FIT5149 S2 2019 Assessment 1:

Predicting the Critical Temperature of a Superconductor

Student information

- Family Name: Shibo
- Given Name: Li
- Student ID: 29874874
- Student email: slii0085@student.monash.edu

Programming Language: R 3.6.1 in Jupyter Notebook

R Libraries used:

- ggplot2: package for plotting
- reshape2: package for reshape dataframe
- h2o, carte: package for modelling and report the model information
  - foreach
- lattice: plotting heatmean
- grid
- gridExtra
- RColorBrewer

In [1]:

```
library(lattice)
library(plyr)
library(foreach)
library(dplyr)
library(ggplot2)
library(reshape2)
library(h2o)
library(caret)
```

```
Attaching package: ‘dplyr’

The following objects are masked from ‘package:plyr’:

    arrange, count, desc, failwith, id, mutate, rename, summarise,
    summarize

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union


----------------------------------------------------------------------
--

Your next step is to start H2O:
    > h2o.init()

For H2O package documentation, ask for help:
    > ??h2o

After starting H2O, you can use the Web UI at http://localhost:54321
For more information visit http://docs.h2o.ai

----------------------------------------------------------------------
--


Attaching package: ‘h2o’

The following objects are masked from ‘package:stats’:

    cor, sd, var

The following objects are masked from ‘package:base’:

    %*%, %in%, &&, apply, as.factor, as.numeric, colnames, colnames<
-,
    ifelse, is.character, is.factor, is.numeric, log, log10, log1p,
    log2, round, signif, trunc, ||
```

In [2]:

```
h2o.init()
```

```
H2O is not running yet, starting it now...

Note:  In case of errors look at the following log files:
    /var/folders/q2/qh_3lww505q8cqzg48cb6q900000gn/T//RtmpUFVIie/h2o
_lishibo_started_from_r.out
    /var/folders/q2/qh_3lww505q8cqzg48cb6q900000gn/T//RtmpUFVIie/h2o
_lishibo_started_from_r.err


Starting H2O JVM and connecting: . Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         1 seconds 640 milliseconds
    H2O cluster timezone:       Australia/Melbourne
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.26.0.2
    H2O cluster version age:    1 month and 19 days
    H2O cluster name:           H2O_started_from_R_lishibo_byj730
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.56 GB
    H2O cluster total cores:    12
    H2O cluster allowed cores:  12
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, C
ore V3, Core V4
    R Version:                  R version 3.6.1 (2019-07-05)
```

Table of Contents

1. Introduction

This report talks about the predicting models performed on a set of **Critical Temperature of Superconductor** to predict the critical temperature based on statistical model. The target of the report is to build 2 or 3 models from `train.csv` data set, which describe properties of `Superconductor` to predict the sales **Critical Temperature**.

`Data Exploration` section each attribute (variable) in the data, calculating some parameters to understand the relationship between variables, especially the **correlation** between variables. Considering there are too many features, examining the correlations between the variables through visual analysis will be employted.

- Pearson Correlation: $r(x, y) = \frac{Cov(x,y)}{\sqrt{(Var[x]*Var[y])}}$

However, the precondition for the `Pearson correlation coefficient` is that the variables satisfy the approximate `normal distribution` . This requires a `normality test` before calculating the correlation coefficient. Moreover, in most cases the variables do not necessarily satisfy the normal distribution, which makes it `impossible` to use the Pearson correlation coefficient.

The `Model Development` section shows three return models. In this section, by presenting the contribution of the process and parameters of the build model to the model, it presents the process from the initial model to the final model and prepares for the next step of `Model Comparison` .

`Model Comparison` section will compare the model mentioned in the previous section. The main comparison will focus on:

- The RMSE or R^2
  - RMSE = $\sqrt{\frac{\sum(y_{predict}-y_{actual})^2}{n}}$
  - $R^2 = 1 - \frac{\sum(y_{predict}-y_{actual})^2}{\sum(y_{actual}-y_{mean})^2}$

The section of `Variable Identification and Explanation` will discuss the contribution of the selected variables to the predicted values, and combined with relevant papers to explain the reasons.

`Conclusion` section will summarize the report.

The data sets in the report are:

1. 'train.csv' data set is from the *Superconducting Material Database* maintained by **JNIMS**, with 21263 observations and 82 columns, which cover 8 properties and 1 for elements numbers and the other for the *critical temperature*.
2. 'unique_m.csv' tells the chemical formula of each materials.

2. Data Exploration

In [3]:

```
## Read the data set into environment
ct.data <- read.csv('train.csv')
## Read 'unique_m.csv'
unique <- read.csv('unique_m.csv')
#data <- rbind(ct.data, unique)

dim(ct.data)
```

21263 82

2.1.1 Overview the 'train.csv' data set

In [4]:

```
# Display the dimensions
cat("The dataset has", dim(ct.data)[1], "observations, each with", dim(ct.data)
[2],
    "columns. The structure is:\n\n")

# Display the structure
str(ct.data)

cat("\nThe first few and last few records in the dataset are:")
# Inspect the first few records
head(ct.data)
# And the last few
tail(ct.data)

cat("\nBasic statistics for each attribute are:")
# Statistical summary
summary(ct.data)
```

```
The dataset has 21263 observations, each with 82 columns. The struct
ure is:

'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 1.1
29 ...
 $ 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 0.8
05 ...
 $ range_fie                      : num  811 811 811 811 811 ...
 $ wtd_range_fie                  : num  736 743 743 740 729 ...
 $ std_fie                        : num  324 290 324 324 324 ...
 $ wtd_std_fie                    : num  356 355 355 355 356 ...
 $ mean_atomic_radius             : num  160 161 160 160 160 ...
 $ wtd_mean_atomic_radius         : num  106 105 105 105 106 ...
 $ gmean_atomic_radius            : num  136 141 136 136 136 ...
 $ wtd_gmean_atomic_radius        : num  84.5 84.4 84.2 84.4 84.8
...
 $ entropy_atomic_radius          : num  1.26 1.51 1.26 1.26 1.26
...
 $ wtd_entropy_atomic_radius      : num  1.21 1.2 1.13 1.17 1.26 ...
 $ range_atomic_radius            : int  205 205 205 205 205 205 205
171 171 171 ...
 $ wtd_range_atomic_radius        : num  42.9 50.6 49.3 46.1 36.5
...
 $ std_atomic_radius              : num  75.2 67.3 75.2 75.2 75.2
...
 $ wtd_std_atomic_radius          : num  69.2 68 67.8 68.5 70.6 ...
 $ mean_Density                   : num  4654 5821 4654 4654 4654
...
 $ wtd_mean_Density               : num  2962 3021 2999 2980 2924
...
 $ gmean_Density                  : num  725 1237 725 725 725 ...
 $ wtd_gmean_Density              : num  53.5 54.1 54 53.8 53.1 ...
 $ entropy_Density                : num  1.03 1.31 1.03 1.03 1.03
...
 $ wtd_entropy_Density            : num  0.815 0.915 0.76 0.789 0.86
...
 $ range_Density                  : num  8959 10489 8959 8959 8959
```

```
...
 $ wtd_range_Density              : num  1580 1667 1667 1623 1492
...
 $ std_Density                    : num  3306 3767 3306 3306 3306
...
 $ wtd_std_Density                : num  3573 3633 3592 3582 3553
...
 $ mean_ElectronAffinity          : num  81.8 90.9 81.8 81.8 81.8
...
 $ wtd_mean_ElectronAffinity      : num  112 112 112 112 111 ...
 $ gmean_ElectronAffinity         : num  60.1 69.8 60.1 60.1 60.1
...
 $ wtd_gmean_ElectronAffinity     : num  99.4 101.2 101.1 100.2 97.8
...
 $ entropy_ElectronAffinity       : num  1.16 1.43 1.16 1.16 1.16
...
 $ wtd_entropy_ElectronAffinity   : num  0.787 0.839 0.786 0.787 0.7
87 ...
 $ 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 1.0
45 ...
 $ 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 0.62
5 ...
 $ entropy_ThermalConductivity    : num  0.308 0.847 0.308 0.308 0.3
08 ...
 $ wtd_entropy_ThermalConductivity: num  0.263 0.568 0.25 0.257 0.27
3 ...
 $ range_ThermalConductivity      : num  400 430 400 400 400 ...
 $ wtd_range_ThermalConductivity  : num  57.1 51.4 57.1 57.1 57.1
...
 $ std_ThermalConductivity        : num  169 199 169 169 169 ...
 $ wtd_std_ThermalConductivity    : num  139 140 139 139 138 ...
 $ mean_Valence                   : num  2.25 2 2.25 2.25 2.25 2.25
2.25 2.25 2.25 2.25 ...
 $ wtd_mean_Valence               : num  2.26 2.26 2.27 2.26 2.24
...
 $ gmean_Valence                  : num  2.21 1.89 2.21 2.21 2.21
...
 $ wtd_gmean_Valence              : num  2.22 2.21 2.23 2.23 2.21
```

```
...
 $ entropy_Valence                : num  1.37 1.56 1.37 1.37 1.37
...
 $ wtd_entropy_Valence            : num  1.07 1.05 1.03 1.05 1.1 ...
 $ range_Valence                  : int  1 2 1 1 1 1 1 1 1 ...
 $ 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 0.4
33 ...
 $ wtd_std_Valence                : num  0.437 0.469 0.445 0.441 0.4
29 ...
 $ critical_temp                  : num  29 26 19 22 23 23 11 33 36
31 ...
```

The first few and last few records in the dataset are:

A data.frame: 6 × 82

number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_g
<int>	<dbl>	<dbl>	<dbl>	
4	88.94447	57.86269	66.36159	
5	92.72921	58.51842	73.13279	
4	88.94447	57.88524	66.36159	
4	88.94447	57.87397	66.36159	
4	88.94447	57.84014	66.36159	
4	88.94447	57.79504	66.36159	

A data.frame: 6 × 82

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass
	<int>	<dbl>	<dbl>	<dbl>
21258	3	89.38983	89.38983	63.69471
21259	4	106.95788	53.09577	82.51538
21260	5	92.26674	49.02137	64.81266
21261	2	99.66319	95.60910	99.43388
21262	2	99.66319	97.09560	99.43388
21263	3	87.46833	86.85850	82.55576

Basic statistics for each attribute are:

```
 number_of_elements mean_atomic_mass  wtd_mean_atomic_mass gmean_ato
mic_mass
 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.:
58.041
 Median :4.000      Median : 84.923   Median : 60.697      Median :
66.362
 Mean   :4.115      Mean   : 87.558   Mean   : 72.988      Mean   :
71.291
 3rd Qu.:5.000      3rd Qu.:100.404   3rd Qu.: 86.104      3rd Qu.:
78.117
 Max.   :9.000      Max.   :208.980   Max.   :208.980      Max.   :2
08.980
 wtd_gmean_atomic_mass entropy_atomic_mass wtd_entropy_atomic_mass
 Min.   :  1.961       Min.   :0.0000      Min.   :0.0000
 1st Qu.: 35.249       1st Qu.:0.9667      1st Qu.:0.7754
 Median : 39.918       Median :1.1995      Median :1.1468
 Mean   : 58.540       Mean   :1.1656      Mean   :1.0639
 3rd Qu.: 73.113       3rd Qu.:1.4445      3rd Qu.:1.3594
 Max.   :208.980       Max.   :1.9838      Max.   :1.9582
 range_atomic_mass wtd_range_atomic_mass std_atomic_mass  wtd_std_at
omic_mass
 Min.   :  0.00    Min.   :  0.00        Min.   :  0.00   Min.   :
0.00
 1st Qu.: 78.51    1st Qu.: 16.82        1st Qu.: 32.89   1st Qu.: 2
8.54
 Median :122.91    Median : 26.64        Median : 45.12   Median : 4
4.29
 Mean   :115.60    Mean   : 33.23        Mean   : 44.39   Mean   : 4
1.45
 3rd Qu.:154.12    3rd Qu.: 38.36        3rd Qu.: 59.32   3rd Qu.: 5
3.63
 Max.   :207.97    Max.   :205.59        Max.   :101.02   Max.   :10
1.02
    mean_fie       wtd_mean_fie      gmean_fie      wtd_gmean_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
  entropy_fie    wtd_entropy_fie    range_fie      wtd_range_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       wtd_std_fie     mean_atomic_radius wtd_mean_atomic
_radius
 Min.   :  0.0   Min.   :  0.00   Min.   : 48.0      Min.   : 48.0
 1st Qu.:114.1   1st Qu.: 92.99   1st Qu.:149.3      1st Qu.:112.1
 Median :266.4   Median :258.45   Median :160.2      Median :126.0
 Mean   :215.6   Mean   :224.05   Mean   :158.0      Mean   :134.7
 3rd Qu.:297.7   3rd Qu.:342.66   3rd Qu.:169.9      3rd Qu.:158.3
 Max.   :499.7   Max.   :479.16   Max.   :298.0      Max.   :298.0
 gmean_atomic_radius wtd_gmean_atomic_radius entropy_atomic_radius
 Min.   : 48.0       Min.   : 48.00          Min.   :0.000
 1st Qu.:133.5       1st Qu.: 89.21          1st Qu.:1.066
 Median :142.8       Median :113.18          Median :1.331
```

```
 Mean   :144.4       Mean   :120.99          Mean   :1.268
 3rd Qu.:155.9       3rd Qu.:150.99          3rd Qu.:1.512
 Max.   :298.0       Max.   :298.00          Max.   :2.142
 wtd_entropy_atomic_radius range_atomic_radius wtd_range_atomic_radi
us
 Min.   :0.0000            Min.   :  0.0       Min.   :  0.00
 1st Qu.:0.8522            1st Qu.: 80.0       1st Qu.: 28.60
 Median :1.2429            Median :171.0       Median : 43.00
 Mean   :1.1311            Mean   :139.3       Mean   : 51.37
 3rd Qu.:1.4257            3rd Qu.:205.0       3rd Qu.: 60.22
 Max.   :1.9037            Max.   :256.0       Max.   :240.16
 std_atomic_radius wtd_std_atomic_radius  mean_Density
 Min.   :  0.00    Min.   : 0.00         Min.   :    1.429
 1st Qu.: 35.11    1st Qu.:32.02         1st Qu.: 4513.500
 Median : 58.66    Median :59.93         Median : 5329.086
 Mean   : 51.60    Mean   :52.34         Mean   : 6111.465
 3rd Qu.: 69.42    3rd Qu.:73.78         3rd Qu.: 6728.000
 Max.   :115.50    Max.   :97.14         Max.   :22590.000
 wtd_mean_Density    gmean_Density       wtd_gmean_Density   entropy
_Density
 Min.   :    1.429   Min.   :    1.429   Min.   :    0.686   Min.
:0.000
 1st Qu.: 2999.158   1st Qu.:  883.117   1st Qu.:   66.747   1st Q
u.:0.914
 Median : 4303.422   Median : 1339.975   Median : 1515.365   Median
:1.091
 Mean   : 5267.189   Mean   : 3460.692   Mean   : 3117.241   Mean
:1.072
 3rd Qu.: 6416.333   3rd Qu.: 5794.965   3rd Qu.: 5766.015   3rd Q
u.:1.324
 Max.   :22590.000   Max.   :22590.000   Max.   :22590.000   Max.
:1.954
 wtd_entropy_Density range_Density   wtd_range_Density  std_Density
 Min.   :0.0000      Min.   :    0   Min.   :    0     Min.   :    0
 1st Qu.:0.6887      1st Qu.: 6648   1st Qu.: 1657     1st Qu.: 2819
 Median :0.8827      Median : 8959   Median : 2083     Median : 3302
 Mean   :0.8560      Mean   : 8665   Mean   : 2903     Mean   : 3417
 3rd Qu.:1.0809      3rd Qu.: 9779   3rd Qu.: 3409     3rd Qu.: 4004
 Max.   :1.7034      Max.   :22589   Max.   :22434     Max.   :10724
 wtd_std_Density mean_ElectronAffinity wtd_mean_ElectronAffinity
 Min.   :    0   Min.   :  1.50        Min.   :  1.50
 1st Qu.: 2564   1st Qu.: 62.09        1st Qu.: 73.35
 Median : 3626   Median : 73.10        Median :102.86
 Mean   : 3319   Mean   : 76.88        Mean   : 92.72
 3rd Qu.: 3959   3rd Qu.: 85.50        3rd Qu.:110.74
 Max.   :10411   Max.   :326.10        Max.   :326.10
 gmean_ElectronAffinity wtd_gmean_ElectronAffinity entropy_ElectronA
ffinity
 Min.   :  1.50         Min.   :  1.50             Min.   :0.0000
 1st Qu.: 33.70         1st Qu.: 50.77             1st Qu.:0.8906
 Median : 51.47         Median : 73.17             Median :1.1383
 Mean   : 54.36         Mean   : 72.42             Mean   :1.0702
 3rd Qu.: 67.51         3rd Qu.: 89.98             3rd Qu.:1.3459
 Max.   :326.10         Max.   :326.10             Max.   :1.7677
 wtd_entropy_ElectronAffinity range_ElectronAffinity wtd_range_Elect
ronAffinity
 Min.   :0.0000               Min.   :  0.0          Min.   :  0.00
 1st Qu.:0.6607               1st Qu.: 86.7          1st Qu.: 34.04
 Median :0.7812               Median :127.0          Median : 71.16
 Mean   :0.7708               Mean   :120.7          Mean   : 59.33
 3rd Qu.:0.8775               3rd Qu.:138.6          3rd Qu.: 76.71
```

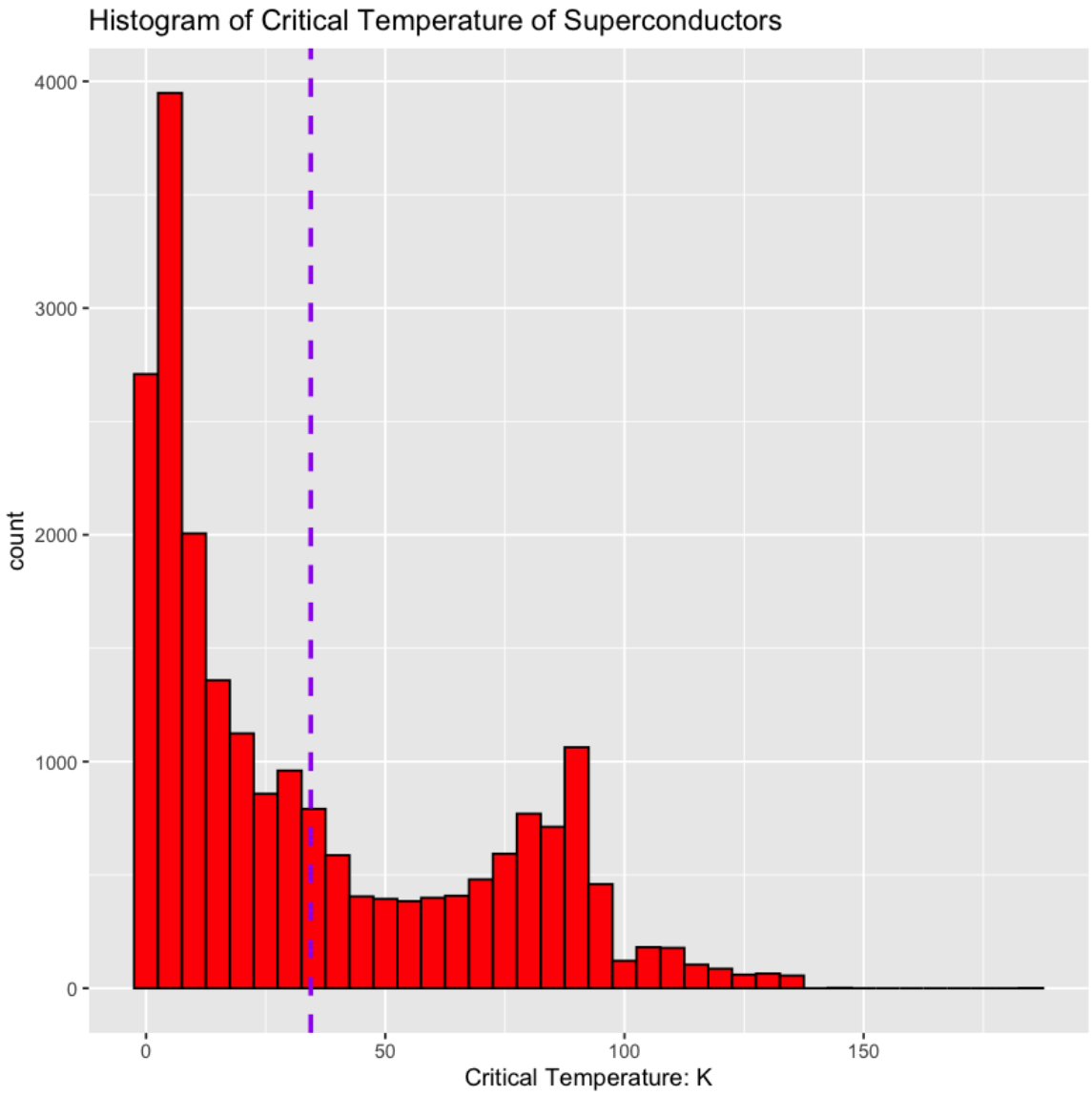
```
 Max.   :1.6754               Max.   :349.0          Max.   :218.70
 std_ElectronAffinity wtd_std_ElectronAffinity mean_FusionHeat
 Min.   :  0.00       Min.   :  0.00           Min.   :  0.222
 1st Qu.: 38.37       1st Qu.: 33.44           1st Qu.:  7.589
 Median : 51.13       Median : 48.03           Median :  9.304
 Mean   : 48.91       Mean   : 44.41           Mean   : 14.296
 3rd Qu.: 56.22       3rd Qu.: 53.32           3rd Qu.: 17.114
 Max.   :162.90       Max.   :169.08           Max.   :105.000
 wtd_mean_FusionHeat gmean_FusionHeat  wtd_gmean_FusionHeat entropy_
FusionHeat
 Min.   :  0.222     Min.   :  0.222   Min.   :  0.222      Min.   :
0.0000
 1st Qu.:  5.033     1st Qu.:  4.110   1st Qu.:  1.322      1st Qu.:
0.8333
 Median :  8.331     Median :  5.253   Median :  4.930      Median :
1.1121
 Mean   : 13.848     Mean   : 10.137   Mean   : 10.141      Mean   :
1.0933
 3rd Qu.: 18.514     3rd Qu.: 13.600   3rd Qu.: 16.429      3rd Qu.:
1.3781
 Max.   :105.000     Max.   :105.000   Max.   :105.000      Max.   :
2.0344
 wtd_entropy_FusionHeat range_FusionHeat wtd_range_FusionHeat std_Fu
sionHeat
 Min.   :0.0000         Min.   :  0.00   Min.   :  0.000      Min.
: 0.000
 1st Qu.:0.6727         1st Qu.: 12.88   1st Qu.:  2.329      1st Q
u.: 4.261
 Median :0.9950         Median : 12.88   Median :  3.436      Median
: 4.948
 Mean   :0.9141         Mean   : 21.14   Mean   :  8.219      Mean
: 8.323
 3rd Qu.:1.1574         3rd Qu.: 23.20   3rd Qu.: 10.499      3rd Q
u.: 9.041
 Max.   :1.7472         Max.   :104.78   Max.   :102.675      Max.
:51.635
 wtd_std_FusionHeat mean_ThermalConductivity wtd_mean_ThermalConduct
ivity
 Min.   : 0.000     Min.   :  0.0266         Min.   :  0.0266
 1st Qu.: 4.603     1st Qu.: 61.0000         1st Qu.: 54.1810
 Median : 5.501     Median : 96.5044         Median : 73.3333
 Mean   : 7.718     Mean   : 89.7069         Mean   : 81.5491
 3rd Qu.: 8.018     3rd Qu.:111.0053         3rd Qu.: 99.0629
 Max.   :51.680     Max.   :332.5000         Max.   :406.9600
 gmean_ThermalConductivity wtd_gmean_ThermalConductivity
 Min.   :  0.0266          Min.   :  0.023
 1st Qu.:  8.3398          1st Qu.:  1.087
 Median : 14.2876          Median :  6.096
 Mean   : 29.8417          Mean   : 27.308
 3rd Qu.: 42.3713          3rd Qu.: 47.308
 Max.   :317.8836          Max.   :376.033
 entropy_ThermalConductivity wtd_entropy_ThermalConductivity
 Min.   :0.0000              Min.   :0.0000
 1st Qu.:0.4578              1st Qu.:0.2507
 Median :0.7387              Median :0.5458
 Mean   :0.7276              Mean   :0.5400
 3rd Qu.:0.9622              3rd Qu.:0.7774
 Max.   :1.6340              Max.   :1.6130
 range_ThermalConductivity wtd_range_ThermalConductivity
 Min.   :  0.00            Min.   :  0.00
 1st Qu.: 86.38            1st Qu.: 29.35
```

```
 Median :399.80            Median : 56.56
 Mean   :250.89            Mean   : 62.03
 3rd Qu.:399.97            3rd Qu.: 91.87
 Max.   :429.97            Max.   :401.44
 std_ThermalConductivity wtd_std_ThermalConductivity  mean_Valence
 Min.   :  0.00          Min.   :  0.00              Min.   :1.000
 1st Qu.: 37.93          1st Qu.: 31.99              1st Qu.:2.333
 Median :135.76          Median :113.56              Median :2.833
 Mean   : 98.94          Mean   : 96.23              Mean   :3.198
 3rd Qu.:153.81          3rd Qu.:162.71              3rd Qu.:4.000
 Max.   :214.99          Max.   :213.30              Max.   :7.000
 wtd_mean_Valence gmean_Valence   wtd_gmean_Valence entropy_Valence
 Min.   :1.000    Min.   :1.000   Min.   :1.000     Min.   :0.000
 1st Qu.:2.117    1st Qu.:2.280   1st Qu.:2.091     1st Qu.:1.061
 Median :2.618    Median :2.615   Median :2.434     Median :1.369
 Mean   :3.153    Mean   :3.057   Mean   :3.056     Mean   :1.296
 3rd Qu.:4.026    3rd Qu.:3.728   3rd Qu.:3.915     3rd Qu.:1.589
 Max.   :7.000    Max.   :7.000   Max.   :7.000     Max.   :2.142
 wtd_entropy_Valence range_Valence   wtd_range_Valence  std_Valence
 Min.   :0.0000      Min.   :0.000   Min.   :0.0000    Min.   :0.000
0
 1st Qu.:0.7757      1st Qu.:1.000   1st Qu.:0.9215    1st Qu.:0.451
8
 Median :1.1665      Median :2.000   Median :1.0631    Median :0.800
0
 Mean   :1.0528      Mean   :2.041   Mean   :1.4830    Mean   :0.839
3
 3rd Qu.:1.3308      3rd Qu.:3.000   3rd Qu.:1.9184    3rd Qu.:1.200
0
 Max.   :1.9497      Max.   :6.000   Max.   :6.9922    Max.   :3.000
0
 wtd_std_Valence  critical_temp
 Min.   :0.0000   Min.   :  0.00021
 1st Qu.:0.3069   1st Qu.:  5.36500
 Median :0.5000   Median : 20.00000
 Mean   :0.6740   Mean   : 34.42122
 3rd Qu.:1.0204   3rd Qu.: 63.00000
 Max.   :3.0000   Max.   :185.00000
```

2.1.2 Plot the Distrubution of Critical Temperature T_c

In [5]:

```
## Plotting
ggplot(ct.data)+
geom_histogram(aes(x=critical_temp), stat = 'bin', binwidth = 5,
               show.legend = FALSE, color = 'black', fill = 'red')+
xlab('Critical Temperature: K')+
ggtitle('Histogram of Critical Temperature of Superconductors')+
geom_vline(aes(xintercept=mean(critical_temp)), color = 'purple', linetype = 'da
shed',
           size = 1,)
```



Hence, based on the graph above, it showes that the critical temperature of superconductors are very low, around 10K-15K, and few of whose are higher than 100K, although there is a peak at 80K.

Besides, the purple dash-line shows the average of critical temperature of superconductors in the dataset, which is around 30K or more.

2.2.1 Summary of data set

The 'train.csv' data set related information:

1. There are 21263 observations (rows) and 82 variables (columns).
2. The first variable is **number of elements**, which is not a property of chemical compound [1].
3. The last variable is the **critical temperature**, the target value.
4. There are 8 properties of chemical compounds recorded in the data set:
    - Atomic Mass
    - FIE(First Ionization Energy)
    - Atomic Radius
    - Density
    - Electron Affinity
    - Fusion Heat
    - Thermal Conductivity
    - Valence
5. There are 10 statistics for each properties:
    - Mean
    - Weighted mean
    - Geometric mean
    - Weighted geometric mean
    - Entropy
    - Weighted entropy
    - Range
    - Weighted range
    - Standard deviation
    - Weighted standard deviation.
6. All data in different columns are **Nummerical** data. Except the following columns are in **integer** class, rest of columns are in **double**:

    - number_of_elements
    - range_atomic_radius
    - range_Valence

2.2.2 Train set and test set

In [6]:

```
## Shuffle the original data set
len <- dim(ct.data)[1]
train.index <- sample(1:len, len, replace = FALSE)
ct.data <- ct.data[train.index,]
## Split into training data and testing data
train.data <- ct.data[1:(0.75*len),]
test.data <- ct.data[-(1:(0.75*len)),]
```

In general, the focus of the evaluation model is to divide the data into **`three sets`** : **training sets**, **validation sets**, and **test sets**.

The model is **trained on the training data** and the model is **evaluated on the validation data**. Once the best parameters are found, the last test is done on the test data.

The reason is that it is always necessary to adjust the model configuration when developing the model. This adjustment process requires the use of the model's performance on the verification data as a feedback. This adjustment process is essentially a learning: finding a good model configuration in a parameter space. Therefore, if you adjust the model configuration based on the performance of the model on the validation set, it will quickly cause the model to overfit on the validation set, even if not training the model directly on the validation set.

The key to this phenomenon is the information leak. Each time the model hyperparameter is adjusted based on the performance of the model on the validation set, some information about the validation data is leaked into the model. If only one parameter is adjusted , there is little information leaked, and the verification set can still evaluate the model reliably. But if we repeat this process multiple times, more and more information about the validation set will be leaked into the model.

Finally, the model is very good on the validation set (man-made), because that's what optimization contributes. What to care about is the performance of the model on new data, not the performance of the validation data, so it is needed to evaluate the model using a completely different, unprecedented set of data, which is the test set. The model must not be able to read any information related to the test set, even if it is indirect. If the model is adjusted based on test set performance, the measure of generalization ability is not accurate.

Hence, considering the limited observations of the original data set, I will use K-folds validations in every model below in section 3.

2.2.3 Feature Evaluation and Selection

A feature of something is **`an interesting or important part or characteristic`** of it.

The essence of `feature selection` is that the superiority of a given subset of features is measured by a specific `evaluation criterion` . Redundant features and irrelevant features in the original feature set are removed by feature selection. Useful features are preserved.

The main reasons for using feature selection are as follows:

- Faster model training speed
- Lower model complexity and better interpretability
- Higher precision (selected features)
- Weakened overfitting

Data standarization and **Data Division**

Here, I split the original data set into `Train.data` for training the model and `Test.data` for testing model, then I scale the train.data to train a model and test on scaled test.data respectively.

The reason why I do that is because if I scale the original data and then split, the test.data will have same information as the train.data, like maximum and minimum. It is not good for modeling for the test.data may impact the model.

In [7]:

```
## Dataframe the train.data, and scale it.
train.data <- data.frame(scale(train.data))
```

Whether the feature diverge:

If a feature does not `diverge` , for example, the **`variance`** is close to zero, that is, the sample has no difference in this feature, and this feature is not useful for distinguishing samples.

The reason why we need to remove the **`features with zero variance`** is because in some condition, the data generated some features with `a single value` (such as `zero-variance feature variables`). Hence, if it is, this may damage the model or instability of the data fit.

Ways to identify **`features with zero variance`** :

The maximum frequency value overs the second frequency value is called `frequency ratio` . The value is close to 1 for the balanced characteristic variable, but is very large for the data which is not balance.

The **unique value ratio** is the number of unique values divided by the total number of samples and multiplied by 100, which is close to zero as the data granularity increases.

If the frequency ratio is greater than a threshold and the **unique values ratio** is less than a threshold, we can consider this characteristic variable to be approximately zero variance.

In [8]:

```
## Find the features with zero variance
variance.data <- nearZeroVar(train.data[,-ncol(train.data)], saveMetrics = FALSE
, allowParallel = TRUE)
variance.data
```

68

In [9]:

```
## Remove the feature
train.data_1 <- train.data[, -variance.data]
```

In [10]:

```
dim(train.data_1)
```

15947 81

The correlations:

Correlation means how do the **`features`** to other **`features`** or **`targets`** related **linearly**.

Here is the fomular of correlation:

$$r(x, y) = \frac{Cov(x_1, x_2)}{\sqrt{(Var[x_1] * Var[x_2])}}$$

For we had normalized the data, the correlation coefficient is *Pearson Correlation Coefficient*. By doing this, it may reduce the multicollinearity between features.

However, correlations coefficient is only on **`linear relations`** , so it may omit those features with **`non-linear relations`** .

For we had `scale()` the training data, the we propose all the features in train.data follows the normal distribution.

In [11]:

```
names <- as.matrix(names(train.data_1))
```

In [12]:

```
head(names)
```

A matrix: 6 × 1 of type chr

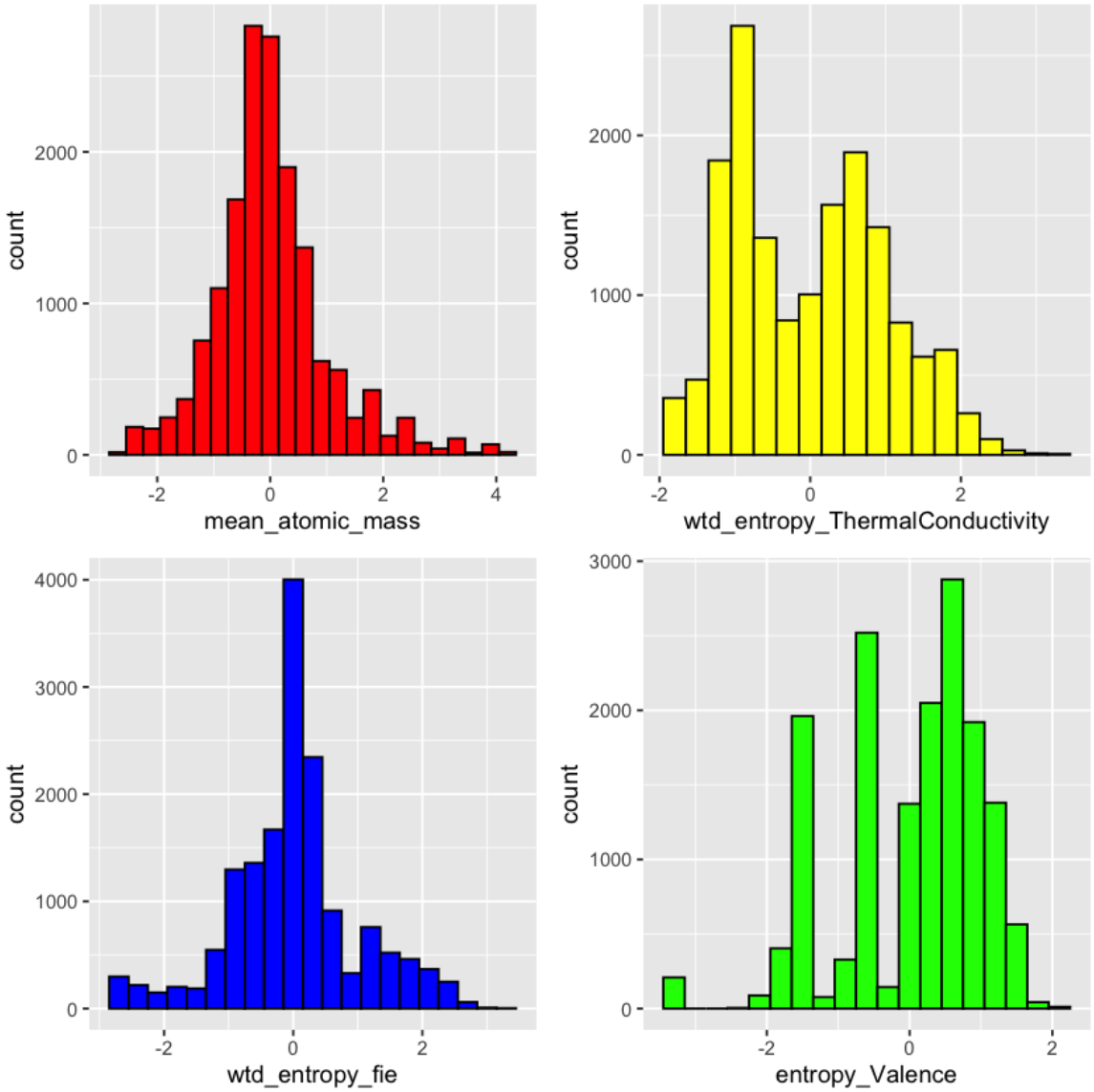
number_of_elements
mean_atomic_mass
wtd_mean_atomic_mass
gmean_atomic_mass
wtd_gmean_atomic_mass
entropy_atomic_mass

In [13]:

```
# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objec
ts)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

In [14]:

```
## Here, plotting 4 examples of features that whether the data follows the norma
l distribution or not even though
## scaled.
p1<- ggplot(train.data_1)+
     geom_histogram(aes(x=mean_atomic_mass), stat = 'bin', binwidth = 0.3,
                    show.legend = FALSE, color = 'black', fill = 'red')+ xlab('m
ean_atomic_mass')

p2<- ggplot(train.data_1)+
     geom_histogram(aes(x=wtd_entropy_fie), stat = 'bin', binwidth = 0.3,
                    show.legend = FALSE, color = 'black', fill = 'blue')+ xlab(
'wtd_entropy_fie')

p3<- ggplot(train.data_1)+
     geom_histogram(aes(x=wtd_entropy_ThermalConductivity), stat = 'bin', binwid
th = 0.3,
                    show.legend = FALSE, color = 'black', fill = 'yellow')+ xlab
('wtd_entropy_ThermalConductivity')

p4<- ggplot(train.data_1)+
     geom_histogram(aes(x=entropy_Valence), stat = 'bin', binwidth = 0.3,
                    show.legend = FALSE, color = 'black', fill = 'green')+ xlab(
'entropy_Valence')

multiplot(p1, p2, p3, p4, cols=2)
```



Hence, according to the graphs above, some features after scaling obey the normal distribution, but some are not.

Caret package provides an API: `findCorrelation()` .

Here, I create a correlation matrix, and use `findCorrelation()` to find features which are highly correlated, like greater than 0.5. According to the returned the index of highly correlated columns, they are moved from the *`train.data`* .

In [15]:

```
dim(train.data_1)
```

15947 81

In [16]:

```
## Create a correlation matrix, excluding the first and the last columns
cMatrix <-  cor(train.data_1[,2:80])
## Find the highly correlated columns, cutoff cor greater than 0.3
highlyCorre.index <- findCorrelation(cMatrix, cutoff = 0.3)
print(highlyCorre.index)
```

```
 [1] 20 17 26 27 30 19 74 34 75 15  6 25 33 29 73 71 69  5 24 56 70
72 68 45 12
[26] 55 32 22  4  7 14 35 54 16 53 31 63 36 10 64 47 50 77  2 52 46
49  3 38 23
[51]  8 37 58 40 48 11 42 43 67 61 66 65 39 21 13 59 76 60 57 79 44
```

In [17]:

```
## Features should be keep
as.matrix(names(train.data_1[,-highlyCorre.index]))
```

A matrix: 10 × 1 of type chr

number_of_elements
wtd_range_atomic_mass
range_fie
range_atomic_radius
wtd_std_Density
wtd_std_ElectronAffinity
mean_ThermalConductivity
wtd_range_Valence
wtd_std_Valence
critical_temp

Therefore, the API requires those columns (by index) above should be removed from the data set for they are always highly correlated to other columns.

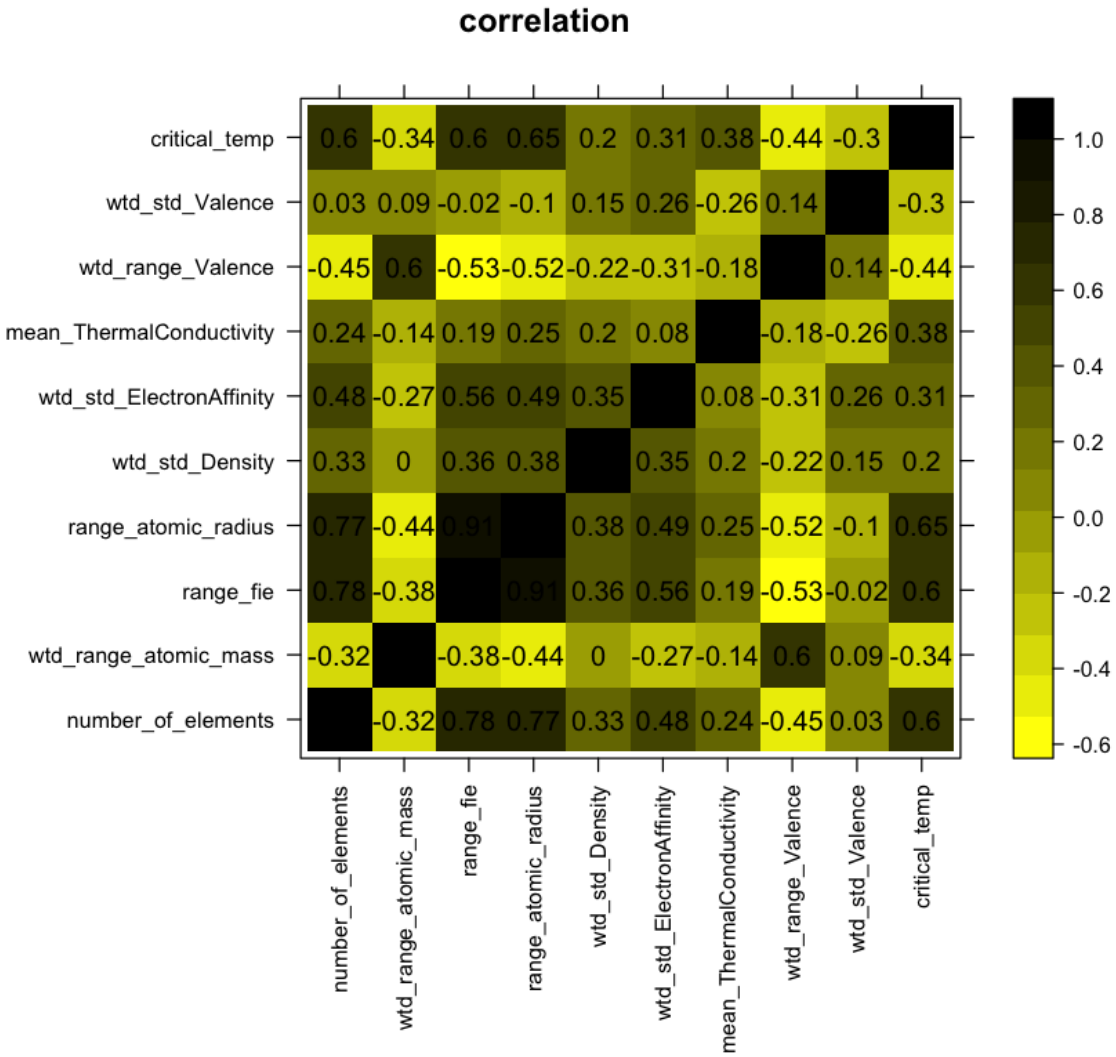
In [18]:

```
## Remove highly correlated columns
train.data.remove_highcor <- train.data_1[,-highlyCorre.index]
```

Correlation Coefficient of train.data

In [19]:

```
## Define the panel
myPanel <- function(x, y, z, ...) {
    panel.levelplot(x,y,z,...)
    panel.text(x, y, round(z, 2))
}
## Define the color scheme
cols = colorRampPalette(c("yellow","black"))
## Plot the correlation matrix.
levelplot(cor(train.data.remove_highcor), col.regions = cols(100), main = "corre
lation", xlab = NULL, ylab = NULL,
          scales = list(x = list(rot = 90)), panel = myPanel)
```



Based on the figure above:

- Most of the columns are not highly related any more, except `range of FIE` and `wtd mean FIT`.
- The left properties include:
  - First Ionization Energy
  - Density
  - Fusion Heat
  - Thermal Conductivity
  - Valence
- The removed properties:
  - Atomic Mass
  - Electron Affinity
  - Atomic Radius

The **`First Ionization Energy`** means:

- *The first ionization energy is the energy required for the **gaseous atom** in the ground state to **lose an electron** in the outermost layer. The **smaller** the first ionization energy, the **easier** it is for an atom to lose an electron; the **larger** the first ionization energy, the **harder** it is for an atom to lose an electron.* [1-1]

The **`Electron Affinity (Eea)`** means:

- *Electron affinity energy, also known as electron affinity, is the energy of affinity between electrons. The electron affinity is the ground state of the gaseous atom to **get the energy released by the electron** into a gaseous anion. The unit is kJ/mol (SI unit is J/mol).*[1-2]

`Valence` means:

- *Valence is a property of an atom of an element that is combined with the atomization of other elements. In general, the valence of the valence is equal to the **number of electrons lost** and lost in each atom at the time of compounding, that is, the amount of **electrons lost** and lost when the element reaches a stable structure, which is often determined by the electronic arrangement of the element, mainly the outermost electron. Arrangement, of course, may also involve a metastable structure composed of sub-layers that can be achieved by the secondary outer layer.* [1-3]

Generally, FIE is very highly correlated with Eea and atomic radius with chemical perspective. Being my Extended Chemistry background, it follows my expectation.

Also, valence, mostly, is determined by the atomic radius and the structure of outermost electron, which means valence is correlated with atomic radius.

The reason why the thermal conductivity is left is because the Thermal conduction relies on **electrons**, atoms, molecules, and lattice thermal motion in the material to transfer heat.

To summay, not only statistically but chemically, the lefted properties are in line with my expectations.

Granted that the features are not high `linear correlated` to other features, but the graph above shows that features also have little `linear correlated` to the target value **critial_temp**.

Therefore, I use Distributed Random Forest to find the importance of features.

Generally, how to evaluate the importance of features by `Random Forest` is based on the Gini:

$$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

A feature X_j's importance on a node m is:

$$VIM_{jm}^{(Gini)} = GI_m - GI_l - GI_r$$

GI_l and GI_r show the Gini in the new nodes.

In h2o, the importance of features is not evaluated by Gini. When calculating variable importances, H2O-3 looks at the squared error before and after the split using a particular variable. The difference is the improvement. H2O uses the improvement in squared error for each feature that was split on (rather than the accuracy). Each features improvement is then summed up at the end to get its total feature importance (and then scaled between 0-1).

2.2.4 Random Forest: Feature importance

Considering there are too many features, although we may reduce features by correlations, the method only based on correlations is not good enough. Hence, I try to employ **`Random Forest`** regression to figure out the most contributing ones to critical temperature.

`Random forest` belongs to `bagging` algorithm, and Bagging belongs to integrated learning method[2]. The general idea of integrated learning is to train multiple weak models to be packaged to form a strong model. The performance of strong model is much better than that of single weak model.

In the training phase, the random forest uses `bootstrap sampling` [3] to collect multiple different sub-set from the input training data set, and trains the data set to train multiple different decision trees in turn. In the forecasting phase, the random forest averages the predictions of multiple internal decision trees to get the final result. The implemented RFR (Random Forest Regression) has the following features:

- Model training (not the primary mission here)
- Model prediction (not the primary mission here)
- Calculate feature importance

After the h2o finishes Random Forest, we can go to Web UI[4] to have some visualized explorations.

However, **`Random Forest Regression`** here is only to do the **features selection**. Models will be on other methods.

In [20]:

```
## Transform dataframe to h2Oframe
train.data.h2o <-as.h2o(train.data_1[,-1])
```

```
  |======================================================================
=====| 100%
```

In [21]:

```
## Implement randomforest algorithm only on training set:
## parameters:
## 1. with nfolds, where n is 10
## 2. ntress, where n = 100
## 3. max_depth, the how deep a tree goes
## It may last few seconds, please be patient.
rd_model_1 <- h2o.randomForest(,'critical_temp',
                               train.data.h2o,
                               nfolds = 10,
                               ntrees = 50,
                               max_depth = 20,
                               model_id = 'rd_fit1')
```

```
  |======================================================================
=====| 100%
```

The importance of features

When calculating variable importances, H2O-s at the squared error before and after the split using a particular variable. The difference is the improvement. H2O uses the improvement in **`squared error`** for each feature that was split on (rather than the accuracy). Each features improvement is then summed up at the end to get its total feature importance (and then scaled between 0-1).

Also, Go to Web UI, in ***getModel*** (Get a list of models in H2O), find the RFR model named 'rd_model_1' and 'rd_model_1_cv_5', you may see the following graphs:

- graph 1: Number of trees v.s. percetage (blue is rd_model_1, and organge is rd_model_1_cv_5)
- graph 2: relative features

In [22]:

```
## Present the feature importance:
rd_model_1_varimp <- h2o.varimp(rd_model_1)
rd_model_1_varimp[1:10,]
```

A H2OTable: 10 × 4

variable	relative_importance	scaled_importance	percentage
<chr>	<dbl>	<dbl>	<dbl>
range_atomic_radius	157613.69	1.00000000	0.24699077
wtd_std_ThermalConductivity	75741.93	0.48055426	0.11869247
wtd_entropy_Valence	48854.11	0.30996106	0.07655752
std_ThermalConductivity	43069.65	0.27326086	0.06749291
wtd_mean_Valence	40190.47	0.25499352	0.06298104
wtd_gmean_Valence	38393.33	0.24359133	0.06016481
wtd_mean_ThermalConductivity	17876.60	0.11342033	0.02801377
entropy_Valence	15430.45	0.09790044	0.02418050
wtd_entropy_atomic_mass	14977.77	0.09502838	0.02347113
range_fie	10328.50	0.06553046	0.01618542

Selection

To filter the top 10 features:

```
{r}
model_varimp[1:10,1]
```

In [23]:

```
as.matrix(selected.feature <- rd_model_1_varimp[1:10,1])
```

A matrix: 10 × 1 of type chr

range_atomic_radius	
wtd_std_ThermalConductivity	
wtd_entropy_Valence	
std_ThermalConductivity	
wtd_mean_Valence	
wtd_gmean_Valence	
wtd_mean_ThermalConductivity	
entropy_Valence	
wtd_entropy_atomic_mass	
range_fie	

In [24]:

```
## Form a selected dataframe for modeling
train.data.selected <- train.data[,selected.feature]
```

In [25]:

```
dim(train.data.selected)
```

15947 10

In [26]:

```
train.label <- data.frame('critical_temp' = train.data[,ncol(train.data)])
```

In [27]:

```
model_1.train.data <- cbind(train.data.selected, train.label)
```

For test.data, employing:

- Scale
- Remove zero variance features
- Select features

In [28]:

```
## * Scale
## * Select features
test.data.selected <- scale(test.data)[,selected.feature]
test.label <- data.frame('critical_temp' = scale(test.data[,ncol(test.data)]))
```

In [29]:

```
model_1.test.data <- cbind(test.data.selected, test.label)
```

In [30]:

```
## h2o.frame model_1.train.data and model_1.test.data
model_1.train.data.h2o <- as.h2o(model_1.train.data)
model_1.test.data.h2o <- as.h2o(model_1.test.data)
```

```
  |======================================================================
=====| 100%
  |======================================================================
=====| 100%
```

3. Model Development

Develop Model

Deeplearning by `h2o.deeplearning()`

Activation and Loss Function In H2O, it supports activation function showing below:

- Tanh: $f(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}$, whose range belongs to $f(\cdot) \in [-1, 1]$.
- Recitified Linear: $f(\alpha) = max(0, \alpha)$, whose range is R_+.
- Maxout: $f(\alpha_1, \alpha_2) = max(\alpha_1, \alpha_2)$, whose range is R.

Also, we may determine any distribution functions below:

- Poisson
- Laplace
- Bernoulli
- Gamma
- Quantile
- Multinomial
- Tweedie
- Huber
- Gaussian

In my deeplearning model, the Loss function is AUTO.

Parallel Distributed Network Training

Minimizing the loss function $L(W, B|j)$ requires the use of SGD. The traditional SGD algorithm is very fast, but after parallelization, the speed of the whole algorithm is slowed down. H2O implements a architecture called **Hogwild!**, which is a A shared memory model, the core idea is to distribute the data to each node for **parallel execution**, and then the data of each node is redistributed to the multi-core of each node, and is executed asynchronously using multiple threads. Thereby improving efficiency.

Specifying the Number of Training Samples

H2O is extensible and can take full advantage of the compute node's ability to use the train samples per iteration parameter. If specified as -1, all nodes process their local data for each iteration. Set to -2, based on computing power and network overhead will automatically adjust the appropriate parameters, parameters will affect the convergence speed of the entire training process.

In [31]:

```
## Parameters:
## 1. train_sample_per_iteration: Number of training samples (globally) per MapR
educe iteration.
## 2. rho: Adaptive learning rate time decay factor.
## 3. Learning rate (higher => less stable, lower => slower convergence)
## 4. L2 regularization (can add stability and improve generalization, causes ma
ny weights to be small.
md_deep_1 <- h2o.deeplearning(, y = 'critical_temp',
                              model_id = 'md_deep_1',
                              nfolds = 10,
                              train_samples_per_iteration = -2,
                              rho = 0.9,
                              l2 = 0.009,
                              stopping_rounds = 0,
                              epochs = 20,
                              training_frame = model_1.train.data.h2o)
```

```
  |=================================================================
=====| 100%
```

In [32]:

```
## Show the details of model
md_deep_1
```

```
Model Details:
==============

H2ORegressionModel: deeplearning
Model ID:  md_deep_1
Status of Neuron Layers: predicting critical_temp, regression, gauss
ian distribution, Quadratic loss, 42,601 weights/biases, 508.5 KB, 3
18,940 training samples, mini-batch size 1
  layer units      type dropout       l1       l2 mean_rate rate_rms
momentum
1     1    10     Input  0.00 %       NA       NA        NA       NA
NA
2     2   200 Rectifier  0.00 % 0.000000 0.009000  0.047467 0.105891
0.000000
3     3   200 Rectifier  0.00 % 0.000000 0.009000  0.557574 0.403474
0.000000
4     4     1    Linear      NA 0.000000 0.009000  0.099523 0.166740
0.000000
  mean_weight weight_rms mean_bias bias_rms
1          NA         NA        NA       NA
2   -0.005214   0.049055 -0.003139 0.052965
3   -0.003336   0.024906 -0.009004 0.038525
4    0.035367   0.130919  0.044251 0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10088 samples *
*

MSE:  0.2206694
RMSE:  0.4697546
MAE:  0.3285501
RMSLE:  NaN
Mean Residual Deviance :  0.2206694



H2ORegressionMetrics: deeplearning
** Reported on cross-validation data. **
** 10-fold cross-validation on training data (Metrics computed for c
ombined holdout predictions) **

MSE:  0.2370971
RMSE:  0.4869262
MAE:  0.3368824
RMSLE:  NaN
Mean Residual Deviance :  0.2370971


Cross-Validation Metrics Summary:
                             mean           sd cv_1_valid cv_2_valid
cv_3_valid
mae                    0.33690444 0.0097405035  0.3368969  0.3447409
0.3282769
mean_residual_deviance 0.23710705 0.0112864915 0.23949192 0.23948304
0.2343062
mse                    0.23710705 0.0112864915 0.23949192 0.23948304
0.2343062
r2                     0.76282275  0.011793612  0.7598584 0.75402963
0.767251
```

```
residual_deviance      0.23710705 0.0112864915 0.23949192 0.23948304
0.2343062
rmse                   0.48666817  0.011426685 0.48937914 0.48937005
0.48405185
rmsle                         0.0          NaN         NaN         NaN
NaN
                       cv_4_valid cv_5_valid cv_6_valid cv_7_valid c
v_8_valid
mae                     0.3356948 0.31979206 0.33040774  0.3682961
0.32056838
mean_residual_deviance 0.22791983 0.22128884 0.22685745  0.2748777
0.21499881
mse                    0.22791983 0.22128884 0.22685745  0.2748777
0.21499881
r2                      0.7754216 0.78129476    0.77296  0.7197785
0.77266246
residual_deviance      0.22791983 0.22128884 0.22685745  0.2748777
0.21499881
rmse                    0.4774095 0.47041348 0.47629556  0.5242878
0.46367964
rmsle                         NaN         NaN         NaN         NaN
NaN
                       cv_9_valid cv_10_valid
mae                     0.3498378  0.33453268
mean_residual_deviance 0.24292384  0.24892291
mse                    0.24292384  0.24892291
r2                      0.7703438   0.7546273
residual_deviance      0.24292384  0.24892291
rmse                   0.49287304  0.49892175
rmsle                         NaN         NaN
```

In [33]:

```
## Retrieve Model Score History
h2o.scoreHistory(md_deep_1)
```

A H2OTable: 4 × 10

timestamp	duration	training_speed	epochs	iterations	samples	training_rmse	training_devia
<chr>	<chr>	<chr>	<dbl>	<int>	<dbl>	<dbl>	<(
2019-09-15 17:20:41	0.000 sec	NA	0	0	0	NaN	I
2019-09-15 17:20:43	1 min 22.469 sec	21477 obs/sec	2	1	31894	0.5851104	0.3423
2019-09-15 17:20:48	1 min 27.717 sec	43003 obs/sec	18	9	287046	0.4962794	0.2462
2019-09-15 17:20:49	1 min 28.225 sec	44794 obs/sec	20	10	318940	0.4697546	0.2206

In [34]:

```
## Show the Model Performance Metrics in H2O
md_deep_1_perf1 <- h2o.performance(model = md_deep_1,
                                   model_1.test.data.h2o)
md_deep_1_perf1
```

```
H2ORegressionMetrics: deeplearning

MSE:  0.2302527
RMSE:  0.4798466
MAE:  0.3386979
RMSLE:  NaN
Mean Residual Deviance :  0.2302527
```

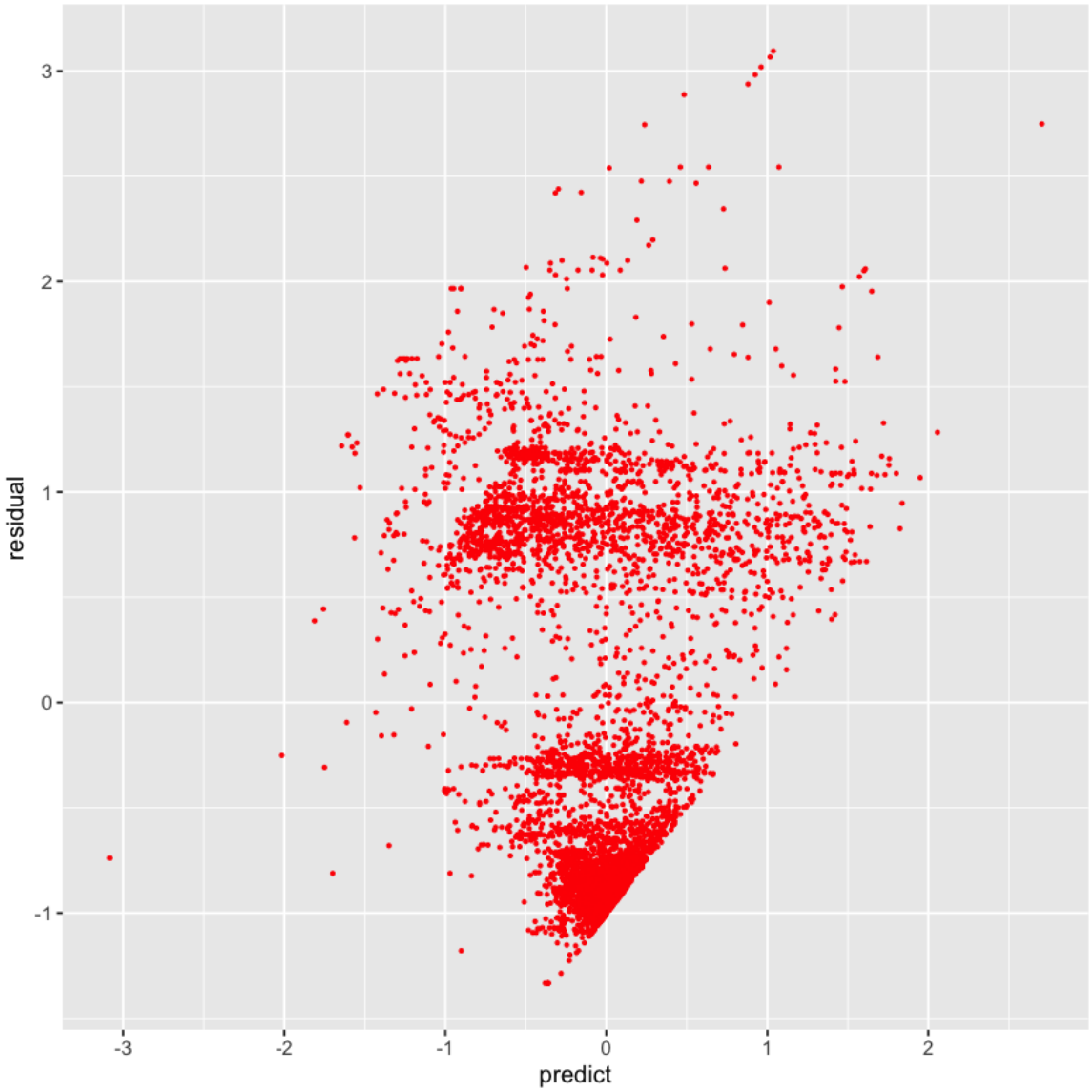
In [35]:

```
## This function is to plot the residual to prediction value
## :param model: the model to test data
## :return None
plot_residual <- function(model){
    ## Model to the test h2oframe
    predic <- as.data.frame(h2o.predict(model,model_1.test.data.h2o))
    ## the true value
    true.value <- as.data.frame(model_1.test.data.h2o[,ncol(model_1.test.data.h2
o)])
    ## The residual
    residual <- predic - true.value
    ## combine to a dataframe
    test <- cbind(predic,residual)
    ## Rename
    names(test) <- c('residual','predict')
    ## Plot the graph
    ggplot(test, aes(x=predict, y=residual)) + geom_point(color = 'red', size =
0.5)
}
```

In [36]:

```
plot_residual(md_deep_1)
```

```
  |======================================================================
=====| 100%
```



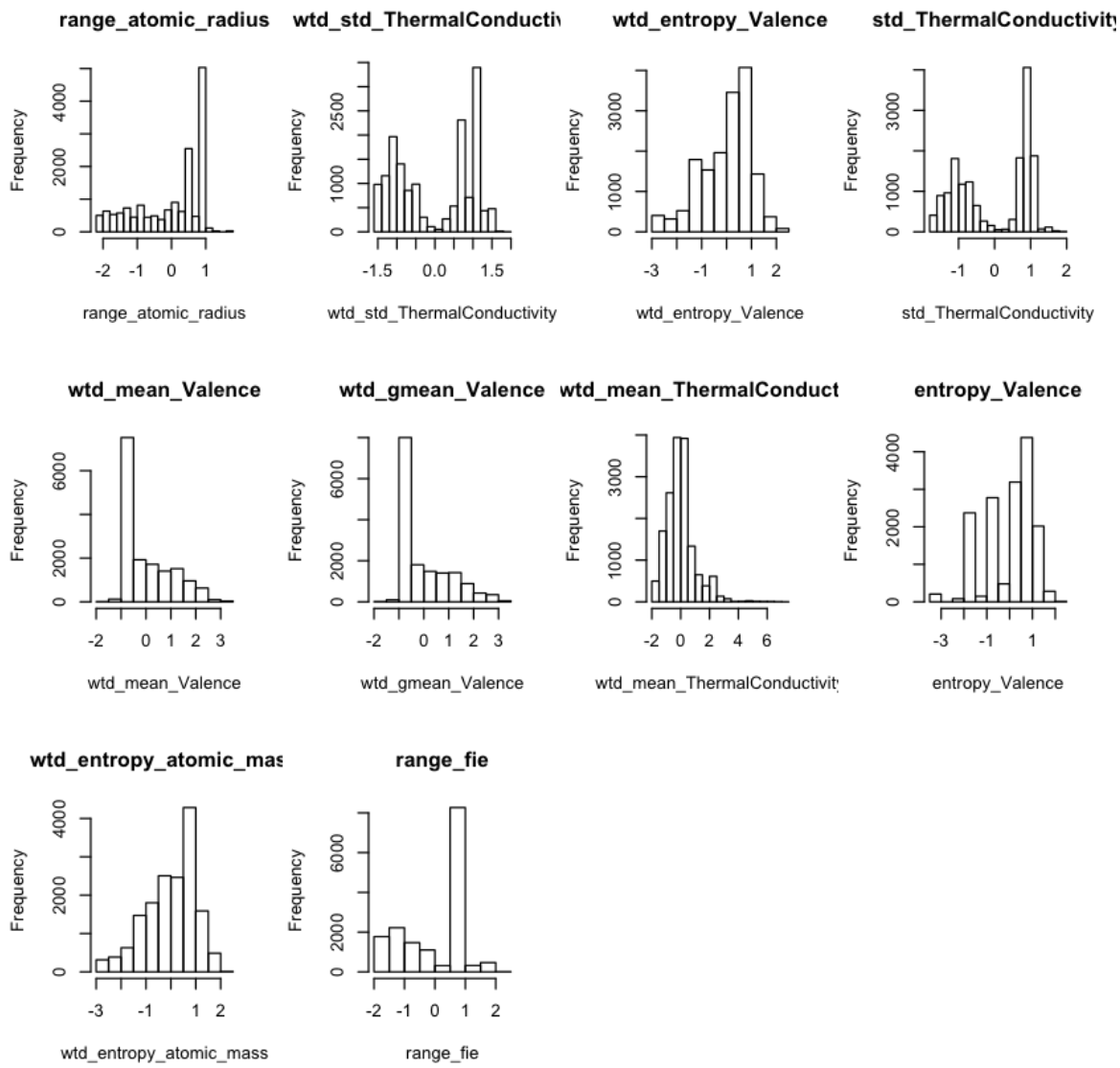
Generalized Linear Model(GLM)

Generalized Linear Model(GLM) by `h2o.glm()`

The linear model has very **`strong limitations`** : the response variable y must obey the **`Gaussian distribution`** ; the main limitation is that the scale of the fitted target y is a real number (−∞, +∞).

In [37]:

```
## Plotting the selected features
par(mfrow = c(3,4))
hist(model_1.train.data[,1],
     main = colnames(model_1.train.data)[1],
     xlab = colnames(model_1.train.data)[1])
hist(model_1.train.data[,2],
     main = colnames(model_1.train.data)[2],
     xlab = colnames(model_1.train.data)[2])
hist(model_1.train.data[,3],
     main = colnames(model_1.train.data)[3],
     xlab = colnames(model_1.train.data)[3])
hist(model_1.train.data[,4],
     main = colnames(model_1.train.data)[4],
     xlab = colnames(model_1.train.data)[4])
hist(model_1.train.data[,5],
     main = colnames(model_1.train.data)[5],
     xlab = colnames(model_1.train.data)[5])
hist(model_1.train.data[,6],
     main = colnames(model_1.train.data)[6],
     xlab = colnames(model_1.train.data)[6])
hist(model_1.train.data[,7],
     main = colnames(model_1.train.data)[7],
     xlab = colnames(model_1.train.data)[7])
hist(model_1.train.data[,8],
     main = colnames(model_1.train.data)[8],
     xlab = colnames(model_1.train.data)[8])
hist(model_1.train.data[,9],
     main = colnames(model_1.train.data)[9],
     xlab = colnames(model_1.train.data)[9])
hist(model_1.train.data[,10],
     main = colnames(model_1.train.data)[10],
     xlab = colnames(model_1.train.data)[10])
```



However, in statistics, the generalized linear model (GLM) is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution.

In [38]:

```
## Parameters
## 1. alpha: Distribution of regularization between the L1 (Lasso) and L2 (Ridg
e) penalties.
    ## A value of 1 for alpha represents Lasso regression,
    ## a value of 0 produces Ridge regression,
    ## and anything in between specifies the amount of mixing between the two.
## 2.lambda: Regularization strength
md_glm_1 <- h2o.glm(,
                    y = 'critical_temp',
                    training_frame = model_1.train.data.h2o,
                    model_id = 'md_glm_1',
                    alpha = 0.7,
                    lambda = 0.09,
                    nfolds = 10
                   )
```

```
  |======================================================================
=====| 100%
```

In [39]:

```
## Show the details of model
md_glm_1
```

```
Model Details:
==============

H2ORegressionModel: glm
Model ID:  md_glm_1
GLM Model: summary
    family     link                               regularization
1 gaussian identity Elastic Net (alpha = 0.7, lambda = 0.09 )
  number_of_predictors_total number_of_active_predictors number_of_i
terations
1                         10                            5
1
                 training_frame
1 model_1.train.data_sid_8188_3

Coefficients: glm coefficients
                          names coefficients standardized_coefficien
ts
1                     Intercept     0.000000                  0.0000
00
2           range_atomic_radius     0.126617                  0.1266
17
3   wtd_std_ThermalConductivity     0.360065                  0.3600
65
4           wtd_entropy_Valence     0.000000                  0.0000
00
5       std_ThermalConductivity     0.000000                  0.0000
00
6              wtd_mean_Valence    -0.103929                 -0.1039
29
7             wtd_gmean_Valence     0.000000                  0.0000
00
8  wtd_mean_ThermalConductivity     0.048343                  0.0483
43
9               entropy_Valence     0.000000                  0.0000
00
10      wtd_entropy_atomic_mass     0.137877                  0.1378
77
11                    range_fie     0.000000                  0.0000
00

H2ORegressionMetrics: glm
** Reported on training data. **

MSE:  0.4327534
RMSE:  0.65784
MAE:  0.5162462
RMSLE:  NaN
Mean Residual Deviance :  0.4327534
R^2 :  0.5672194
Null Deviance :15946
Null D.o.F. :15946
Residual Deviance :6901.119
Residual D.o.F. :15941
AIC :31912.62



H2ORegressionMetrics: glm
** Reported on cross-validation data. **
** 10-fold cross-validation on training data (Metrics computed for c
```

```
ombined holdout predictions) **

MSE:  0.4348913
RMSE:  0.6594629
MAE:  0.5172947
RMSLE:  NaN
Mean Residual Deviance :  0.4348913
R^2 :  0.5650814
Null Deviance :15948.4
Null D.o.F. :15946
Residual Deviance :6935.212
Residual D.o.F. :15941
AIC :31991.21


Cross-Validation Metrics Summary:
                            mean           sd cv_1_valid cv_2_valid
cv_3_valid
mae                    0.5172442 0.0046480894  0.5304925 0.50835735
0.5151319
mean_residual_deviance 0.4348826  0.007924237 0.46013242 0.42258134
0.43767387
mse                    0.4348826  0.007924237 0.46013242 0.42258134
0.43767387
null_deviance          1594.8398    54.524208   1685.732  1584.9005
1438.178
r2                     0.5645335  0.006146579 0.56104785  0.5701473
0.54899234
residual_deviance      693.52124    19.265842  738.97266   681.2011
647.7573
rmse                   0.6594022  0.005973038  0.6783306 0.65006256
0.66156924
rmsle                        0.0          NaN        NaN        NaN
NaN
                       cv_4_valid cv_5_valid cv_6_valid cv_7_valid c
v_8_valid
mae                      0.517373  0.5175584  0.5112707 0.51880217
0.5205527
mean_residual_deviance 0.42204392  0.4395014 0.43348753 0.42875376
0.43003425
mse                    0.42204392  0.4395014 0.43348753 0.42875376
0.43003425
null_deviance           1519.1681   1579.268  1555.3318  1653.4674
1672.2816
r2                     0.56423014  0.5536074  0.5632596 0.57134646
0.5817345
residual_deviance       661.34283  704.52075  679.27496     708.73
697.51556
rmse                    0.6496491   0.662949  0.6583977  0.6547929
0.65577
rmsle                         NaN        NaN        NaN        NaN
NaN
                       cv_9_valid cv_10_valid
mae                    0.50861865   0.5242847
mean_residual_deviance 0.42735255  0.44726497
mse                    0.42735255  0.44726497
null_deviance           1570.3986   1689.6726
r2                      0.5638686   0.5671006
residual_deviance        684.6188   731.27826
rmse                    0.6537221   0.6687787
rmsle                         NaN         NaN
```

In [40]:

```
## Retrieve Model Score History
h2o.scoreHistory(md_glm_1)
```

A H2OTable: 1 × 5

timestamp	duration	iterations	negative_log_likelihood	objective
<chr>	<chr>	<int>	<dbl>	<dbl>
2019-09-15 17:20:51	0.000 sec	0	15946	0.9999373

In [41]:

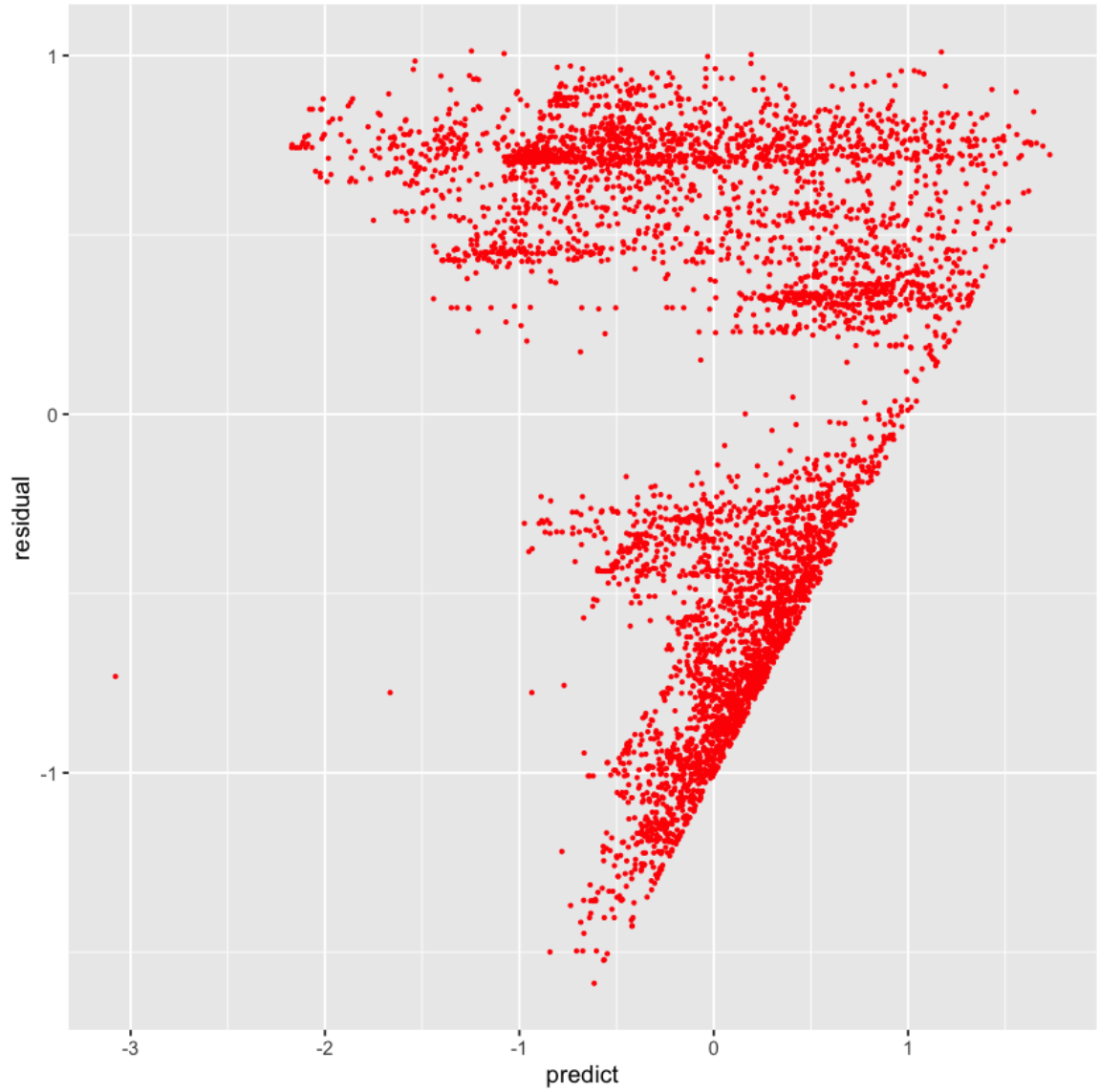
```
## Show the Model Performance Metrics in H2O
md_glm_1_perf1 <- h2o.performance(model = md_glm_1,model_1.test.data.h2o)
md_glm_1_perf1
```

```
H2ORegressionMetrics: glm

MSE:  0.4347313
RMSE:  0.6593415
MAE:  0.5189017
RMSLE:  NaN
Mean Residual Deviance :  0.4347313
R^2 :  0.565187
Null Deviance :5315
Null D.o.F. :5315
Residual Deviance :2311.031
Residual D.o.F. :5310
AIC :10671.78
```

In [42]:

```
plot_residual(md_glm_1)
```

```
  |======================================================================
=====| 100%
```



Xgboost

Xgboost by `h2o.xgboost()`

The idea of the `Xgboost` algorithm is to `constantly add trees` , constantly transforming features to grow a tree, and adding a tree each time is actually **`learning a new function to fit the residual of the last prediction`** . When we get the k tree in training, we need to predict the score of a sample.

In fact, according to the characteristics of this sample, we will fall into the corresponding leaf node in each tree. Each leaf node corresponds to a score, and finally only The score corresponding to each tree needs to be added to the predicted value of the sample.

$$\hat{y} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$$
$$where\ F = \{f(x) = w_{q(x)}\}\ (q : R^m \to T, w \in R^T)$$

The object function of `Xgboost` algorithm:

$$Obj = \overbrace{\sum_{i=1}^{n} l(y_i, \hat{y}_i)}^{Training\ loss} + \underbrace{\sum_{k=1}^{K} \Omega(f_x)}_{Complexity\ of\ the\ Tree}$$
$$where\ \Omega(f) = \gamma T + \frac{1}{2}\lambda \boldsymbol{w}^2$$

The Traiing loss section is to evaluate the difference between prediction and true value. Ω function is the regularization part, where T stands for the number of leaves and the w stands for the score it.

In [43]:

```
## Parameter
## nfolds: number of kv folds
## ntrees: the number of trees
## max_depth: maximum depth of a tree
## learn_rate: the learning rate
## reg_lambda: the regularization rate
md_xg_1 <- h2o.xgboost(, y = 'critical_temp',
                       training_frame = model_1.train.data.h2o,
                       model_id = 'md_xg_1',
                       nfolds = 10,
                       ntrees = 50,
                       max_depth = 10,
                       learn_rate = 0.09,
                       reg_lambda = 0.09)
```

```
  |======================================================================
=====| 100%
```

In [44]:

```
## Retrieve Model Score History
h2o.scoreHistory(md_xg_1)
```

A H2OTable: 44 × 6

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
2019-09-15 17:21:17	24.111 sec	0	1.1180059	1.0288310	1.24993729
2019-09-15 17:21:17	24.159 sec	1	1.0269312	0.9453514	1.05458763
2019-09-15 17:21:17	24.193 sec	2	0.9446233	0.8694771	0.89231309
2019-09-15 17:21:17	24.228 sec	3	0.8703676	0.8008545	0.75753981
2019-09-15 17:21:17	24.264 sec	4	0.8034622	0.7386334	0.64555144
2019-09-15 17:21:17	24.303 sec	5	0.7428288	0.6819254	0.55179458
2019-09-15 17:21:17	24.344 sec	6	0.6884503	0.6307142	0.47396384
2019-09-15 17:21:17	24.387 sec	7	0.6394499	0.5841971	0.40889619
2019-09-15 17:21:17	24.434 sec	8	0.5950256	0.5416728	0.35405543
2019-09-15 17:21:17	24.481 sec	9	0.5551774	0.5029089	0.30822195
2019-09-15 17:21:17	24.532 sec	10	0.5192397	0.4676011	0.26960990
2019-09-15 17:21:17	24.587 sec	11	0.4866644	0.4352279	0.23684219
2019-09-15 17:21:17	24.645 sec	12	0.4581658	0.4062749	0.20991588
2019-09-15 17:21:18	24.705 sec	13	0.4324742	0.3798043	0.18703394
2019-09-15 17:21:18	24.773 sec	14	0.4099187	0.3559725	0.16803335
2019-09-15 17:21:18	24.841 sec	15	0.3897173	0.3344392	0.15187958
2019-09-15 17:21:18	24.912 sec	16	0.3717470	0.3147436	0.13819584
2019-09-15 17:21:18	24.991 sec	17	0.3553360	0.2965973	0.12626368
2019-09-15 17:21:18	25.068 sec	18	0.3414584	0.2805144	0.11659384
2019-09-15 17:21:18	25.154 sec	19	0.3286951	0.2657477	0.10804044
2019-09-15 17:21:18	25.236 sec	20	0.3177580	0.2526733	0.10097012
2019-09-15 17:21:18	25.325 sec	21	0.3081422	0.2408812	0.09495161
2019-09-15 17:21:18	25.431 sec	22	0.2989741	0.2297272	0.08938550

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
2019-09-15 17:21:18	25.528 sec	23	0.2910278	0.2198814	0.08469717
2019-09-15 17:21:18	25.624 sec	24	0.2842418	0.2110117	0.08079341
2019-09-15 17:21:19	25.727 sec	25	0.2772288	0.2023850	0.07685579
2019-09-15 17:21:19	25.828 sec	26	0.2726395	0.1954467	0.07433230
2019-09-15 17:21:19	25.934 sec	27	0.2677597	0.1887438	0.07169525
2019-09-15 17:21:19	26.040 sec	28	0.2630666	0.1826300	0.06920404
2019-09-15 17:21:19	26.154 sec	29	0.2574785	0.1766388	0.06629519
2019-09-15 17:21:19	26.273 sec	30	0.2526370	0.1711996	0.06382547
2019-09-15 17:21:19	26.398 sec	31	0.2487292	0.1667003	0.06186624
2019-09-15 17:21:19	26.529 sec	32	0.2448313	0.1623453	0.05994237
2019-09-15 17:21:19	26.657 sec	33	0.2403819	0.1579599	0.05778348
2019-09-15 17:21:20	26.794 sec	34	0.2371660	0.1544307	0.05624770
2019-09-15 17:21:20	26.941 sec	35	0.2327200	0.1505018	0.05415861
2019-09-15 17:21:20	27.076 sec	36	0.2291800	0.1470450	0.05252346
2019-09-15 17:21:20	27.219 sec	37	0.2262505	0.1440315	0.05118927
2019-09-15 17:21:20	27.364 sec	38	0.2237833	0.1414653	0.05007895
2019-09-15 17:21:20	27.512 sec	39	0.2212642	0.1390210	0.04895783
2019-09-15 17:21:21	27.667 sec	40	0.2171473	0.1362677	0.04715296
2019-09-15 17:21:21	27.817 sec	41	0.2142183	0.1339347	0.04588947
2019-09-15 17:21:21	27.974 sec	42	0.2116132	0.1318825	0.04478015
2019-09-15 17:21:21	28.303 sec	50	0.1933236	0.1192155	0.03737402

In [45]:

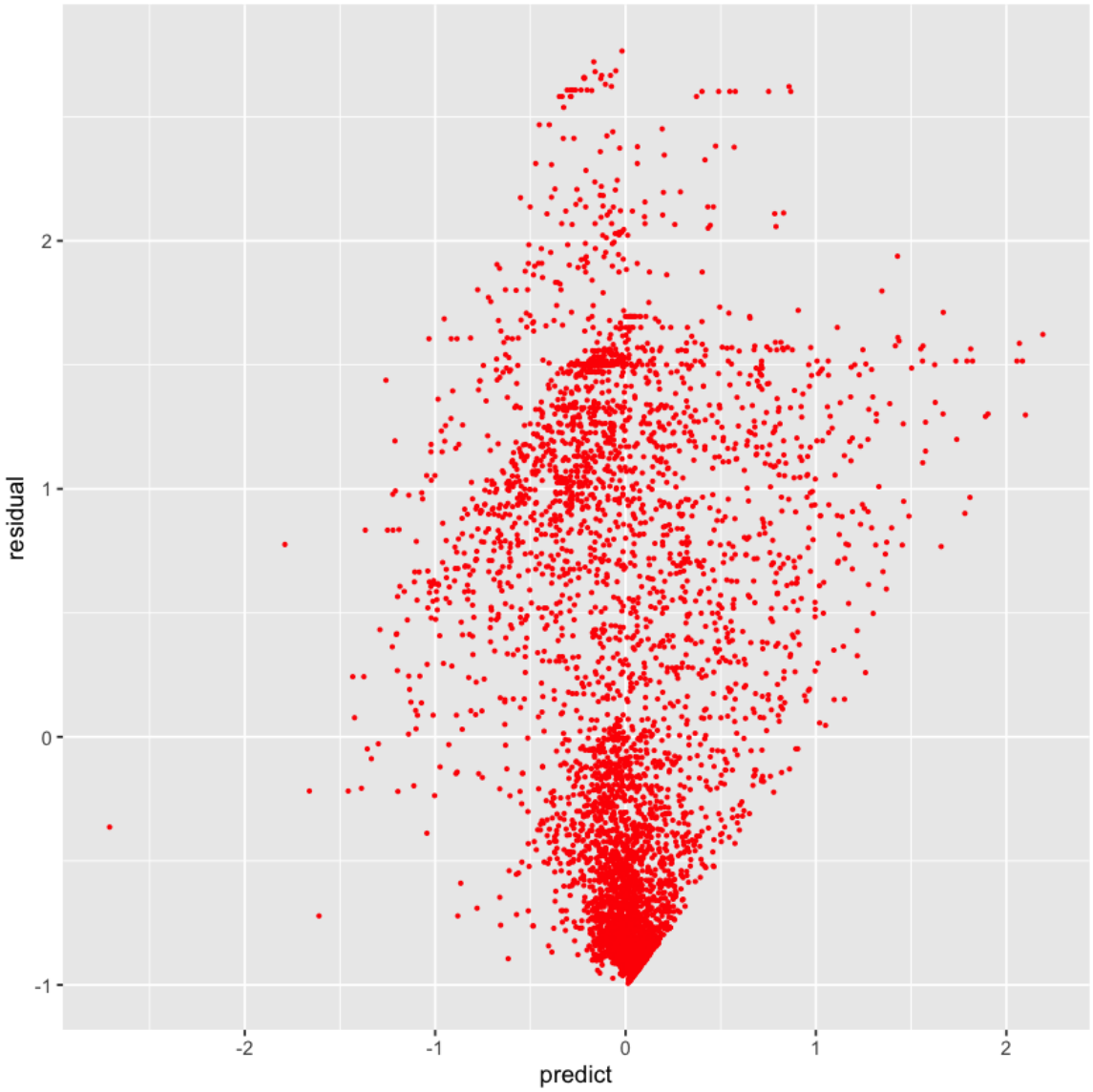
```
## Show the Model Performance Metrics in H2O
md_xg_1_perf1 <- h2o.performance(model = md_xg_1, model_1.test.data.h2o)
md_xg_1_perf1
```

```
H2ORegressionMetrics: xgboost

MSE:  0.1433412
RMSE:  0.3786043
MAE:  0.234011
RMSLE:  NaN
Mean Residual Deviance :  0.1433412
```

In [46]:

```
plot_residual(md_xg_1)
```

```
  |======================================================================
=====| 100%
```



AutoML

AutoML by `h2o.automl()`

`AutoML` is one of many tools in data science, and it can solve some of the tasks in data science tasks.

AutoML also includes two categories: `traditional AutoML` and `deep AutoML`.

Traditional AutoML is to solve the **traditional machine learning modeling problem**, which is oriented to traditional machine learning related algorithms, such as linear regression, logistic regression, decision tree and so on.

The **deep AutoML** is more oriented to the modeling of neural networks in deep learning.

AutoML encapsulates the process of constructing the network structure, adjusting the network structure, adjusting the hyperparameters, and evaluating the model through its own set of algorithms, all of which are automated. Structural adjustments and parameter adjustments that may have no purpose are changed into structurally ordered adjustments through scientific algorithms, which reduces the threshold for machine learning and shortens the entire modeling process.

NAS: Neural Architecture Search, neural network structure search, that is, the automatic generation of neural network structure needs to be realized through some structure and algorithm.

Hyper-parameter optimization: Hyper-parameter optimization, which automatically optimizes hyper-parameters in neural networks.

Meta-learning: meta-learning, or learning to learn.

By implementing the h2o.automl( ) algorithm, models will be trained and return the best one to users if calling:

```
{r}
model.leader <- model@leader
```

In my implementation, I set " `stackedensemble` = TRUE", which means I do not refuse to have stacked ensemble model[5].

Ensemble algorithm

The **`ensemble`** algorithm refers to combining **multiple models** for `better results` and making the ensembled model `more generalizable`.

For multiple models, how to combine these models is mainly in the following different ways:

- Find the best performing model as the final prediction model;
- Vote or average the prediction results of multiple models;
- Weighted average to the prediction results of multiple models.

The Ensemble method is a kind of supervised learning. After the training is completed, it can be regarded as a single "hypothesis" (or model), but the "hypothesis" is not necessarily in the original "hypothesis" space. Therefore, the Ensemble method has more flexibility.

Empirically, if the diversity between the models to be combined is significant, then Ensemble usually has a better result, so there are also many Ensemble methods dedicated to improving the differences between the models to be combined; it has been found that using multiple strong learning algorithms is more effective than models that are designed to promote diversity.

The following figure uses different subsets of the training set for training (to obtain appropriate variability, similar to reasonable sampling), to get different errors, and then combine them appropriately to reduce the error.

Stacking

Stacking is a layered model integration framework. Taking two layers as an example, the first layer is composed of multiple base learners, the input is the original training set, and the second layer model is

In [47]:

```
## Parameter
## 1. max_runtime_secs: Maximum allowed runtime in seconds for the entire model
 training process.
## 2. max_models: Maximum number of models to build in the AutoML process (does
 not include Stacked Ensembles).
md_aml_1 <- h2o.automl(y = 'critical_temp',
                       training_frame = model_1.train.data.h2o,
                       max_runtime_secs = 180,
                       max_models = 3,
                       project_name = 'md_aml_1')
```

```
  |======================================================================
=====| 100%
```

In [48]:

```
## Show the model details:
md_aml_1.leader <- md_aml_1@leader
md_aml_1.leader
```

```
Model Details:
==============

H2ORegressionModel: stackedensemble
Model ID:  StackedEnsemble_AllModels_AutoML_20190915_172124
NULL


H2ORegressionMetrics: stackedensemble
** Reported on training data. **

MSE:  0.0325049
RMSE:  0.1802911
MAE:  0.1070401
RMSLE:  NaN
Mean Residual Deviance :  0.0325049



H2ORegressionMetrics: stackedensemble
** Reported on cross-validation data. **
** 5-fold cross-validation on training data (Metrics computed for co
mbined holdout predictions) **

MSE:  0.09428794
RMSE:  0.3070634
MAE:  0.1800032
RMSLE:  NaN
Mean Residual Deviance :  0.09428794
```

Hence, I know that based on **AutoML**, h2o suggests me the stackedensemble regression is the best one.

In [49]:

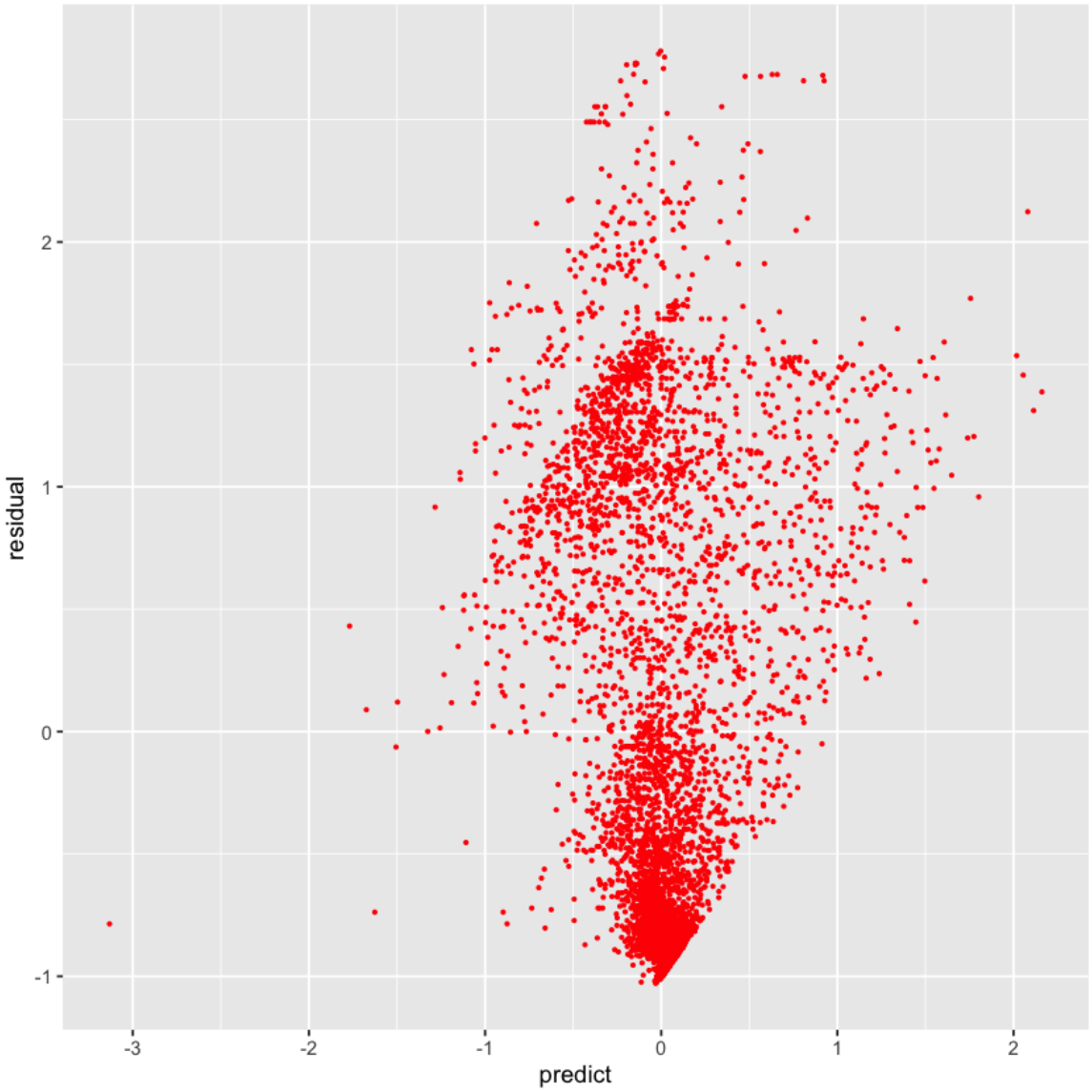
```
## The performance of the model
md_se_1 <- h2o.performance(md_aml_1.leader,model_1.test.data.h2o)
md_se_1
```

```
H2ORegressionMetrics: stackedensemble

MSE:  0.1316513
RMSE:  0.3628378
MAE:  0.2250091
RMSLE:  NaN
Mean Residual Deviance :  0.1316513
```

In [50]:

```
plot_residual(md_aml_1.leader)
```

```
  |======================================================================
=====| 100%
```



4. Model Comparison

In order to evaluate the quality of the model, several statistics will be used:

- RMSE = $\sqrt{\frac{\sum(y_{predict}-y_{actual})^2}{n}}$

In [51]:

```
## Showing the statistics for every model
c(md_se_1, md_xg_1_perf1, md_glm_1_perf1, md_deep_1_perf1)
```

```
[[1]]
H2ORegressionMetrics: stackedensemble

MSE:  0.1316513
RMSE:  0.3628378
MAE:  0.2250091
RMSLE:  NaN
Mean Residual Deviance :  0.1316513


[[2]]
H2ORegressionMetrics: xgboost

MSE:  0.1433412
RMSE:  0.3786043
MAE:  0.234011
RMSLE:  NaN
Mean Residual Deviance :  0.1433412


[[3]]
H2ORegressionMetrics: glm

MSE:  0.4347313
RMSE:  0.6593415
MAE:  0.5189017
RMSLE:  NaN
Mean Residual Deviance :  0.4347313
R^2 :  0.565187
Null Deviance :5315
Null D.o.F. :5315
Residual Deviance :2311.031
Residual D.o.F. :5310
AIC :10671.78


[[4]]
H2ORegressionMetrics: deeplearning

MSE:  0.2302527
RMSE:  0.4798466
MAE:  0.3386979
RMSLE:  NaN
Mean Residual Deviance :  0.2302527
```

Based on the above information, I found that after comparison, the performance of the model trained by stacked ensemble is the best because it has the lowest MSE. Although the accuracy is still not very good, it is already far better than the other three models. In comparison, GLM has the largest MSE, so GLM is not the main choice for my training model.

It is true that according to the residual graphs, the size of the residuals is distributed on the upper and lower sides of the x-axis, but it is not an average distribution, but a tendency to converge to the upper right. This means that in the initial features selection, the selected features did not help to train good models. Therefore, if you want to improve the accuracy of the model, the first thing to do is to re-select the features. After reselecting the features, adjust the training model parameters.

By comparison, I can draw a conclusion: GLM is a good regression prediction algorithm. But compared to xgboost and deeplearning, or the stacked ensemble method, the accuracy of GLM is slightly worse.

Re-select features

In [57]:

```
as.matrix(selected.feature2 <- rd_model_1_varimp[1:6,1])
train.data.selected2 <- train.data[,selected.feature2]
model_1.train.data2 <- cbind(train.data.selected2, train.label)
model_1.train.data2.h2o <- as.h2o(model_1.train.data2)
```

A matrix: 6 × 1 of type chr

range_atomic_radius
wtd_std_ThermalConductivity
wtd_entropy_Valence
std_ThermalConductivity
wtd_mean_Valence
wtd_gmean_Valence

```
  |======================================================================
=====| 100%
```

In [58]:

```
## Parameters:
## 1. train_sample_per_iteration: Number of training samples (globally) per MapR
educe iteration.
## 2. rho: Adaptive learning rate time decay factor.
## 3. Learning rate (higher => less stable, lower => slower convergence)
## 4. L2 regularization (can add stability and improve generalization, causes ma
ny weights to be small.
md_deep_2 <- h2o.deeplearning(, y = 'critical_temp',
                              model_id = 'md_deep_2',
                              nfolds = 10,
                              train_samples_per_iteration = -2,
                              rho = 0.9,
                              l2 = 0.009,
                              stopping_rounds = 0,
                              epochs = 20,
                              training_frame = model_1.train.data2.h2o)

## Parameters
## 1. alpha: Distribution of regularization between the L1 (Lasso) and L2 (Ridg
e) penalties.
    ## A value of 1 for alpha represents Lasso regression,
    ## a value of 0 produces Ridge regression,
    ## and anything in between specifies the amount of mixing between the two.
## 2.lambda: Regularization strength
md_glm_2 <- h2o.glm(,
                    y = 'critical_temp',
                    training_frame = model_1.train.data2.h2o,
                    model_id = 'md_glm_2',
                    alpha = 0.7,
                    lambda = 0.09,
                    nfolds = 10
                   )

## Parameter
## nfolds: number of kv folds
## ntrees: the number of trees
## max_depth: maximum depth of a tree
## learn_rate: the learning rate
## reg_lambda: the regularization rate
md_xg_2 <- h2o.xgboost(, y = 'critical_temp',
                       training_frame = model_1.train.data2.h2o,
                       model_id = 'md_xg_1',
                       nfolds = 10,
                       ntrees = 50,
                       max_depth = 10,
                       learn_rate = 0.09,
                       reg_lambda = 0.09)

## Parameter
## 1. max_runtime_secs: Maximum allowed runtime in seconds for the entire model
 training process.
## 2. max_models: Maximum number of models to build in the AutoML process (does
 not include Stacked Ensembles).
md_aml_2 <- h2o.automl(y = 'critical_temp',
                       training_frame = model_1.train.data2.h2o,
                       max_runtime_secs = 180,
                       max_models = 3,
                       project_name = 'md_aml_2')
```

```
## Show the model details:
md_aml_2.leader <- md_aml_2@leader
```

```
  |======================================================================
=====| 100%
  |======================================================================
=====| 100%
  |======================================================================
=====| 100%
  |======================================================================
=====| 100%
```

In [59]:

```
md_deep_2_perf2 <- h2o.performance(md_glm_2,model_1.test.data.h2o)
md_glm_2_perf2 <- h2o.performance(md_deep_2,model_1.test.data.h2o)
md_xg_2_perf2 <- h2o.performance(md_xg_2,model_1.test.data.h2o)
md_se_2 <- h2o.performance(md_aml_2.leader,model_1.test.data.h2o)
```

In [60]:

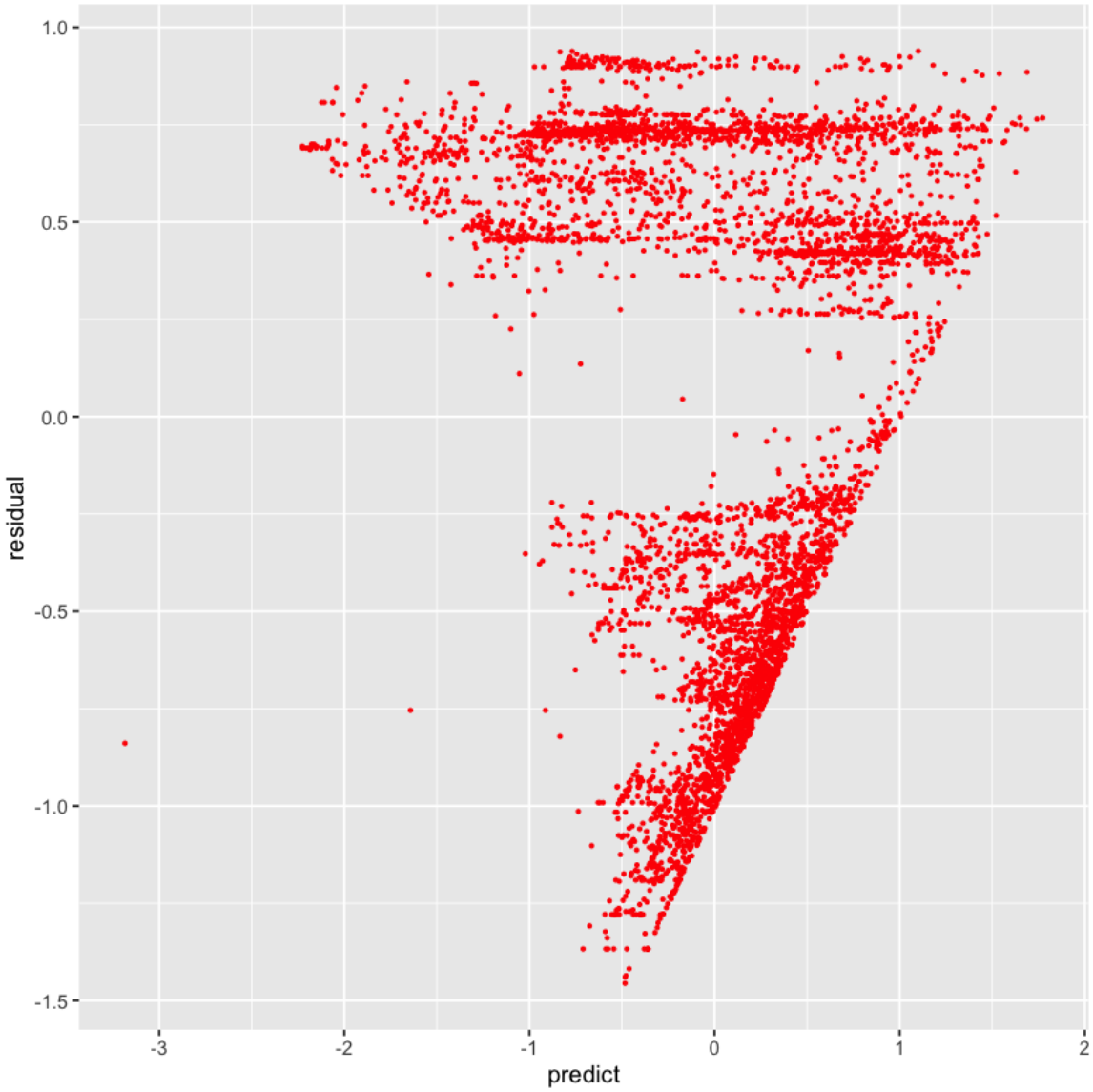
```
c(md_deep_2_perf2,md_glm_2_perf2,md_xg_2_perf2,md_se_2)
```

```
[[1]]
H2ORegressionMetrics: glm

MSE:  0.446489
RMSE:  0.6681983
MAE:  0.5233812
RMSLE:  NaN
Mean Residual Deviance :  0.446489
R^2 :  0.553427
Null Deviance :5315
Null D.o.F. :5315
Residual Deviance :2373.535
Residual D.o.F. :5311
AIC :10811.65


[[2]]
H2ORegressionMetrics: deeplearning

MSE:  0.2884317
RMSE:  0.5370584
MAE:  0.3825259
RMSLE:  NaN
Mean Residual Deviance :  0.2884317


[[3]]
H2ORegressionMetrics: xgboost

MSE:  0.1694698
RMSE:  0.4116672
MAE:  0.2524125
RMSLE:  NaN
Mean Residual Deviance :  0.1694698


[[4]]
H2ORegressionMetrics: stackedensemble

MSE:  0.6037581
RMSE:  0.7770187
MAE:  0.6950106
RMSLE:  NaN
Mean Residual Deviance :  0.6037581
```

In [67]:

```
plot_residual(md_glm_2)
```

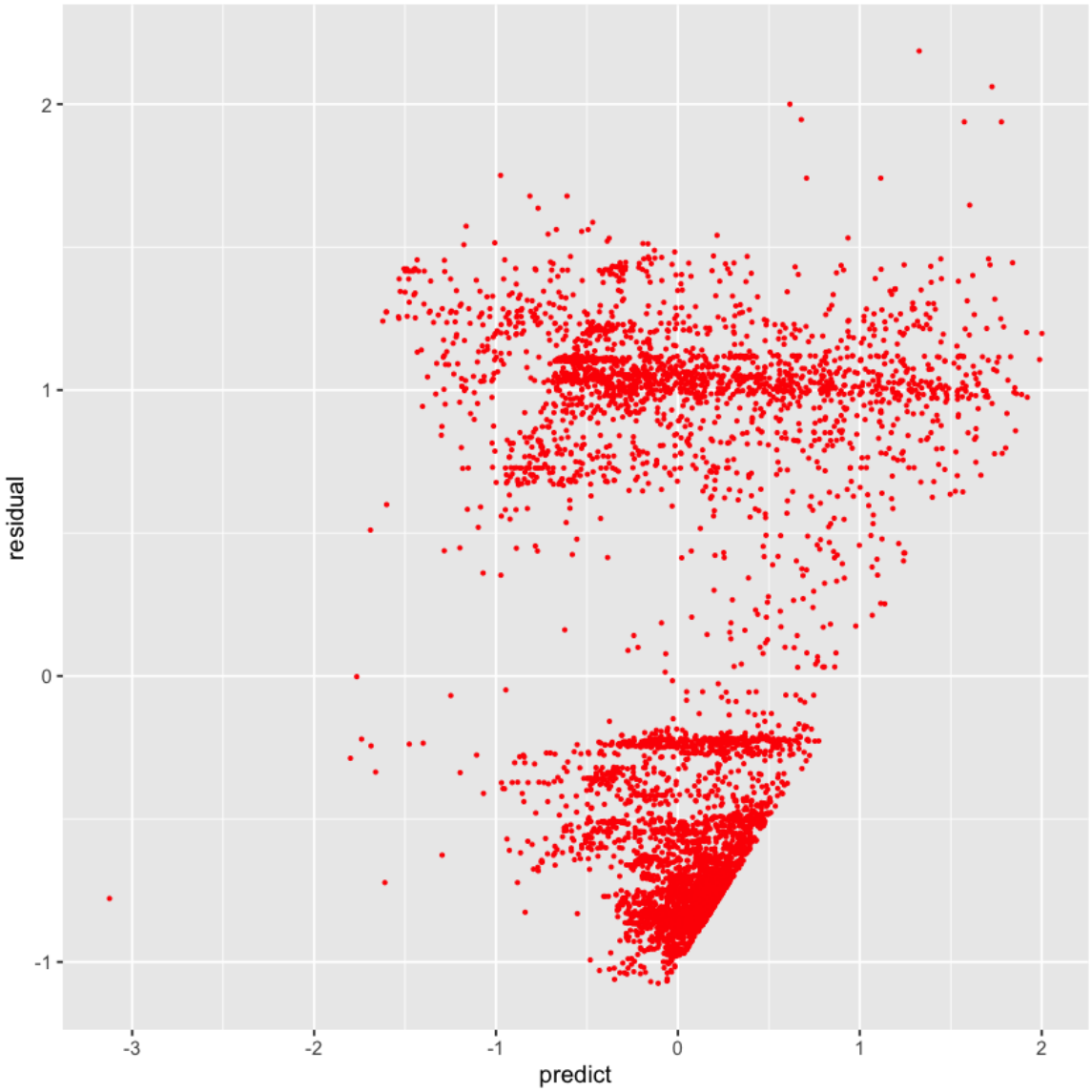
```
  |======================================================================
=====| 100%
```



In [68]:

```
plot_residual(md_deep_2)
```

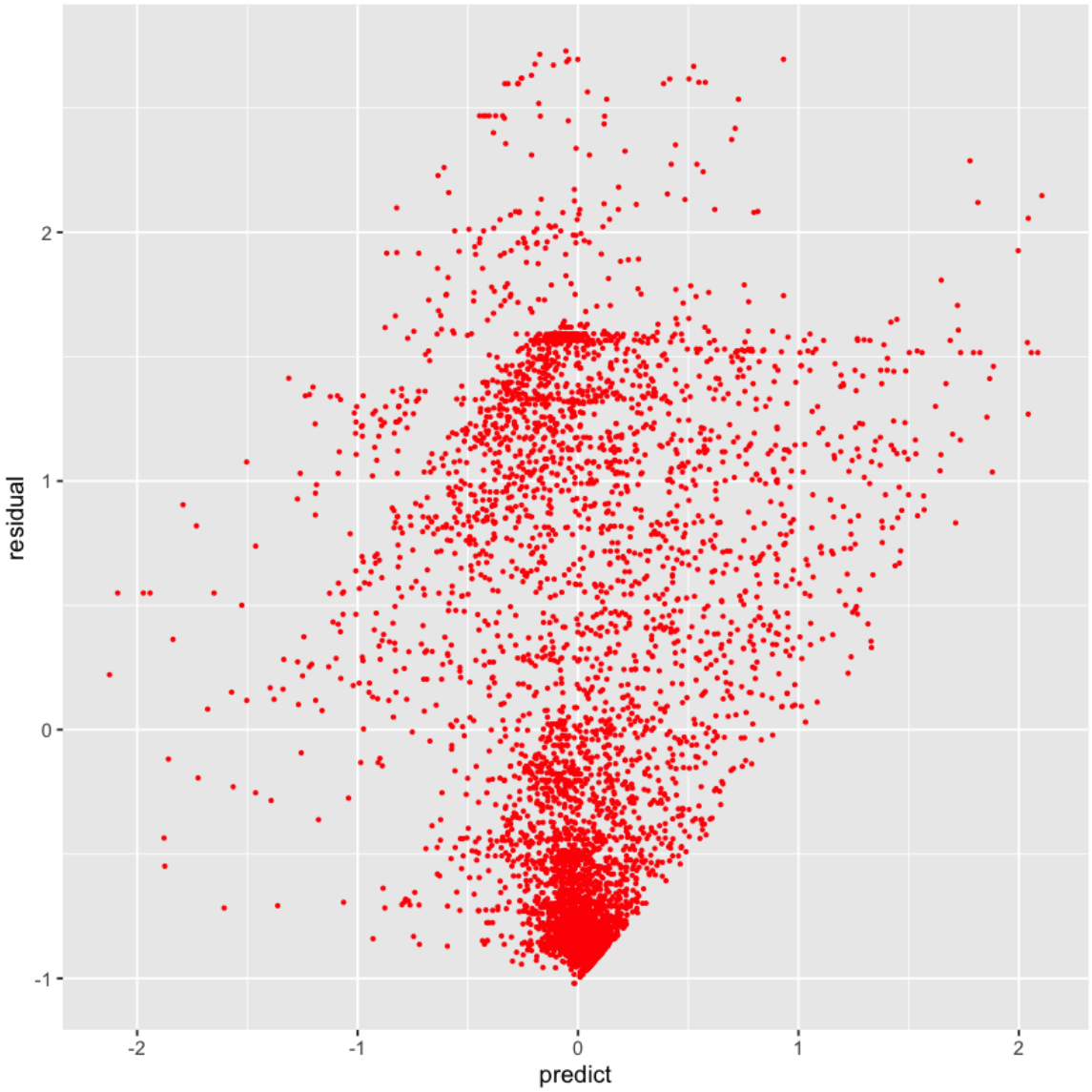
```
  |======================================================================
=====| 100%
```



In [69]:

```
plot_residual(md_xg_2)
```

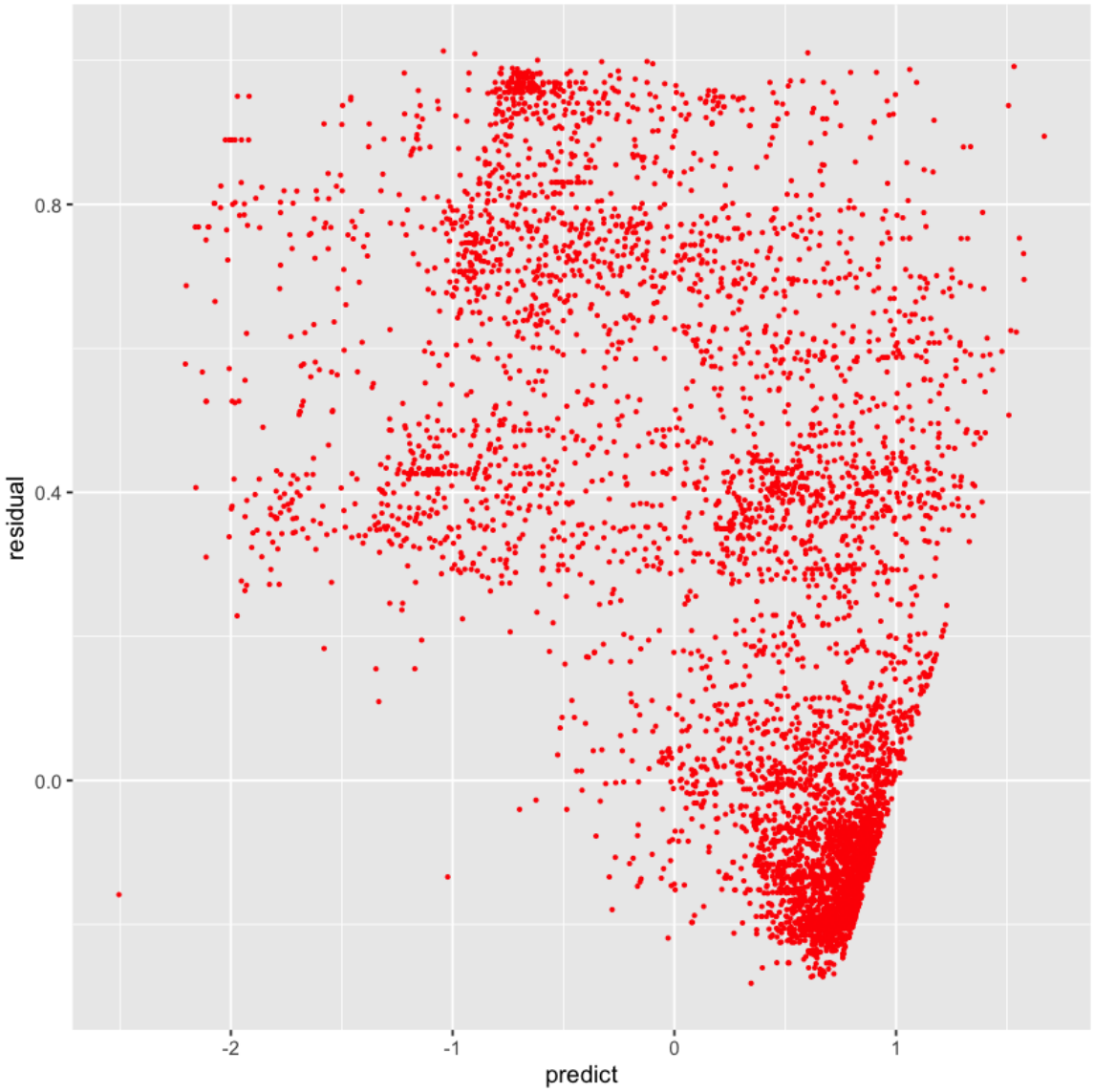
```
  |======================================================================
=====| 100%
```



In [70]:

```
plot_residual(md_aml_2.leader)
```

```
  |======================================================================
=====| 100%

Warning message in doTryCatch(return(expr), name, parentenv, handle
r):
“Test/Validation dataset is missing column 'gmean_Density': substitu
ting in a column of NaN”Warning message in doTryCatch(return(expr),
name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_std_ElectronAffinit
y': substituting in a column of NaN”Warning message in doTryCatch(re
turn(expr), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_gmean_ElectronAffini
ty': substituting in a column of NaN”Warning message in doTryCatch(r
eturn(expr), name, parentenv, handler):
“Test/Validation dataset is missing column 'std_atomic_mass': substi
tuting in a column of NaN”Warning message in doTryCatch(return(exp
r), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_gmean_ThermalConduct
ivity': substituting in a column of NaN”Warning message in doTryCatc
h(return(expr), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_range_ElectronAffini
ty': substituting in a column of NaN”Warning message in doTryCatch(r
eturn(expr), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_entropy_fie': substi
tuting in a column of NaN”Warning message in doTryCatch(return(exp
r), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_std_Valence': substi
tuting in a column of NaN”Warning message in doTryCatch(return(exp
r), name, parentenv, handler):
“Test/Validation dataset is missing column 'wtd_entropy_ThermalCondu
ctivity': substituting in a column of NaN”Warning message in doTryCa
tch(return(expr), name, parentenv, handler):
“Test/Validation dataset is missing column 'mean_ThermalConductivit
y': substituting in a column of NaN”
```


residual
0.8
0.4
0.0
-2
-1
0
1
predict

5. Variable Identification and Explanation

In [66]:

```
## Show the importance of variables to a model: 10 features
h2o.varimp(md_xg_1)[1:3,]
h2o.varimp(md_glm_1)[1:3,]
h2o.varimp(md_deep_1)[1:3,]

## Show the importance of variables to a model: 6 features
h2o.varimp(md_xg_2)[1:3,]
h2o.varimp(md_glm_2)[1:3,]
h2o.varimp(md_deep_2)[1:3,]
```

A H2OTable: 3 × 4

variable	relative_importance	scaled_importance	percentage
<chr>	<dbl>	<dbl>	<dbl>
range_atomic_radius	40999.12	1.0000000	0.4593641
wtd_mean_ThermalConductivity	14743.77	0.3596119	0.1651928
wtd_std_ThermalConductivity	11308.35	0.2758192	0.1267015

A data.frame: 3 × 4

variable	relative_importance	scaled_importance	percentage
<fct>	<dbl>	<dbl>	<dbl>
wtd_std_ThermalConductivity	0.3600654	1.0000000	0.4635049
wtd_entropy_atomic_mass	0.1378775	0.3829234	0.1774869
range_atomic_radius	0.1266171	0.3516502	0.1629916

A H2OTable: 3 × 4

variable	relative_importance	scaled_importance	percentage
<chr>	<dbl>	<dbl>	<dbl>
wtd_std_ThermalConductivity	1.0000000	1.0000000	0.1385344
wtd_mean_ThermalConductivity	0.9037772	0.9037772	0.1252043
wtd_gmean_Valence	0.7539058	0.7539058	0.1044419

A H2OTable: 3 × 4

variable	relative_importance	scaled_importance	percentage
<chr>	<dbl>	<dbl>	<dbl>
range_atomic_radius	48606.938	1.0000000	0.54690786
wtd_std_ThermalConductivity	17023.008	0.3502177	0.19153679
wtd_mean_Valence	6567.159	0.1351074	0.07389132

A data.frame: 3 × 4

variable	relative_importance	scaled_importance	percentage
<fct>	<dbl>	<dbl>	<dbl>
wtd_std_ThermalConductivity	0.4172671	1.0000000	0.5661977
range_atomic_radius	0.1504913	0.3606593	0.2042045
wtd_mean_Valence	0.1047171	0.2509594	0.1420926

A H2OTable: 3 × 4

variable	relative_importance	scaled_importance	percentage
<chr>	<dbl>	<dbl>	<dbl>
range_atomic_radius	1.0000000	1.0000000	0.1967672
wtd_entropy_Valence	0.9554759	0.9554759	0.1880063
wtd_std_ThermalConductivity	0.9153846	0.9153846	0.1801177

Variable importance is calculated by sum of the decrease in error when split by a variable. Then the relative importance is the variable importance divided by the highest variable importance value so that values are bounded between 0 and 1.

6. Conclusion

Although in this report I used the caret package's findCorrelation() API to remove highly correlated variables, the results of the model are still not ideal. After all, feature selection is not a simple matter. Just considering zero variance or correlation coefficients does not completely determine which features are worth retaining for modeling and which features should be deleted. Therefore, this modeling report has the following parts to be improved:

1. More detailed feature selection. If feature selection is not made before modeling begins, it is likely that the model being trained will be extremely complex, resulting in small errors but over-fitting. Therefore, the characteristics are not as good as possible. On the contrary, it is necessary to analyze the model by selecting the appropriate features through analysis.
2. Model selection and parameter settings. Model training is not a difficult task due to the existence of the h2o package. However, the adjustment of the parameters needs to be carried out continuously. Considering the parameter diversity of h2o, for regression, the perfection of the model requires more time to be used in the parameter adjustment, and thus the prediction ability of the update model.
3. For this report, the part of data visualization has become a shortcoming. Because h2o does not provide a rich API for mapping, it integrates the data visualization part into a Web UI. Although users can see the results of data visualization directly through the Web UI, I should use a richer way to report on the graphical interface.

7. References

1. *Chem Encyclopedia*
   https://www.infoplease.com/encyclopedia/science/chemistry/concepts/compound/properties-of-compounds
   (https://www.infoplease.com/encyclopedia/science/chemistry/concepts/compound/properties-of-compounds)
   A. *Ionization Energy*
      https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Suppl
      (https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supp
   B. *Electron Affinity*
      https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Suppl
      (https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supp
   C. *Valence*
      https://chem.libretexts.org/Bookshelves/General_Chemistry/Book%3A_ChemPRIME_(Moore_et_al.)/(
      (https://chem.libretexts.org/Bookshelves/General_Chemistry/Book%3A_ChemPRIME_(Moore_et_al.)/
2. *What is the difference between Bagging and Boosting?* https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/ (https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/)
3. *Bootstrap Sample: Definition, Example* https://www.statisticshowto.datasciencecentral.com/bootstrap-sample/ (https://www.statisticshowto.datasciencecentral.com/bootstrap-sample/)
4. *H2o Web UI* http://localhost:54321 (http://localhost:54321)
5. *Stacked Ensemble** http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html (http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html)

In [ ]: