am   25.695426
20               wm_v  609.087256
21             std_tc   17.356693
22               r_ar  110.953949
23            wstd_tc   21.202031
24      critical_temp    2.819749
```

VIF values are said to be high if they exceed >5. But here, the VIF values seem to be completely out of bounds. This high correlation/multicollinearity could be explained by the fact that the variables have been derived from 8 basic properties.

"The fact that some or all predictor variables are correlated among themselves does not, in general, inhibit our ability to obtain a good fit nor does it tend to affect inferences about mean responses or predictions of new observations." —Applied Linear Statistical Models, p289, 4th Edition.

Thus, we could try examining the future model by eliminating variables which have VIF>150 in our dataset, just out of intution.

In [62]:

```
v <-vifstep(trains[1:24],th=150)
v
```

```
5 variables from the 24 input variables have collinearity problem:

e_fie gm_v wgm_v r_fie e_ar

After excluding the collinear variables, the linear correlation co
efficients ranges between:
min correlation ( e_fh ~ gm_d ):  -0.4537581
max correlation ( number_of_elements ~ e_v ):  0.9691109

---------- VIFs of the remained variables --------
            Variables       VIF
1                e_am 22.441076
2             std_fie 32.187058
3                gm_d  5.130122
4                e_fh 18.156723
5               we_fh 12.723701
6              std_ar 70.237477
7            wstd_fie 47.288675
8                we_v 24.944791
9                 e_v 52.915476
10            wstd_ar 47.261214
11 number_of_elements 25.788545
12                m_v 16.780625
13              we_ar 43.187501
14              we_am 20.315218
15               wm_v 16.026122
16             std_tc 15.525964
17               r_ar 81.321476
18            wstd_tc 18.871100
19      critical_temp  2.717609
```

Thus, now we are left with 18 variables and one target in this dataset, by arbitrarily setting the limit as 150. The mean VIF has also drastically reduced as it is a stepwise function and the removal of one variables affects the VIF of another. We are going to exclude the variables with the collinearity problem from the `trains` dataset and proceed to compute a linear regression model.

In [63]:

```
trains <- exclude(trains,v)
```

In [64]:

```
#fitting the linear regression model
lm.mod <- lm(critical_temp~., trains)
```

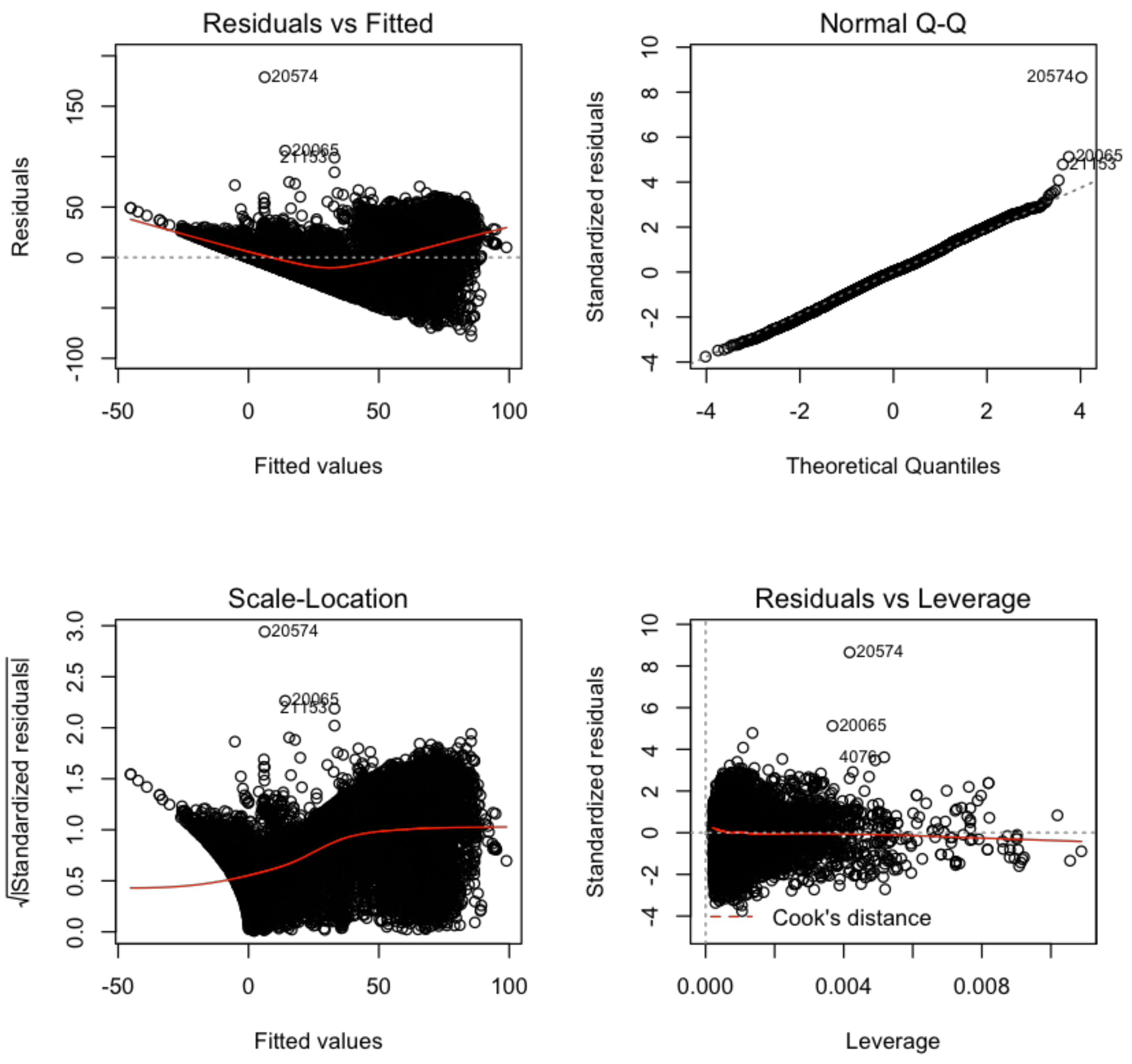
In [65]:

```
summary(lm.mod)
```

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

Residuals:
    Min      1Q  Median      3Q     Max 
-77.846 -13.251   0.291  12.953 178.801 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)         3.029e+01  1.490e+00  20.331  < 2e-16 ***
e_am               -5.771e+01  2.074e+00 -27.820  < 2e-16 ***
std_fie             9.925e-02  8.224e-03  12.069  < 2e-16 ***
gm_d               -5.458e-04  9.742e-05  -5.603 2.14e-08 ***
e_fh                1.434e+00  1.771e+00   0.810  0.41798    
we_fh               1.842e+01  1.515e+00  12.159  < 2e-16 ***
std_ar             -1.652e+00  5.701e-02 -28.985  < 2e-16 ***
wstd_fie           -1.115e-01  8.586e-03 -12.993  < 2e-16 ***
we_v               -1.639e+01  2.099e+00  -7.810 6.07e-15 ***
e_v                 3.146e+01  2.934e+00  10.725  < 2e-16 ***
wstd_ar             2.341e-01  4.325e-02   5.412 6.30e-08 ***
number_of_elements -1.516e+00  5.522e-01  -2.745  0.00606 ** 
m_v                -5.679e+00  6.268e-01  -9.061  < 2e-16 ***
we_ar              -3.605e+01  2.574e+00 -14.008  < 2e-16 ***
we_am               5.351e+01  1.771e+00  30.222  < 2e-16 ***
wm_v                2.135e+00  5.413e-01   3.944 8.05e-05 ***
std_tc             -1.859e-01  1.031e-02 -18.032  < 2e-16 ***
r_ar                6.543e-01  2.058e-02  31.796  < 2e-16 ***
wstd_tc             3.814e-01  1.040e-02  36.690  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 20.7 on 16992 degrees of freedom
Multiple R-squared:  0.6357,	Adjusted R-squared:  0.6353 
F-statistic:  1647 on 18 and 16992 DF,  p-value: < 2.2e-16
```

In [66]:

```
par(mfrow=c(2,2))
plot(lm.mod)
```



In [67]:

```
ggplot(data=trainss, aes(lm.mod$residuals)) +
geom_histogram(binwidth = 1, color = "black", fill = "purple4") +
theme(panel.background = element_rect(fill = "white"),
axis.line.x=element_line(),
axis.line.y=element_line()) +
ggtitle("Histogram for Model Residuals")
```

```
Error in ggplot(data = trainss, aes(lm.mod$residuals)): object 'tr
ainss' not found
Traceback:

1. ggplot(data = trainss, aes(lm.mod$residuals))
```

The relationship between the predictors and the target variable is not linearly explainable as the `Residuals v/s Fitted` plot indicates.

Furthermore, upon inspection of the predictor variables, we can see that std_tc and wstd_tc, two variables which we have identified to have **bimodal** distribution are present in this subset of variables. This can also be a reason why a simple multiple linear regression model cannot explain all of the data.

In [68]:

```
# Predicting the critical_temp of the test predictors
mod.target <-predict(lm.mod, new = test.predictors)

# Computing the MSE
lm.model_mse <- mean((test.target - mod.target)^2)
print(paste('Mean Squared Error of Multiple Linear Regression:',lm.model_mse))

# Computing the R-Squared Value
lm.model_rsv <- cor(test.target, mod.target)^2
print(paste('R-Squared for Multiple Linear Regression:',lm.model_rsv))
```

```
[1] "Mean Squared Error of Multiple Linear Regression: 435.4987306
49402"
[1] "R-Squared for Multiple Linear Regression: 0.626257247849855"
```

MODEL 2: MULTIPLE LINEAR REGRESSION WITH WRAPPER METHODS

- Wrapper methods are like a hit-and-a-miss scenario where they check the performance of a model by adding/removing each predictor and evaluating the change it brings into the model.
- There are different approaches to wrapper methods namely: **Sequential Selection and Heuristic Search Algorithms**
- "Wrapper methods use the predictor as a black box and the predictor performance as the objective function to evaluate the variable subset. Since evaluating 2N subsets becomes a NP-hard problem, suboptimal subsets are found by employing search algorithms which find a subset heuristically." (Chandrashekar et. al, 2013)

In [71]:

```
# Set seed for reproducibility
set.seed(123)
# Set up repeated k-fold cross-validation for 10 folds
train.control <- trainControl(method = "cv", number = 10)
# Train the model
step.model <- train(critical_temp ~., data = train,
                    method = "leapSeq", #This method uses both forwards and ba
ckwards selection and eliminates the need for greed
                    tuneGrid = data.frame(nvmax = 1:81),
                    trControl = train.control
                    )
step.model$results
#step.model$bestTune
```

nvmax	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
1	23.68916	0.5227404	18.20678	0.4069329	0.011887677	0.2407592
2	22.72123	0.5609134	17.63992	0.4265662	0.013243107	0.2858098
3	21.81559	0.5952298	17.18490	0.3495601	0.009700402	0.2261947
4	21.36543	0.6117857	16.76538	0.4129202	0.011317030	0.2269454
5	20.89348	0.6287547	16.45187	0.3617765	0.011126575	0.2164826
6	20.57948	0.6398353	16.14364	0.3678355	0.010915427	0.2178291
7	20.33289	0.6484015	15.72183	0.4111112	0.011469591	0.2414392
8	20.09061	0.6567433	15.47778	0.4279212	0.011746159	0.2497992
9	19.83752	0.6653455	15.42911	0.3457477	0.008130484	0.2007563
10	20.16865	0.6514220	15.68044	1.6233801	0.066505469	1.3642993
11	22.41792	0.5660468	17.58547	3.0654539	0.114512794	2.5262213
12	21.25030	0.6099809	16.60399	2.8483853	0.108029852	2.3198174
13	19.82388	0.6638406	15.37016	2.0370101	0.067735780	1.6687459
14	19.16262	0.6877243	14.80532	0.2736966	0.004956687	0.2463729
15	19.01752	0.6924112	14.65064	0.3059947	0.006303751	0.2404426
16	19.44300	0.6765084	15.01911	1.5724466	0.057981638	1.4242728
17	18.88007	0.6968536	14.51555	0.2904394	0.005724171	0.2236763
18	18.78607	0.6998470	14.41649	0.2915485	0.005835299	0.2336664
19	20.60940	0.6342794	16.02621	2.4607281	0.087086668	2.2145209
20	18.69161	0.7028562	14.35599	0.2738353	0.005626557	0.2313863
21	18.57924	0.7064556	14.24645	0.3014835	0.005920535	0.2716255
22	19.37666	0.6774710	14.94110	1.8993880	0.069766237	1.5882238
23	18.40918	0.7117715	14.09914	0.2548139	0.006345504	0.2059777
24	18.38581	0.7125252	14.08202	0.2728976	0.006628166	0.2091420
25	19.08543	0.6881440	14.64717	1.5801252	0.055514261	1.3354431

26	18.63174	0.7039005	14.27983	1.1067421	0.036645866	0.8086115
27	19.21596	0.6843011	14.67815	1.6283279	0.052372671	1.1817011
28	18.24507	0.7169125	13.94784	0.2694993	0.006151948	0.2307212
29	18.21141	0.7179406	13.90853	0.2777830	0.006390650	0.2149014
30	18.43840	0.7106606	14.12119	1.0174455	0.026788844	0.7862241
31	18.64797	0.7031817	14.25729	1.1872418	0.039339938	0.9027445
32	18.09313	0.7215803	13.79492	0.3049385	0.007592059	0.2065819
33	19.01011	0.6914811	14.54367	1.0998648	0.038255199	0.9027605
34	18.04784	0.7229609	13.75418	0.3044595	0.007433787	0.2297705
35	18.26334	0.7158542	13.92888	0.8583813	0.026949655	0.6827564
36	18.21129	0.7175463	13.86347	0.8408911	0.025948468	0.6643917
37	18.33017	0.7135792	13.94022	0.8107829	0.028364157	0.6507498
38	17.90110	0.7274873	13.59181	0.2901231	0.006887490	0.2269319
39	17.89368	0.7277133	13.57576	0.3031369	0.007202308	0.2358013
40	17.86368	0.7286186	13.56269	0.2930121	0.007161437	0.2133945
41	18.05465	0.7222897	13.72239	0.8662924	0.027236845	0.6723537
42	18.30250	0.7142444	13.88876	1.0767502	0.035580775	0.8624708
43	18.04033	0.7228254	13.66925	0.7241600	0.023379375	0.5008489
44	17.80136	0.7305028	13.48966	0.2834799	0.006980234	0.2275015
45	17.77710	0.7312435	13.46677	0.2796734	0.006662921	0.2214543
46	18.15720	0.7188282	13.77894	1.0023664	0.032844118	0.8558430
47	17.72646	0.7327716	13.43971	0.2760439	0.006190935	0.2358137
48	17.70410	0.7334460	13.42231	0.2694166	0.006124425	0.2227117
49	17.84761	0.7288561	13.54338	0.6678392	0.019693658	0.5894378
50	17.69405	0.7337535	13.40881	0.2927363	0.006223073	0.2306924
51	17.80895	0.7300921	13.50491	0.5936678	0.016996158	0.5317792
52	17.84387	0.7289717	13.53064	0.6549630	0.019364024	0.4990973
53	17.82716	0.7295885	13.50328	0.4887244	0.014393049	0.3741216
54	17.63949	0.7353934	13.34324	0.2906538	0.005933659	0.2354656
55	18.05306	0.7225897	13.69892	0.7230776	0.020236011	0.6610805
56	17.93107	0.7263307	13.59783	0.5850526	0.018125085	0.4904180
57	17.73788	0.7323421	13.42153	0.4359578	0.011800055	0.4062169
58	18.02683	0.7233215	13.67513	0.6826092	0.020108730	0.6530140
59	17.86901	0.7282432	13.53504	0.5938014	0.017745486	0.5233816
60	17.86166	0.7285571	13.54700	0.7431412	0.018483893	0.6489782
61	17.59010	0.7368740	13.29717	0.2722493	0.005676918	0.2422476
62	17.79634	0.7306577	13.50951	0.5890504	0.013190131	0.5685358
63	17.68683	0.7339000	13.39955	0.3872490	0.010827912	0.3350161

```
64  17.57850  0.7372227  13.29196  0.2752096  0.005573310  0.2413526
65  17.58316  0.7370853  13.29592  0.2794671  0.005699298  0.2459209
66  17.63704  0.7354503  13.33618  0.3417941  0.007988569  0.2772516
67  17.65080  0.7350909  13.34174  0.3438630  0.005901890  0.2912669
68  17.68291  0.7340194  13.36900  0.2555359  0.008317623  0.2356220
69  17.65382  0.7349253  13.35224  0.3157337  0.008289990  0.2738759
70  17.63094  0.7356248  13.34773  0.2620633  0.006528547  0.2204127
71  17.66944  0.7345113  13.37430  0.3123297  0.005784910  0.2583230
72  17.62957  0.7356590  13.33717  0.3440330  0.008223652  0.2605901
73  17.67362  0.7343543  13.37414  0.3309930  0.007837515  0.2748610
74  17.72753  0.7327076  13.40194  0.2672438  0.007458989  0.2067941
75  17.56799  0.7375415  13.30167  0.2738096  0.005565643  0.2393507
76  17.72245  0.7328912  13.39675  0.2284759  0.005137378  0.2065644
77  17.61301  0.7361910  13.31895  0.2765239  0.005722663  0.2348059
78  17.61472  0.7361476  13.32542  0.2362516  0.004294750  0.2164531
79  17.63038  0.7356645  13.32622  0.2943008  0.006586147  0.2307190
80  17.61461  0.7361425  13.30967  0.2428038  0.004963498  0.2219284
81  17.56611  0.7376001  13.30393  0.2745499  0.005563751  0.2388539
```

- From the above output, we can see that we can get a healthy r-squared of 0.709 at n=21, thus let's run the command again with the nvmax set to 21 variables only.
- Furthermore, the rmse is also considerably low at n=21.

In [72]:

```r
# Set seed for reproducibility
set.seed(123)
# Set up repeated k-fold cross-validation
train.control <- trainControl(method = "cv", number = 10)
# Train the model
step.model <- train(critical_temp ~., data = train,
                    method = "leapSeq",
                    tuneGrid = data.frame(nvmax = 1:21),
                    trControl = train.control
                    )
step.model$results
```

nvmax	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
1	23.68916	0.5227404	18.20678	0.4069329	0.011887677	0.2407592
2	22.72123	0.5609134	17.63992	0.4265662	0.013243107	0.2858098
3	21.81559	0.5952298	17.18490	0.3495601	0.009700402	0.2261947
4	21.36543	0.6117857	16.76538	0.4129202	0.011317030	0.2269454
5	20.89348	0.6287547	16.45187	0.3617765	0.011126575	0.2164826
6	20.57948	0.6398353	16.14364	0.3678355	0.010915427	0.2178291
7	20.33289	0.6484015	15.72183	0.4111112	0.011469591	0.2414392
8	20.09061	0.6567433	15.47778	0.4279212	0.011746159	0.2497992
9	19.83752	0.6653455	15.42911	0.3457477	0.008130484	0.2007563
10	20.16865	0.6514220	15.68044	1.6233801	0.066505469	1.3642993
11	22.41792	0.5660468	17.58547	3.0654539	0.114512794	2.5262213
12	21.25030	0.6099809	16.60399	2.8483853	0.108029852	2.3198174
13	19.82388	0.6638406	15.37016	2.0370101	0.067735780	1.6687459
14	19.16262	0.6877243	14.80532	0.2736966	0.004956687	0.2463729
15	19.01752	0.6924112	14.65064	0.3059947	0.006303751	0.2404426
16	19.44300	0.6765084	15.01911	1.5724466	0.057981638	1.4242728
17	18.88007	0.6968536	14.51555	0.2904394	0.005724171	0.2236763
18	18.78607	0.6998470	14.41649	0.2915485	0.005835299	0.2336664
19	20.60940	0.6342794	16.02621	2.4607281	0.087086668	2.2145209
20	18.69161	0.7028562	14.35599	0.2738353	0.005626557	0.2313863
21	18.57924	0.7064556	14.24645	0.3014835	0.005920535	0.2716255

In [73]:

```
#Extracting the dataset
a <- coef(step.model$finalModel, 21)
a <- data.frame(a)
variables <- as.vector(rownames(a))
variables <- variables[2:21]
variables[21] <- 'critical_temp'
variables
```

```
'e_am'  'r_am'  'wstd_am'  'm_fie'  'r_ar'  'std_ar'  'r_d'  'm_ea'  'wm_ea'
'gm_ea'  'wgm_ea'  'r_ea'  'std_ea'  'wstd_ea'  'r_fh'  'gm_tc'  'we_tc'  'r_tc'
'wstd_tc'  'r_v'  'critical_temp'
```

Now we have extracted the dataset with variables that have been selected using the wrapper methods' **stepwise functionality.**

In [74]:

```
#Extracting the data for only the selected columns
trainstep <- extract_columns(train,variables)
dim(trainstep)
```

```
17011   21
```

In [75]:

```
#Fitting the linear model
step.mod <- lm(critical_temp~., trainstep)
```

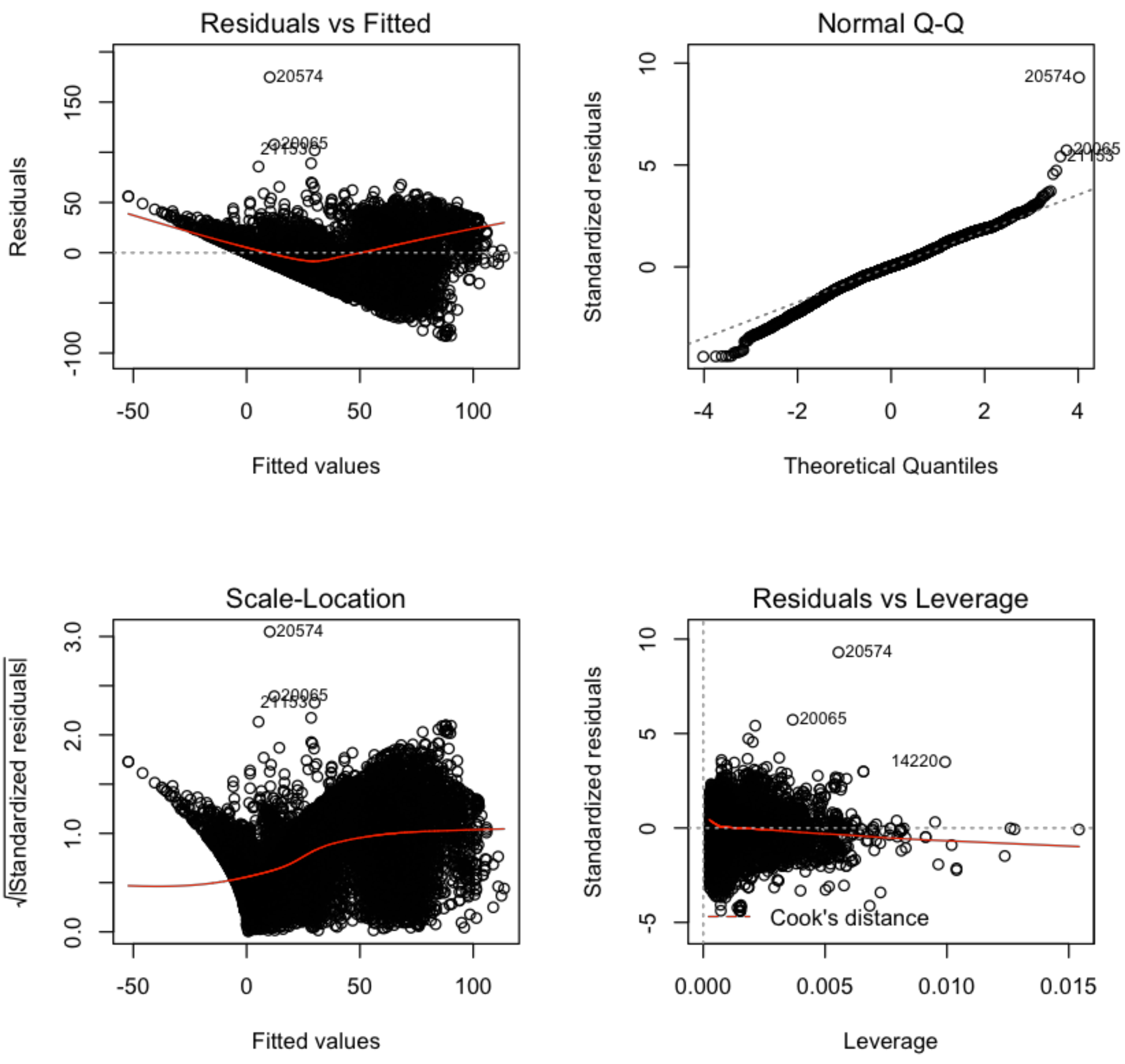
In [76]:

```
#Summary of the fit
summary(step.mod)
```

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

Residuals:
    Min      1Q  Median      3Q     Max 
-82.943 -10.675   0.404  11.644 174.797 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -5.600e+00  2.093e+00  -2.676  0.00747 ** 
e_am        -1.570e+01  1.068e+00 -14.692  < 2e-16 ***
r_am         2.375e-01  8.684e-03  27.355  < 2e-16 ***
wstd_am     -5.472e-01  2.171e-02 -25.211  < 2e-16 ***
m_fie        2.168e-02  2.503e-03   8.662  < 2e-16 ***
r_ar         4.951e-01  1.729e-02  28.634  < 2e-16 ***
std_ar      -8.494e-01  4.113e-02 -20.651  < 2e-16 ***
r_d         -5.414e-04  5.325e-05 -10.167  < 2e-16 ***
m_ea        -7.132e-01  4.086e-02 -17.457  < 2e-16 ***
wm_ea        1.369e+00  3.978e-02  34.404  < 2e-16 ***
gm_ea        7.002e-01  3.687e-02  18.992  < 2e-16 ***
wgm_ea      -1.405e+00  3.916e-02 -35.885  < 2e-16 ***
r_ea        -2.986e-01  1.470e-02 -20.308  < 2e-16 ***
std_ea       1.505e+00  5.037e-02  29.876  < 2e-16 ***
wstd_ea     -1.119e+00  3.401e-02 -32.909  < 2e-16 ***
r_fh        -1.440e-01  8.890e-03 -16.197  < 2e-16 ***
gm_tc       -6.210e-02  6.498e-03  -9.556  < 2e-16 ***
we_tc        1.679e+01  8.017e-01  20.940  < 2e-16 ***
r_tc        -9.350e-02  3.939e-03 -23.736  < 2e-16 ***
wstd_tc      4.560e-01  1.020e-02  44.692  < 2e-16 ***
r_v         -2.132e+00  1.535e-01 -13.888  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18.85 on 16990 degrees of freedom
Multiple R-squared:  0.698,	Adjusted R-squared:  0.6976 
F-statistic:  1963 on 20 and 16990 DF,  p-value: < 2.2e-16
```

In [77]:

```
par(mfrow=c(2,2))
plot(step.mod)
```



In [78]:

```
# Predicting the critical_temp of the test predictors
mod.target <-predict(step.mod, new = test.predictors)

# Computing the MSE
lm.mod2_mse <- mean((test.target - mod.target)^2)
print(paste('Mean Squared Error of Multiple Linear Regression:',lm.mod2_mse))

# Computing the R-Squared Value
lm.mod2_rsv <- cor(test.target, mod.target)^2
print(paste('R-Squared for Multiple Linear Regression:',lm.mod2_rsv))
```

```
[1] "Mean Squared Error of Multiple Linear Regression: 364.5878937
75746"
[1] "R-Squared for Multiple Linear Regression: 0.687142499053063"
```

We can see from the above output that :

1. **The R-squared** value is higher than the previous multiple linear regression model. Which could mean better prediction accuracy
2. **The F-statistic** has also jumped up a notch.

```
In [79]:
```

```
cor(trainstep[1:20], trainstep[21])
```

	critical_temp
e_am	0.5435378
r_am	0.4914067
wstd_am	0.3588237
m_fie	0.1010390
r_ar	0.6556092
std_ar	0.5623458
r_d	0.2637002
m_ea	-0.1959834
wm_ea	0.1088088
gm_ea	-0.3811771
wgm_ea	-0.1095284
r_ea	0.2789385
std_ea	0.2616284
wstd_ea	0.3124665
r_fh	-0.1431193
gm_tc	-0.3883755
we_tc	-0.1195392
r_tc	0.6866783
wstd_tc	0.7228292
r_v	-0.1448442

`An interesting observation` is that: the variables that have been chosen by the step function do not seem to be correlated with the target variable at all. This is something that could be a huge red flag during prediction.

Although the **R squared** seemed to have improved, the relationship between the regressors and the target does not seem to be strong and the evidence from the correlation matrix suggests against it.

Thus, we can safely forgo taking this model into consideration and move on to other methods.

MODEL 3 : Extreme Gradient Boost (XGBoost)

In the paper that is the basis of this assignment, the authors seemed to stress on the fact that the XGBoost model was able to give them more prediction accuracy in comparison to the multiple regression model developed by them. Intuitively, I have also decided to explore the boost in accuracy that this model brings to the table.

- Upon a quick research, it has also been found that XGBoost is a gradient boosting library and has some inbuilt **regularisation methods** which are immune to multi-collinearity, a phenomenon which seemed to have a significant affect on the linear models developed before.
- It is also enables with some internal **cross validation** techniques, which can be used during model development processes.

In [80]:

```
#install.packages('xgboost')
```

XGBoost takes in data in the form of a matrix and thus, we will be converting the training and testing dataframes into matrices.

In [82]:

```
#preparing matrices for parsing
xgbtrain <- xgb.DMatrix(data = as.matrix(train.predictors),label = as.matrix(t
rain.target))
xgbtest <- xgb.DMatrix(data = as.matrix(test.predictors),label=as.matrix(test.
target))
```

In [83]:

```
#Generating default parameters to build the first model on
params <- list(booster = "gbtree", objective = "reg:linear", eta=0.3, gamma=0,
               max_depth=6, min_child_weight=1, subsample=1, colsample_bytree=
1)
```

"Using the inbuilt xgb.cv function, let's calculate the best nround for this model. In addition, this function also returns CV error, which is an estimate of test error."

In [84]:

```
xgbcv <- xgb.cv( params = params, data = xgbtrain, nrounds = 100, showsd= T,
                nfold = 10, print.every.n = 10)
```

```
Warning message:
“'print.every.n' is deprecated.
Use 'print_every_n' instead.
See help("Deprecated") and help("xgboost-deprecated").”

[1]     train-rmse:35.471250+0.093575   test-rmse:35.551220+0.5855
04
[11]    train-rmse:10.076469+0.094814   test-rmse:11.478412+0.3104
85
[21]    train-rmse:8.736347+0.086450    test-rmse:10.770055+0.3712
25
[31]    train-rmse:8.048167+0.074983    test-rmse:10.495155+0.3961
97
[41]    train-rmse:7.553851+0.055001    test-rmse:10.317308+0.4191
18
[51]    train-rmse:7.147134+0.078783    test-rmse:10.197543+0.3978
92
[61]    train-rmse:6.805248+0.073109    test-rmse:10.104826+0.4047
22
[71]    train-rmse:6.507882+0.054745    test-rmse:10.029500+0.3969
94
[81]    train-rmse:6.255416+0.055524    test-rmse:9.969836+0.40539
1
[91]    train-rmse:6.051318+0.053126    test-rmse:9.948029+0.41043
9
[100]   train-rmse:5.889784+0.038735    test-rmse:9.926119+0.42014
9
```

In [85]:

```
#first default - model training
xgb.mod <- xgb.train(params = params,
                        data = xgb.DMatrix(data = as.matrix(train.predictors),
label = as.matrix(train.target)),
                        nrounds = 3000, nfold = 10, showsd = T, stratified = T
,
                        print_every_n = 10,early_stopping_rounds = 100,
                        watchlist = list(test = xgb.DMatrix(data = as.matrix(t
est.predictors),label = as.matrix(test.target))))
```

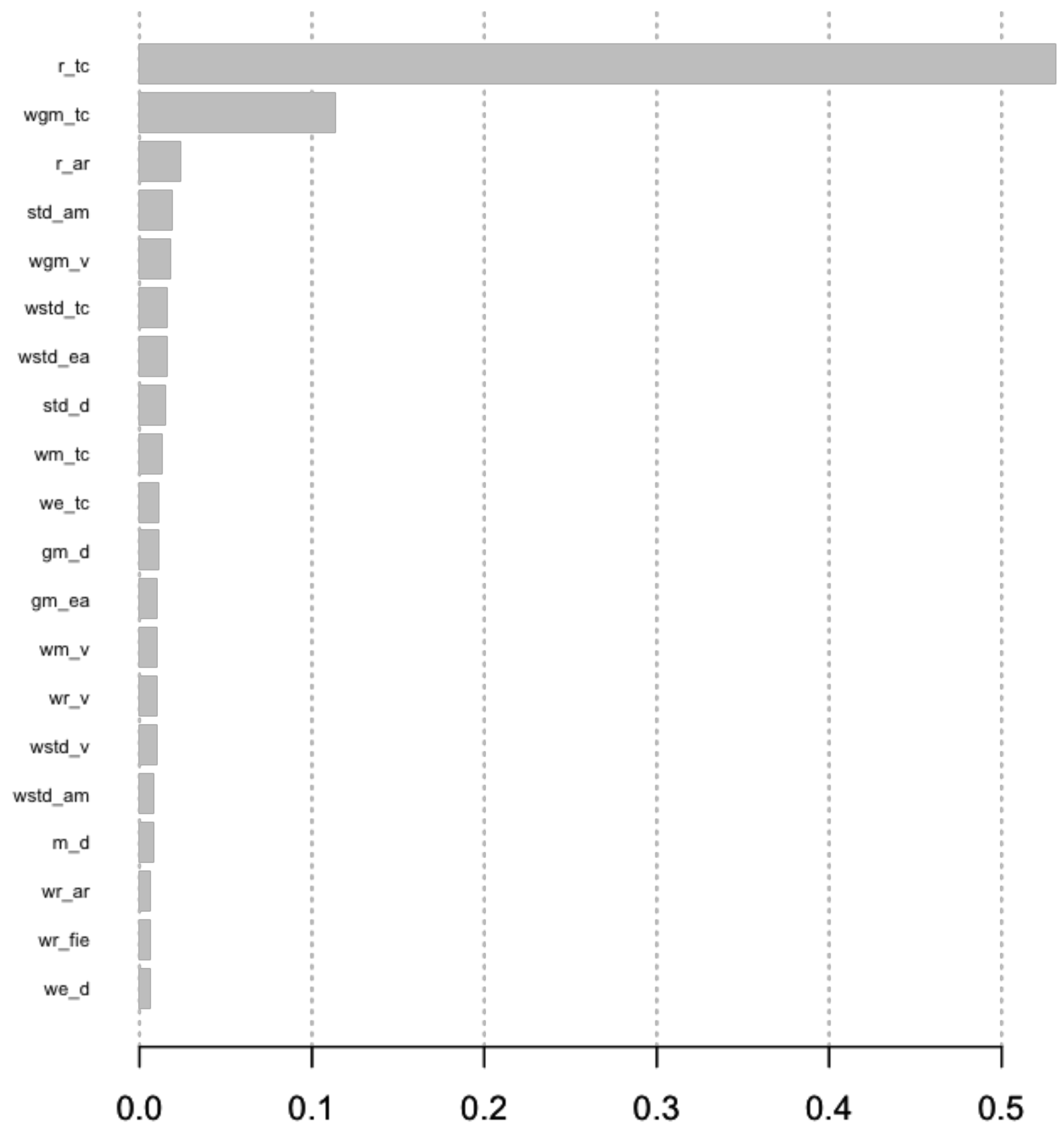
```
[1]	test-rmse:35.276165
Will train until test_rmse hasn't improved in 100 rounds.

[11]	test-rmse:11.352916
[21]	test-rmse:10.667747
[31]	test-rmse:10.356871
[41]	test-rmse:10.166113
[51]	test-rmse:10.034712
[61]	test-rmse:9.980517
[71]	test-rmse:9.910322
[81]	test-rmse:9.872714
[91]	test-rmse:9.834014
[101]	test-rmse:9.815414
[111]	test-rmse:9.812716
[121]	test-rmse:9.785287
[131]	test-rmse:9.759849
[141]	test-rmse:9.723497
[151]	test-rmse:9.711094
[161]	test-rmse:9.684143
[171]	test-rmse:9.693506
[181]	test-rmse:9.680573
[191]	test-rmse:9.682490
[201]	test-rmse:9.680367
[211]	test-rmse:9.680950
[221]	test-rmse:9.680128
[231]	test-rmse:9.669400
[241]	test-rmse:9.662138
[251]	test-rmse:9.669702
[261]	test-rmse:9.684066
[271]	test-rmse:9.682046
[281]	test-rmse:9.686298
[291]	test-rmse:9.690759
[301]	test-rmse:9.686247
[311]	test-rmse:9.692360
[321]	test-rmse:9.692547
[331]	test-rmse:9.695355
[341]	test-rmse:9.697485
Stopping. Best iteration:
[241]	test-rmse:9.662138
```

- From this model, let us print out the top 20 variables with the **highest gain.**

In [87]:

```
high.gain <- xgb.importance(colnames(train.predictors), model = xgb.mod)

xgb.plot.importance(importance_matrix = high.gain[1:20])
print(high.gain[1:20])
```

```
     Feature        Gain       Cover   Frequency
 1:     r_tc 0.531166561 0.005682358 0.001995596
 2:   wgm_tc 0.113363507 0.010460162 0.012248830
 3:     r_ar 0.023875344 0.003304206 0.002614919
 4:   std_am 0.019208324 0.009162641 0.008601707
 5:    wgm_v 0.017703838 0.018990899 0.009358657
 6:  wstd_tc 0.016252701 0.036438644 0.022777319
 7:  wstd_ea 0.016223963 0.028113847 0.016652904
 8:    std_d 0.014967749 0.007928180 0.005780347
 9:    wm_tc 0.013093914 0.019978405 0.016446463
10:    we_tc 0.011455211 0.020896899 0.020919350
11:     gm_d 0.011314336 0.007545330 0.005917974
12:    gm_ea 0.010425610 0.010107176 0.005367465
13:     wm_v 0.010021468 0.029707991 0.013969171
14:     wr_v 0.009878199 0.016949034 0.015001376
15:   wstd_v 0.009873890 0.007009513 0.012042389
16:  wstd_am 0.008527059 0.011123855 0.020850537
17:      m_d 0.007878599 0.008160793 0.006537297
18:    wr_ar 0.006869601 0.017325244 0.018786127
19:   wr_fie 0.006833986 0.027753100 0.020231214
20:     we_d 0.006698233 0.024610887 0.020093587
```



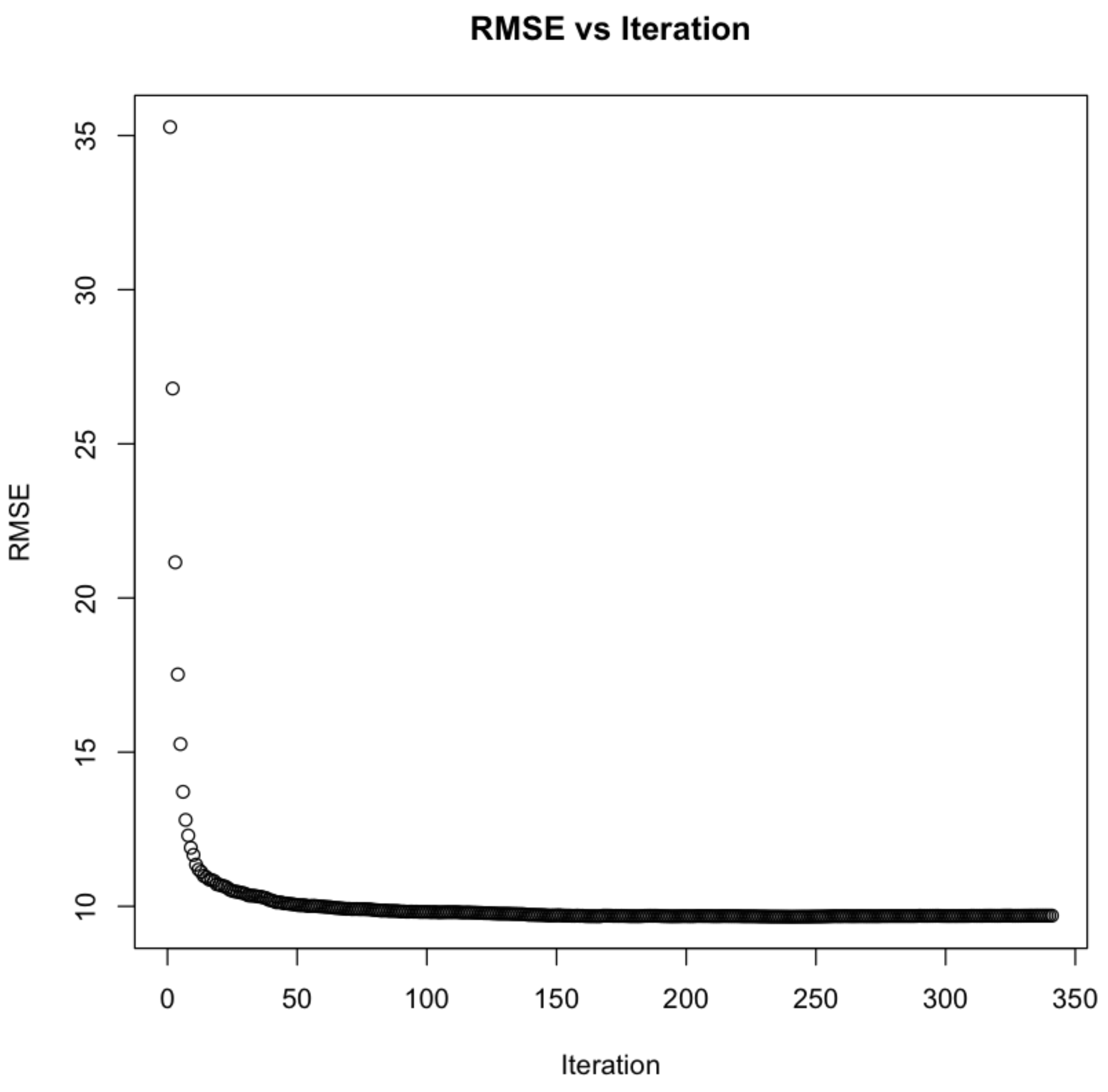
In [88]:

```
xgb.log <- data.frame(xgb.mod$evaluation_log)
head(xgb.log)
```

iter	test_rmse
1	35.27616
2	26.79476
3	21.15707
4	17.52185
5	15.26232
6	13.71331

In [89]:

```
plot(xgb.log, main= 'RMSE vs Iteration',xlab = 'Iteration', ylab = 'RMSE')
```



- It can be observed from the above plot that the RMSE reduces drastically from the first iteration to about the 20th. After this, there is a slow decrease and the RMSE reaches a point after which the change is insignificant.

In [91]:

```
# Test the model on test data
test_pred <- predict(xgb.mod, newdata = xgbtest)

# Minimum RMSE
min_rmse <- min(xgb.log$test_rmse)
print(paste('Lowest RMSE with XGBoost :',xgb.log[xgb.log$test_rmse == min_rmse
,'test_rmse']))

# Checking R-Squared Value
rsq_XGB <- cor(test.target, test_pred)^2
print(paste('R-Squared for XGB :',rsq_XGB))
```

```
[1] "Lowest RMSE with XGBoost : 9.662138"
[1] "R-Squared for XGB : 0.919962613273422"
```

4. Model Comparsion

Broadly, models can be compared by using certain tests of statistics. Or evaluated based on their **R squared** values. For the models that have been generated above, we will be using the Rsquared and RMSE values that have been calculated based on the predictions made by these models. Furthermore, the significance values or p-values of the coefficients can also be assessed in order to compare.

10 fold cross validation for 1st model:

In [92]:

```
# Define train control for k fold cross validation
train_control <- trainControl(method="cv", number=10)
# Fit Linear Regression with critical temp and the 'trains' dataset which cont
ains all the extracted regressors
model1 <- train(critical_temp~., data=trains, trControl=train_control, method=
"lm")
# Summarise Results
print(model1)
```

```
Linear Regression

17011 samples
   18 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 15310, 15310, 15309, 15309, 15311, 15310,
...
Resampling results:

  RMSE      Rsquared   MAE
  20.70998  0.6352009  16.20647

Tuning parameter 'intercept' was held constant at a value of TRUE
```

10 fold cross validation for 2nd model:

```
In [93]:
```

```
# Define train control for k fold cross validation
train_control <- trainControl(method="cv", number=10)
# Fit Linear Regression with critical temp and the 'trainstep' dataset which c
ontains all the extracted regressors
model2 <- train(critical_temp~., data=trainstep, trControl=train_control, meth
od="lm")
# Summarise Results
print(model2)
```

```
Linear Regression

17011 samples
   20 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 15310, 15309, 15309, 15309, 15311, 15310,
...
Resampling results:

  RMSE      Rsquared   MAE
  18.85998  0.6974165  14.43609

Tuning parameter 'intercept' was held constant at a value of TRUE
```

10 fold cross validation for 2nd model:

In [94]:

```
xgbcv <- xgb.cv( params = params, data = xgbtrain, nrounds = 100, showsd= T,
                 nfold = 10, print.every.n = 10)
print(paste('R-Squared for XGB :',rsq_XGB))
```

```
Warning message:
“'print.every.n' is deprecated.
Use 'print_every_n' instead.
See help("Deprecated") and help("xgboost-deprecated").”

[1]	train-rmse:35.488320+0.058075	test-rmse:35.545168+0.3795
46
[11]	train-rmse:10.093161+0.083885	test-rmse:11.534892+0.5059
18
[21]	train-rmse:8.765811+0.116767	test-rmse:10.771117+0.4860
24
[31]	train-rmse:8.089108+0.147080	test-rmse:10.474384+0.4946
50
[41]	train-rmse:7.551393+0.128546	test-rmse:10.282844+0.4876
25
[51]	train-rmse:7.141851+0.116996	test-rmse:10.162015+0.5115
47
[61]	train-rmse:6.825651+0.108048	test-rmse:10.060993+0.5056
91
[71]	train-rmse:6.562148+0.101006	test-rmse:9.994928+0.51549
6
[81]	train-rmse:6.309626+0.073465	test-rmse:9.942285+0.51502
4
[91]	train-rmse:6.092490+0.064584	test-rmse:9.910442+0.51161
8
[100]	train-rmse:5.929964+0.059036	test-rmse:9.867889+0.50787
4
[1] "R-Squared for XGB : 0.919962613273422"
```

- After cross validation on all the three models we can see that:
- Model 1 has an R squared of 0.635 and an RMSE of 20.71.
- Model 2 has an R squared of 0.699 and an RMSE of 18.79.
- And Model 3, has the lowest test RMSE and an R squared of 0.91, making it the model that BEST predicts the critical temperature of a semi conductor.

5. Variable Identification and Explanation

NOTE:

Since the models have been developed with highly independent feature selection mechanisms for each model respectively, I believe that the reason to explain the identified variables doesn't exist in this case. Meaning, step by step feature selection was NOT DONE to develop and enhance a single model. Each model has its own feature selection mehods involded.

Furthermore, reasoning for feature selection/removal has been provided in the model development stage for every model and thus, this section becomes redundant.

6. Conclusion

"Among the machine learning methods used in practice, gradient tree boosting is one technique that shines in many applications. Tree boosting has been shown to give state-of-the-art results on many standard classification benchmarks" (Li, 2010)

As we have seen above, the RMSE and out-of-sample R squared that have been obtained by implementing the XGBoost method have had phenomenal in comparison to the stepwise linear regression models. Furthermore, the XGBoost model has also eliminated the effect of multicollinearity that has been found in the predictors which has had a drastic affect on the previously developed models, thereby restricting their R squared values.

Thus, XGBoost has yet again proved to be one of the best Machine Learning model building methods.

7. References

1. Hamidieh, K. (2018), “A data-driven statistical model for predicting the critical temperature of a superconductor”.
2. Chris Chatfield (1986), European Journal of Operational Research 23 (1986) 5-13 North-Holland.
3. Reitermanova, Z. (2010), “Data splitting”, WDS’s 10 Proceedings of Contributed Papers, Part, pp. 31-36.
4. Härdle W.K., Simar L. (2012), Regression Models. In: Applied Multivariate Statistical Analysis. Springer, Berlin, Heidelberg
5. Max Kuhn (2008), "Building Predictive Models in R Using the caret Package", Journal of Statistical Software, November 2008, Volume 28, Issue 5.
6. M. Karagiannopoulos, D. Anyfantis, S. B. Kotsiantis, and P. E. Pintelas, "Feature Selection for Regression Problems" http://www.math.upatras.gr/ (http://www.math.upatras.gr/)˜dany/ Downloads/hercma07.pdf. (last accessed: 2015-Oct-22).
7. Beginners Tutorial on XGBoost and Parameter Tuning in R, https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/ (https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/)
8. Chandrashekar G., Sahin F. (2013), "A survey on feature selection methods", www.elsevier.com/locate/compeleceng.
9. Li, P. (2010). Robust Logitboost and adaptive base class (ABC) Logitboost. In Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI’10), pages 302–311.

In [ ]:

```

```

In [ ]:

```

```