Analysis of the Superconductor data set. Predicting critical temperature.

Mariya Kim

10/21/2021

Contents.

Section 1. Introduction.
Section 2. Analysis.
2.1. Data set overview.
2.2. Data set features (predictors).
2.2.1. Correlations.
2.2.2. Clustering.
2.3. Choosing algorithms.
2.3.1. Linear Regression.
2.3.2. k-Nearest Neighbors.
2.3.3. Random Forest.
2.4. Reducing the number of features (predictors).
2.4.1. using correlations;
2.4.2. using features variance;
2.4.3. using randomForest() "importance" matrix.
2.5. Choosing the best algorithm and predictors for the final model.
Section 3. Results.
Section 4. Conclusion.
4.1. Summary.
4.2. Future work.

Section 1. Introduction.

1.1. About Superconductor Dataset.

Superconductor data set contains data on 21263 superconductors. And can be downloaded from both kaggle.com and ics.uci.edu web sites. There two files: **train.csv** contains critical temperatures for superconducting materials (in the 82nd column *critical_temp*) and 81 features extracted from 21263 superconductors; all features are based on properties of the materials such as thermal conductivity, atomic radius, valence, electron affinity, and atomic mass and determined from chemical formula of the material, contained in the second file (**unique_m.csv**; this file will not be used in the this project). **Superconductor Dataset** originally was published in Computational Materials Science journal by Hamidieh, K., and the abstract is available at sciencedirect.com. The reported out-of-sample prediction is ± 9.5 K (kelvin) based on root-mean-squared-error (RMSE) [1].

The electrical resistance of superconductors disappears when the material is cooled. This property makes superconductors very useful in many areas. However, the phenomenon of superconductivity occurs at extremely low temperatures, close to absolute zero. The temperature at which conductivity tends to infinity and resistivity tends to zero is called the *critical temperature* (measured in kelvin, K) [2].

The goal. The goal of this project is to predict the *critical temperature* of a superconducting material based on the features extracted from the superconductor's chemical formula.

The analysis includes the following key steps:

- Exploring the structure of the data set. To find patterns and relationships in a data set, various approaches will be used, such as: data visualization, search for correlations and clusters.
- Choosing a suitable algorithm. The following three algorithms will be compared:
  (1) Linear Regression;
  (2) k-Nearest Neighbors;
  (3) Random Forest.
- Filtering features. There are 81 predictors in **Superconductor** data set, so reducing the number of predictors optimizes the final algorithm.

Section 2. Analysis.

Packages:

```
if(!require(readxl)) install.packages("readxl")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(tidyr)) install.packages("tidyr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(caret)) install.packages("caret")
if(!require(reshape2)) install.packages("reshape2")
if(!require(stringr)) install.packages("stringr")
if(!require(RColorBrewer)) install.packages("RColorBrewer")
if(!require(matrixStats)) install.packages("matrixStats")
if(!require(randomForest)) install.packages("randomForest")
if(!require(knitr)) install.packages("knitr")
if(!require(tinytex)) install.packages("tinytex")
if(!require(markdown)) install.packages("markdown")
```

2.1. Data set overview.

The original file **train.csv** was re-saved as **4_Superconductors.xlsx** file.

```
fileName <- "4_Superconductors.xlsx"
superconductors <- read_excel(fileName) # Reading  Superconductor Dataset xlsx file
```

The dimensions of the data set are **21263, 82**. There are **0** missing entries.

```
n <- 1:82
class_col <- sapply(n, function(n){
  class(pull(superconductors[,n]))
})
# To see columns classes: class_col %>% unique()
```

All columns are **numeric** variables, with different scales. The first six rows of some predictors and outcome, critical_temp (the last column) are shown below:

```
## # A tibble: 6 x 5
##   number_of_eleme~ wtd_gmean_atomi~ wtd_range_fie entropy_FusionH~ critical_temp
##              <dbl>            <dbl>         <dbl>            <dbl>         <dbl>
## 1                4             36.1          736.             1.09            29
## 2                5             36.4          743.             1.37            26
## 3                4             36.1          743.             1.09            19
## 4                4             36.1          740.             1.09            22
## 5                4             36.1          729.             1.09            23
## 6                4             36.1          714.             1.09            23
```

Variable *number_of_elements* should be integer by definition:

```
identical(superconductors$number_of_elements,
          as.numeric(as.integer(superconductors$number_of_elements)))
```

```
## [1] TRUE
```

The variable we need to predict is *critical_temp* ranges from 2.1×10^{-4} to **185**.

Now, **superconductors** data set will be divided into three parts: **train** set, **test** set and **validation** set. 20% of the original data set will be used for the final evaluation of the algorithm as **validation** set.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
index_val <- createDataPartition(y = superconductors$critical_temp,
                                 times = 1, p = 0.2, list = FALSE)
validation <- superconductors[index_val, ]
sc <- superconductors[-index_val, ]
```

The **sc** set contains 80% of the original data and is split into **train** and **test** sets.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
index <- createDataPartition(y = sc$critical_temp, times = 1, p = 0.4, list = FALSE)
test_set <- sc[index, ]
train_set <- sc[-index, ]
```

Set **sc** does not have **validation** set rows and will be used for data visualization and to define algorithms parameters as **train** set and **test** set or using cross validation approach.

2.2. Data set features (predictors).

2.2.1 Correlations.

The correlation heatmap (Fig. 1) shows that there are some highly correlated variables in this data set that can be removed to reduce the number of features.

```
correlations <- round(cor(sc), 3) %>% data.frame()
correlations <- rownames_to_column(correlations, "col_name")
correlations_long <- melt(correlations) %>%
  set_names(c("col_name", "variable", "value")) %>% arrange(desc(value))
```

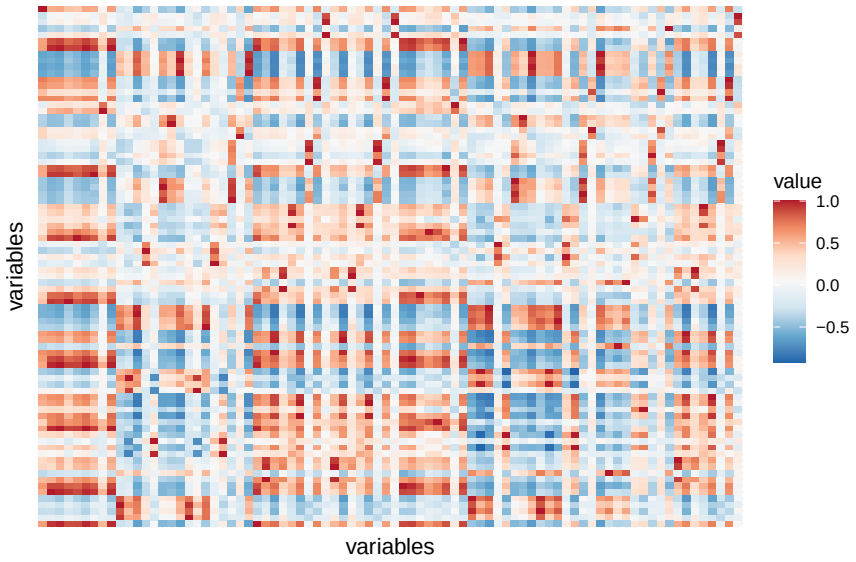

Fig. 1. Correlation heatmap.

Plotting *critical_temp* versus predictors shows that some of them (especially those with similar names) have similar shapes.

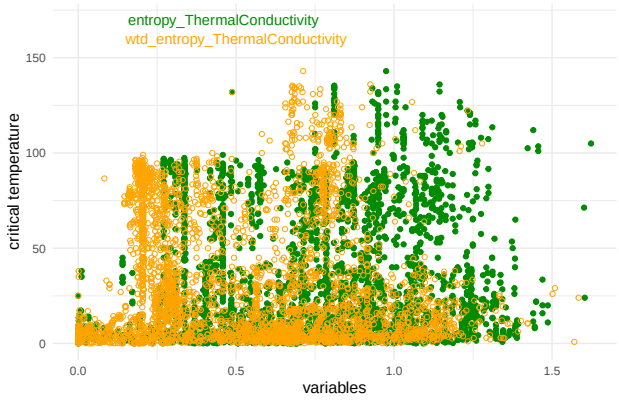

Fig. 2. Critical Temperature vs Thermal Conductivity variables.

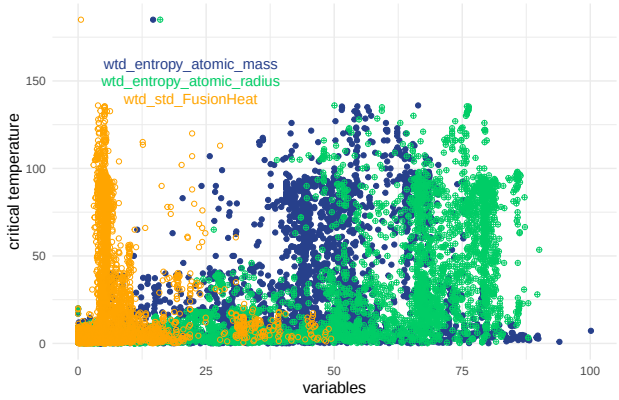

Fig. 3. Critical Temperature vs wtd_entropy variables

There are 81 features in total. Here, the names of the first 20 features:

```
head(colnames(superconductors), n = 20)
```

```
##  [1] "number_of_elements"      "mean_atomic_mass"
##  [3] "wtd_mean_atomic_mass"    "gmean_atomic_mass"
##  [5] "wtd_gmean_atomic_mass"   "entropy_atomic_mass"
##  [7] "wtd_entropy_atomic_mass" "range_atomic_mass"
##  [9] "wtd_range_atomic_mass"   "std_atomic_mass"
## [11] "wtd_std_atomic_mass"     "mean_fie"
## [13] "wtd_mean_fie"            "gmean_fie"
## [15] "wtd_gmean_fie"           "entropy_fie"
## [17] "wtd_entropy_fie"         "range_fie"
## [19] "wtd_range_fie"           "std_fie"
```

There are some patterns in column names (except the 1st column *number_of_elements*). The second part of the column name refers to the physical/chemical property that the feature is based on, such as atomic mass, density, electron affinity, etc. So columns can be grouped based on these properties and we can take a closer look at correlations within groups.

Another way to group features is to use the first part of the column name, which is related to calculations such as mean, std, entropy (related to calculations in this case) etc.

Using "atomic_mass" as a pattern

```
col_name <- str_subset(colnames(sc), "atomic_mass") # length(col_name)
```

results in 10 column names, all columns that ends with "_atomic_mass".

Now, removing "_atomic_mass" part from the column names allow us to extract patterns for the first part of the column names.

```
patterns1 <- str_replace(col_name, "_atomic_mass", "")
```

A ^ symbol has to be added to each pattern, otherwise, for example "mean" pattern also will detect "wtd_mean", "gmean", and "wtd_gmean".

```
patt_unite <- data.frame(a = rep("^", 10), b = patterns1)
patterns1 <- unite(patt_unite, a, b, col = "c", sep = "") %>% pull()
```

Removing the first part of the column name allows us to extract patterns for the second part of the column names.

```
string <- toString(patterns1)    # creating a regex that contains all variants of the first
string <- str_replace_all(string, ", ", "|")  #  part of the column names, separated by "|"
patterns2 <- str_replace(colnames(sc), string, "") %>% unique()
patterns2 <- patterns2[2:9] # removing "number_of_elements" and "critical_temp"
```

Thus there 10 groups if features are grouped by calculations and 8 if they are grouped by physical/chemical property. The patterns that can be used to group features are summarized in **Table 1**.

```
table_1 <- data.frame(by_phys_chem_property = c(patterns2, " ", " "),
                      by_calculations = patterns1)
kable(table_1, caption = "The patterns that can be used to group features.")
```

Table 1: The patterns that can be used to group features.

by_phys_chem_property	by_calculations
_atomic_mass	^mean
_fie	^wtd_mean
_atomic_radius	^gmean
_Density	^wtd_gmean
_ElectronAffinity	^entropy
_FusionHeat	^wtd_entropy
_ThermalConductivity	^range
_Valence	^wtd_range
	^std
	^wtd_std

One way to select features is to simply pick one, the most correlated with the critical temperature, from each group. Both patterns are combined to find these features (**Table 2**).

```
corr_all <- correlations_long %>%  # correlation coefficients of all features
  filter(variable == "critical_temp") %>%                # with critical_temp;
  select(col_name, value) %>%
  arrange(desc(abs(value)))       # ordering correlation coefficients (by absolute values);

p <- c(patterns2, patterns1)       # this function takes each group of features and keeps
table_2 <- map_df(p, function(p){  # the first feature with the highest correlation coefficient.
  corr_all %>% filter(col_name %in% str_subset(colnames(sc), p)) %>% slice(n = 1)
})
```

Indices for the features that are listed in **Table 2**:

```
index_tbl2 <- which(colnames(sc) %in% table_2$col_name)
```

Fewer column indices are detected because groups are overlapping:

```
nrow(table_2)
```

```
## [1] 18
```

```
length(index_tbl2)
```

```
## [1] 15
```

Table 2: Features, one from each group, that are the most correlated with the critical temperature.

col_name	value
wtd_entropy_atomic_mass	0.627
range_fie	0.598
range_atomic_radius	0.653
gmean_Density	-0.540
entropy_ElectronAffinity	0.435
wtd_entropy_FusionHeat	0.565
wtd_std_ThermalConductivity	0.722
wtd_mean_Valence	-0.631
mean_Valence	-0.599
wtd_mean_Valence	-0.631
gmean_Valence	-0.572
wtd_gmean_Valence	-0.614
entropy_Valence	0.598
wtd_entropy_atomic_mass	0.627
range_ThermalConductivity	0.689
wtd_range_ThermalConductivity	0.472
std_ThermalConductivity	0.655
wtd_std_ThermalConductivity	0.722

There is one feature that does not belong to any group. It is *number_of_elements* and this integer variable ranges from 1 to 9 and can be used as factor to stratify the data.

For example, the correlation coefficient of *entropy_ThermalConductivity* with *critical_temp* is 0.088. However, plotting these variables on a graph and coloring by *number_of_elements* clearly shows the groups of points colored by *number_of_elements* (**Fig. 4**).

Also, some separation for different critical temperatures can be seen in the *range_Valence vs critical_temp plot* (correlation coefficient is -0.144, **Fig. 5**)

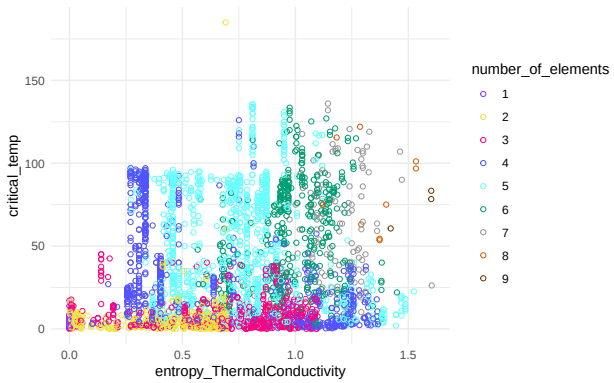

Fig. 4. Critical Temperature vs entropy_ThermalConductivity colored by number_of_elements

Fig. 5. Critical Temperature vs range_Valence colored by number_of_elements

An interesting effect can be seen on *Critical Temperature vs entropy_ElectronAffinity* plot. **Figure 6** clearly shows the positive correlation of the critical temperature with *entropy_ElectronAffinity* (a correlation coefficient is 0.435). However, if we group the data by *number_of_elements*, then all slopes calculated for the *critical temperature vs entropy_ElectronAffinity* curves will be negative (**Table 3**). So here we can observe the Simpson's paradox (**Fig. 7**).

```
# slope of the regression line for entropy_ElectronAffinity and critical_temp:
slope_entropy_ElectronAffinity <- sc %>%
  summarise(slope = cor(entropy_ElectronAffinity, critical_temp)*
              sd(critical_temp)/sd(entropy_ElectronAffinity)) %>% pull()
intercept_entropy_ElectronAffinity <- mean(sc$critical_temp) -  # intercept
  mean(sc$entropy_ElectronAffinity)*slope_entropy_ElectronAffinity
table_3 <- sc %>% group_by(number_of_elements)  %>%
  summarise(slopes = round(cor(entropy_ElectronAffinity, critical_temp)*
              sd(critical_temp)/sd(entropy_ElectronAffinity), 3),
            intercepts = round(mean(critical_temp) -
              mean(entropy_ElectronAffinity)*slopes, 3),
            corr_coef = round(cor(entropy_ElectronAffinity, critical_temp), 3) )
kable(table_3, caption = "Correlation coefficients entropy_ElectronAffinity with
 critical_temperature, gruoped by number_of_elements.")
```

Table 3: Correlation coefficients entropy_ElectronAffinity with critical_temperature, gruoped by number_of_elements.

number_of_elements	slopes	intercepts	corr_coef
1	NA	NA	NA
2	-5.529	8.745	-0.123
3	-15.900	23.473	-0.259
4	-66.208	111.275	-0.252
5	-117.344	201.255	-0.438
6	-126.416	235.232	-0.465
7	-36.554	121.506	-0.145
8	-19.189	119.659	-0.067
9	-205.199	421.855	-0.286

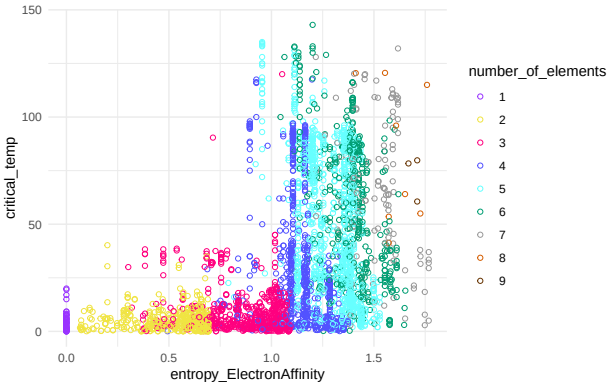

Fig. 6. Critical Temperature vs entropy_ElectronAffinity colored by number_of_elements

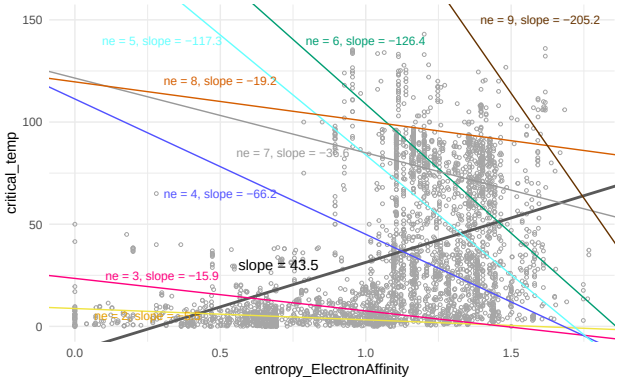

Fig. 7. Critical Temperature vs entropy_ElectronAffinity and correlation lines

Grouping features and building correlation heatmaps shows that in some groups the features correlate less with each other, in some more.

The correlation heatmap for features in the "Thermal Conductivity" group shows that features have strong correlations with no more than 1-4 other features in this group (**Fig 8**).

```
# correlation coefficients for the features in "ThermalConductivity" group:
corr_ThermalConductivity <-  sc[str_subset(colnames(sc), "ThermalConductivity")] %>% cor() %>% data.fra
corr_ThermalConductivity <- rownames_to_column(corr_ThermalConductivity, "col_name")
corr_ThermalConductivity <- melt(corr_ThermalConductivity) %>%
  set_names(c("col_name", "variable", "value")) # converting to the long the format for ggplot
```

The correlation heatmap for features grouped by the "^entropy" pattern shows that many features are highly correlated with others (all but one *entropy_ThermalConductivity*, **Fig. 9**)

```
# correlation coefficients for the features in "^entropy" group:
corr_entropy <-  sc[str_subset(colnames(sc), "^entropy")] %>%
  cor() %>% data.frame()
corr_entropy <- rownames_to_column(corr_entropy, "col_name")
corr_entropy <- melt(corr_entropy) %>%  # converting to the long the format for ggplot
  set_names(c("col_name", "variable", "value"))
```

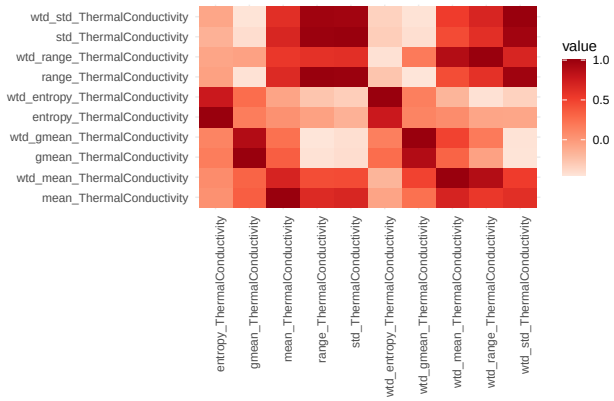

Fig. 8. Correlation heatmap of Thermal Conductivity variables.

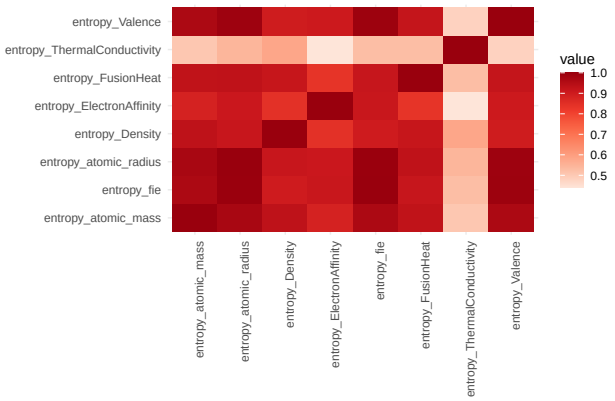

Fig. 9. Correlation heatmap of ^entropy variables.

Thus we want to keep features that are less correlated with each other within their group and have higher correlation with the critical temperature when stratified by *number_of_elements*, and will be summarized in **Table 4**.

```
# features that are less correlated with each other within their group:
corr_less <- map_df(p, function(p){
  corr <- cor(sc[which(colnames(sc) %in% str_subset(colnames(sc), p))])
  min_corr_features <- corr %>% abs() %>% colMeans() %>% as.data.frame() %>%
    rownames_to_column("col_name") %>% filter(. < 0.4) %>% select(col_name)
  min_corr_features
})
corr_less <- left_join(corr_less, corr_all) %>% arrange(desc(abs(value)))
# some variables are doubled in the "corr_less", due to groups overlapping
# so there are fewer indices in the list "index_corr_less"
index_corr_less <- which(colnames(sc) %in% corr_less$col_name)
# Adding indices for the column number_of_elements (1) and critical_temp (82)
index_corr_less_1_82 <- c(index_corr_less, 1, 82)
ne <- 1:9 # ne is number_of_elements ranges from 1 to 9
# Computing the correlation coefficients with critical_temp, stratified
# by number_of_elements:
table_4 <- map_dfc(ne, function(ne){
```

```
  corr <- sc[index_corr_less_1_82] %>% filter(number_of_elements == ne) %>%
    select(-number_of_elements) %>% cor() %>% as.data.frame() %>% select(critical_temp)
})
table_4 <- rownames_to_column(table_4, "col_name")
table_4 <- left_join(table_4, corr_all)
table_4 <- table_4 %>% set_names("col_name", "ne_1", "ne_2", "ne_3", "ne_4",
                                 "ne_5", "ne_6", "ne_7", "ne_8", "ne_9", "all") %>%
  mutate(ne_1 = round(ne_1, 2), ne_2 = round(ne_2, 2), ne_3 = round(ne_3, 2),
         ne_4 = round(ne_4, 2), ne_5 = round(ne_5, 2), ne_6 = round(ne_6, 2),
         ne_7 = round(ne_7, 2), ne_8 = round(ne_8, 2), ne_9 = round(ne_9, 2),
         all = round(all, 2))
# keeping features that have higher correlation coefficients with
# critical temperature when grouped by number_of_elements (at least in one group):
table_4 <- table_4 %>%
  filter(abs(ne_1) >= 0.45 | abs(ne_2) >= 0.45 |abs(ne_3) >= 0.45 |
           abs(ne_4) >= 0.45 | abs(ne_5) >= 0.45 |abs(ne_6) >= 0.45 |
           abs(ne_7) >= 0.45 |abs(ne_8) >= 0.45 | abs(ne_9) >= 0.45)
```

Table 4: Variables that have higher correlation with the critical temperature when stratified by *number_of_elements* and less correlated with each other

col_name	ne_1	ne_2	ne_3	ne_4	ne_5	ne_6	ne_7	ne_8	ne_9	all
entropy_Density	NA	-0.01	-0.02	-0.23	-0.20	-0.13	-0.41	-0.45	0.29	0.46
std_Density	NA	-0.19	-0.24	-0.06	0.14	0.35	0.38	0.66	0.29	0.11
gmean_ElectronAffinity	0.10	-0.11	-0.25	-0.23	-0.40	-0.52	-0.20	0.30	-0.29	-0.39
entropy_ElectronAffinity	NA	-0.12	-0.26	-0.25	-0.44	-0.47	-0.14	-0.07	-0.29	0.44
wtd_mean_FusionHeat	0.22	0.28	0.13	-0.42	-0.30	-0.10	0.06	0.16	-0.70	-0.39
wtd_range_FusionHeat	NA	0.30	0.20	-0.38	-0.24	-0.09	0.06	0.19	-0.84	-0.32
wtd_std_FusionHeat	NA	0.29	0.12	-0.32	-0.31	-0.19	-0.14	-0.34	-0.69	-0.20
mean_ThermalConductivity	0.02	-0.07	0.18	0.44	0.48	0.56	0.40	0.12	-0.29	0.38
wtd_mean_ThermalConductivity	0.02	-0.09	-0.02	0.46	0.52	0.45	0.44	0.34	0.20	0.38
entropy_ThermalConductivity	NA	0.03	-0.05	-0.58	-0.21	0.13	0.19	-0.05	0.29	0.09
wtd_entropy_ThermalConductivity	NA	0.08	-0.01	-0.56	-0.27	-0.04	0.26	0.41	-0.91	-0.11
wtd_range_ThermalConductivity	NA	-0.10	-0.01	0.63	0.51	0.36	0.30	0.18	0.41	0.47
range_Valence	NA	0.11	-0.10	-0.60	-0.37	-0.27	-0.04	0.40	-0.29	-0.14
std_Valence	NA	0.11	-0.11	-0.61	-0.35	-0.25	0.00	0.20	-0.29	-0.21
wtd_std_Valence	NA	0.09	-0.10	-0.63	-0.46	-0.33	-0.06	0.04	-0.37	-0.30
critical_temp	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Combining indices for the features that have higher correlation with the critical temperature (**Table 2**) and features that are less correlated with each other within their group and have higher correlation with the critical temperature when stratified by *number_of_elements* (**Table 4**):

```
index_tbl4 <- which(colnames(sc) %in% table_4$col_name)
# - combining indices from table 2 and table 4;
# - adding index for column number_of_elements (column 1);
# - making sure, that there no doubled indices:
index_corr <- c(1, index_tbl2, index_tbl4) %>% unique()
```

30 indices include **29** indices for features and index for outcome (critical_temp).

2.2.2. Clustering.

Another way to discover the structure of a data set is to use clustering algorithms. Here a heatmap() function is used to find patterns in the data set.

```
x <- sc[1:81]  %>% as.matrix()
y <- sc[82] %>% round() %>% pull()
rownames(x) <- y   # critical_temp column is turned into row names
x <- sweep(x, 2, colMeans(x)) %>% as.matrix() # removing the center
```

To reduce the number of features, that will be plotted we keep only variables with higher variance.

```
sds <- colSds(x) # computing SDs
ind <- order(sds, decreasing = TRUE)[1:20] # indices for the columns with higher variance
```

Heatmap shows that features extracted from the same physical/chemical property (with similar second part of the column name) are listed together (**Fig. 10**). Features with higher variance are in the groups: Density, fie and Thermal Conductivity.

```
heatmap(t(x[,ind]), col = brewer.pal(11, "Spectral"),
                    scale = "column", Colv = NA)
```

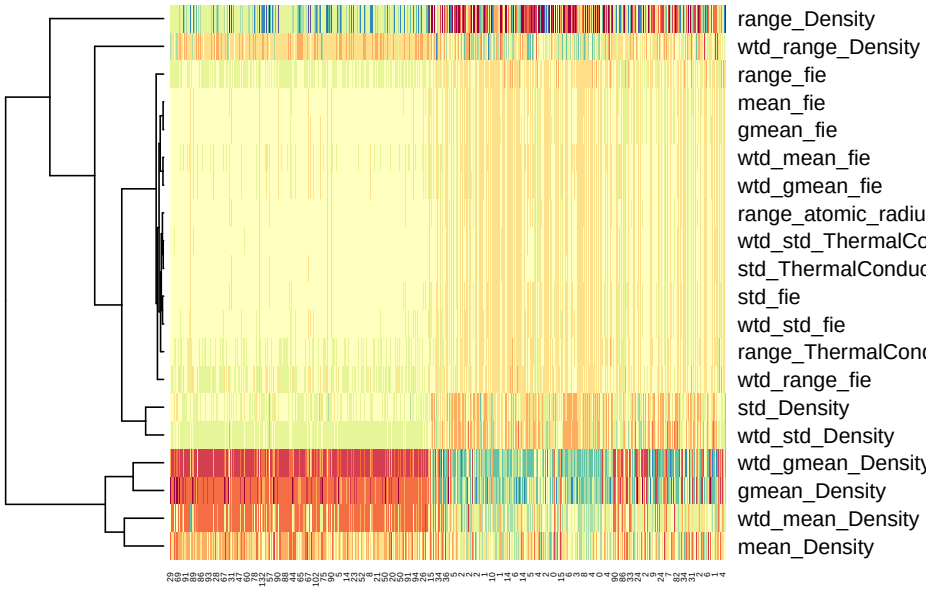

Fig. 10. Heatmap of the features with higher variance.

The indexes for variables with higher variance can also be used to reduce the number of features. To compare different approaches to filtering features, the same number of features will be used to compile a list of indices.

```
c <- length(index_corr) - 1
index_sd <- order(sds, decreasing = TRUE)[1:c]
index_sd <- c(index_sd, 82)
```

Clusters within groups of features also can be discovered using heatmap() function. **Figure 11** shows cluster heatmaps for features grouped by "_ElectronAffinity" and "^wtd_mean" patterns, the first of which is associated with a material property, the second with calculations. In **Figure 11 (upper)** the features are paired:

maen and *gmean*;
std and *wtd_std*;
entropy and *wtd_entropy*;
wtd_mean and *wtd_gmean*.

indicating similar predictors that were obtained using similar calculations.

In **Figure 11 (lower)** all features are based on different properties; *wtd_mean_Density* stands out and shows that Density is the least similar to other material properties.

```
ind <- which(colnames(sc) %in% colnames(sc[str_subset(colnames(sc), "_ElectronAffinity")]))
heatmap(t(x[,ind]), col = brewer.pal(11, "Spectral"),
          scale = "column",  Colv = NA)
ind <- which(colnames(sc) %in% colnames(sc[str_subset(colnames(sc), "^wtd_mean")]))
heatmap(t(x[,ind]), col = brewer.pal(11, "Spectral"),
        scale = "column",  Colv = NA)
```

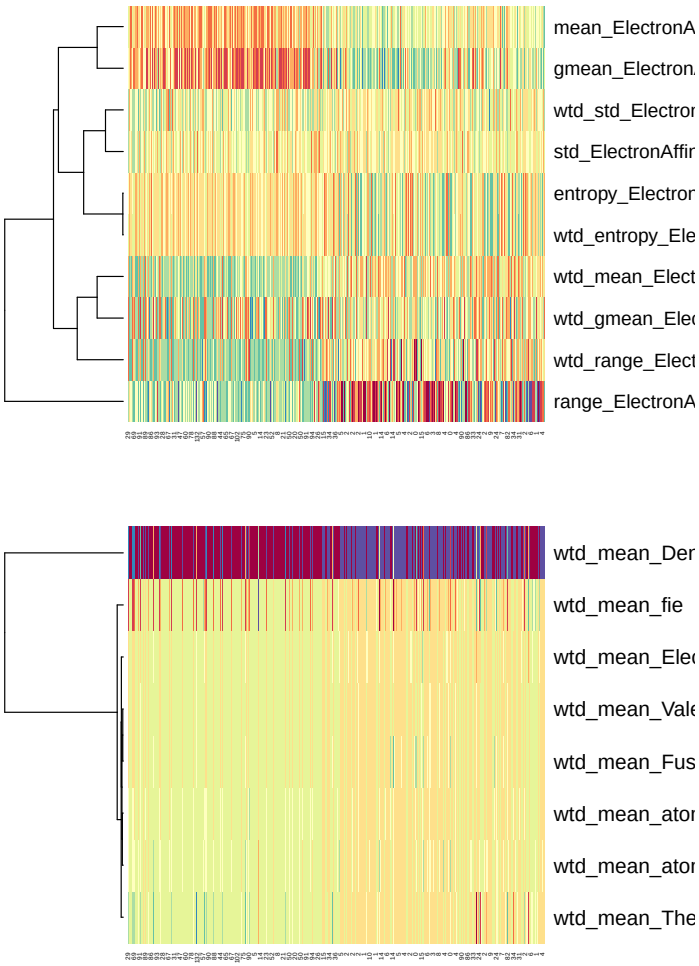

Fig. 11. Cluster heatmaps of the groups of features.

2.3. Choosing an algorithm.

Several algorithms have been learned as part of the *Data Science, Machine Learning* course, on **edx**.

```
method <- c("lm", "glm", "knn", "lda", "qda",  "rpart", "rf")
table_5 <- map_dfr(method, function(method){
  modelLookup(method)
})
kable(table_5, caption = "Algorithms.")
```

Table 5: Algorithms.

model	parameter	label	forReg	forClass	probModel
lm	intercept	intercept	TRUE	FALSE	FALSE
glm	parameter	parameter	TRUE	TRUE	TRUE
knn	k	#Neighbors	TRUE	TRUE	TRUE
lda	parameter	parameter	FALSE	TRUE	TRUE
qda	parameter	parameter	FALSE	TRUE	TRUE
rpart	cp	Complexity Parameter	TRUE	TRUE	TRUE
rf	mtry	#Randomly Selected Predictors	TRUE	TRUE	TRUE

The choice of models is based on the properties of the data set:

- the outcome is continuous variable;
- there are many predictors;
- RMSE is used to estimate the result.

Bin smoothing techniques (loess, ksmooth) were also part of the course, but these algorithms are not suitable for many predictors.

The following three methods will be used for this project:
(1) Linear Regression. Although the relationships outcome - predictor are not linear, this method can be used as a starting point and for comparison.
(2) k-Nearest Neighbors, since it admits a continuous outcome and many predictors.
(3) Random forest as an extended regression tree, can also be used for continuous outcome and many predictors.

Different approaches to filtering predictors will also be explored.
The results will be compared using:
- the root-mean-squared-error (which is an error in $\pm$ K (kelvin degrees));
- estimate the computing time required to fit the data, using system.time() function.

Root-mean-squared-error function:

```
RMSE <- function(true_temp, predicted_temp){
  sqrt(mean((true_temp - predicted_temp)^2))
}
```

The selected algorithms will be compared using all 81 predictors.

2.3.1. Linear Regression, using lm() function and all predictors.

```
fit_lm <- lm(critical_temp ~ ., data = train_set)
pred_lm <- predict(fit_lm, newdata = test_set)
rmse_lm <- RMSE(test_set$critical_temp, pred_lm)
# Measuring the time required to compute linear regression.
time_lm <- system.time(lm(critical_temp ~ ., data = train_set))
```

2.3.2. k-Nearest Neighbors (KNN). Since knn is based on calculating distances and all columns are in very different ranges, all predictors are scaled:

```
# scaling train set;
train_scaled <- train_set %>%
  select(-critical_temp) %>%
  scale() %>%
  cbind(train_set$critical_temp)  %>% as.data.frame()
colnames(train_scaled)[82] <- "critical_temp"
# scaling test set.
test_scaled <- test_set %>%
  select(-critical_temp) %>%
  scale() %>%
  cbind(test_set$critical_temp) %>% as.data.frame()
colnames(test_scaled)[82] <- "critical_temp"
```

Finding the best number of neighbours (k) using train() function and cross validation technique.

```
set.seed(1, sample.kind="Rounding")
control <- trainControl(method = "cv", number = 10, p = 0.9)
tune_knn <- train(critical_temp ~ .,
                  method = "knn",
                  data = train_scaled,
                  tuneGrid = data.frame(k = seq(1, 30, 2)),
                  trControl = control)
```

Fitting the data using the best k and all predictors.

```
set.seed(1, sample.kind="Rounding")
fit_knn <- train(critical_temp ~ ., method = "knn",
                   data = train_scaled,
                   tuneGrid = data.frame(k = tune_knn$bestTune))

pred_knn <- predict(fit_knn, test_scaled, type = "raw")
rmse_knn <- RMSE(test_set$critical_temp, pred_knn)

# measuring the time required to run KNN algorithm
time_knn <- system.time(train(critical_temp ~ ., method = "knn",
                  data = train_scaled,
                  tuneGrid = data.frame(k = 3)))
```

2.3.3. Random Forest.

First, using all predictors and default parameters:

```
set.seed(1, sample.kind="Rounding")
fit_rf <- randomForest(critical_temp ~ ., data = train_set)
pred_rf <- predict(fit_rf, newdata = test_set)
rmse_rf_def <- RMSE(test_set$critical_temp, pred_rf)
```

Second, finding the best parameters. randomForest() will be used to tune the following parameters: *ntree*, *mtry* and *nodesize*. And result will be compared with default parameters, which are mtry = **27** (p/3 = 81/3, where p is the number of predictors); nodesize = 5 (for regression, according to the help file) and ntree = **500**.

A smaller data set (about 1/3 of the train set) will be used to tune parameters.

```
set.seed(1, sample.kind="Rounding")
train <- slice_sample(train_set, prop = 0.3)
```

To find best "ntrees" the other parameters are set to default values.

```
set.seed(1, sample.kind="Rounding")
nt <- c(400, 500, 700, 1000) # the defaul ntree is 500
tune_rf_ntree <- sapply(nt, function(nt){
  tune_rf <- randomForest(critical_temp ~ ., data = train,
                          ntree = nt, nodesize = 5, mtry = 27)
  pred_rf <- predict(tune_rf, newdata = test_set)
  rmse_rf <- RMSE(test_set$critical_temp, pred_rf)
  rmse_rf
})
ntree_min <- nt[which.min(tune_rf_ntree)]
```

Finding mtry:

```
set.seed(1, sample.kind="Rounding")
mtry <- c(21, 25, 27, 29 ,33, 40, 50)
tune_rf_mtry <- sapply(mtry, function(mtry){
  tune_rf <- randomForest(critical_temp ~ ., data = train,
                          ntree = ntree_min, nodesize = 5, mtry = mtry)
  pred_rf <- predict(tune_rf, newdata = test_set)
  rmse_rf <- RMSE(test_set$critical_temp, pred_rf)
  rmse_rf
})
mtry_min <- mtry[which.min(tune_rf_mtry)]
```

Finding nodesize:

```
set.seed(1, sample.kind="Rounding")
ns <- c(4, 5, 6)
tune_rf_nodesize <- sapply(ns, function(ns){
  tune_rf <- randomForest(critical_temp ~ ., data = train,
                          ntree = ntree_min, nodesize = ns, mtry = mtry_min)
  pred_rf <- predict(tune_rf, newdata = test_set)
  rmse_rf <- RMSE(test_set$critical_temp, pred_rf)
  rmse_rf
})
nodesize_min <- ns[which.min(tune_rf_nodesize)]
```

Using whole train set with best parameters:

```
set.seed(1, sample.kind="Rounding")
fit_rf_tuned <- randomForest(critical_temp ~ ., data = train_set,
                          ntree = ntree_min, nodesize = nodesize_min, mtry = mtry_min)
pred_rf <- predict(fit_rf, newdata = test_set)
rmse_rf_tuned <- RMSE(test_set$critical_temp, pred_rf)
```

Comparing the results for the default and tuned settings...

```
rmse_rf_def
```

```
## [1] 10.29903
```

```
rmse_rf_tuned
```

```
## [1] 10.29903
```

... does not show any difference, so the default settings will be used below.

There is an argument 'strata' for randomForest(); according to the help file:
"A (factor) variable that is used for stratified sampling."

Thus *number_of_elements* can be used as factor for this argument.

```
# Changing number_of_elements to factor:
train_set$number_of_elements <- as.factor(train_set$number_of_elements)
test_set$number_of_elements <- as.factor(test_set$number_of_elements)

set.seed(1, sample.kind="Rounding")
fit_rf_str <- randomForest(critical_temp ~ .,
                           data = train_set,
                           strata = train_set$number_of_elements)
pred_rf <- predict(fit_rf_str, newdata = test_set)
rmse_rf_str <- RMSE(test_set$critical_temp, pred_rf)
```

Also did not improve RMSE:

```
rmse_rf_str
```

```
## [1] 10.30185
```

... therefore will not be used.

```
train_set$number_of_elements <- as.numeric(train_set$number_of_elements)
test_set$number_of_elements <- as.numeric(test_set$number_of_elements)

# measuring the time required to fit random forest algorithm
time_rf <- system.time(randomForest(critical_temp ~ ., data = train_set))
```

2.4. Reducing the number of features (predictors).

The random forest algorithm performed the best, however, it takes about twice as long as the KNN algorithm (**Table 6**).

```
table_6 <- data.frame(method = c("Linear Regression", "k-Nearest Neighbors",
                                 "Random Forest"),
                      RMSE = c(rmse_lm, rmse_knn, rmse_rf_def),
                      time = c(time_lm[3], time_knn[3], time_rf[3]))
kable(table_6, caption = "Root mean squared errors and time required for fitting
      calculated for various algorithms using 81 predictors.")
```

Table 6: Root mean squared errors and time required for fitting calculated for various algorithms using 81 predictors.

method	RMSE	time
Linear Regression	18.18404	0.06
k-Nearest Neighbors	11.78558	93.98
Random Forest	10.29903	208.97

The next step is to reduce the number of predictors. The random forest algorithm will be used to compare approaches to filtering predictors.

2.4.1. 'index_corr' is the list of the column indices for the features, which are more correlated with critical temperature (listed in **Table 2** and **Table 4**).

Random Forest algorithm, using 29 predictors that most correlate with critical temperature.

```
set.seed(1, sample.kind="Rounding")
fit_rf_corr <- randomForest(critical_temp ~ .,
                            data = train_set[index_corr])
pred_rf <- predict(fit_rf_corr, newdata = test_set[index_corr])
rmse_rf_corr <- RMSE(test_set$critical_temp, pred_rf)

#  time required to fit random forest with 29 predictors
time_rf_corr <- system.time(randomForest(critical_temp ~ .,
                                         data = train_set[index_corr]))
```

2.4.2. 'index_sd' is the list of the column indices for the features with high variance (shown on the cluster heatmap, **Fig. 10**). Random Forest algorithm, using 29 predictors with high variance.

```
set.seed(1, sample.kind="Rounding")
fit_rf_sd <- randomForest(critical_temp ~ .,
                            data = train_set[index_sd])
pred_rf <- predict(fit_rf_sd, newdata = test_set[index_sd])
rmse_rf_sd <- RMSE(test_set$critical_temp, pred_rf)
```

2.4.3. In the output of the randomForest() function, there is an **importance** matrix, that also can be used to filter predictors.

```
c <- length(index_corr) - 1
importance <- fit_rf$importance %>% as.data.frame()  %>%
  rownames_to_column() %>% slice_max(order_by = IncNodePurity , n = c)
# the list of indices for the most useful predictors in 'importance' matrix
index_imp <- which(colnames(sc) %in% importance$rowname)
# Adding critical_temp column
index_imp <- c(index_imp, 82)
```

Running Random Forest algorithm using top 29 predictors from **importance** matrix:

```
set.seed(1, sample.kind="Rounding")
fit_rf_imp <- randomForest(critical_temp ~ .,
                            data = train_set[index_imp])
pred_rf <- predict(fit_rf_imp, newdata = test_set[index_imp])
rmse_rf_imp <- RMSE(test_set$critical_temp, pred_rf)
```

Moreover, setting an argument "importance = TRUE" provides additional information in **importance** matrix. From help file: "For Regression, the first column is the mean decrease in accuracy and the second the mean decrease in MSE".

```
set.seed(1, sample.kind="Rounding")
fit_rf <- randomForest(critical_temp ~ ., data = train_set, importance = TRUE)
importance_matrix <- fit_rf$importance %>% as.data.frame()  %>% rownames_to_column()
```

Importance plot (**Fig. 12**) is used to pick the cut off: the IncNodePurity rises sharply after it reaches the value around 110000, which means that the importance of the predictors also increases more.

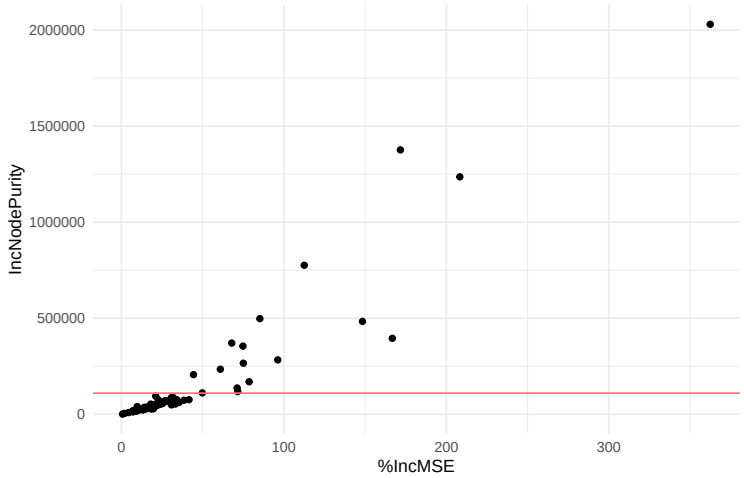

Fig. 12. Importance plot.

```
table_7 <- importance_matrix %>% filter(IncNodePurity > 110000) %>% arrange(desc(IncNodePurity))
# indices for these 17 features:
index_imp_top <- which(colnames(sc) %in% table_7$rowname)
# Adding critical_temp column
index_imp_top <- c(index_imp_top, 82)
kable(table_7, caption = "Importance matrix for top 17 predictors.")
```

Table 7: Importance matrix for top 17 predictors.

rowname	%IncMSE	IncNodePurity
range_ThermalConductivity	362.44883	2030204.9
wtd_std_ThermalConductivity	171.73773	1376129.1
range_atomic_radius	208.31533	1235879.5
wtd_entropy_Valence	112.55782	775759.3
std_ThermalConductivity	85.21322	497838.5
wtd_mean_Valence	148.36409	483559.1
wtd_gmean_Valence	166.76075	395328.1
wtd_gmean_ThermalConductivity	67.91438	370724.5
wtd_entropy_atomic_mass	74.81247	354821.8
wtd_std_ElectronAffinity	96.25859	283097.0
wtd_mean_ThermalConductivity	75.05134	265909.9
wtd_gmean_ElectronAffinity	60.90437	234528.7
wtd_range_ElectronAffinity	44.36560	206366.0
std_atomic_mass	78.64743	169280.8
wtd_entropy_ThermalConductivity	71.24894	136985.2
wtd_std_Valence	71.61217	118340.4
std_ElectronAffinity	49.80894	111369.2

Filtering predictors with IncNodePurity> 110000 results in 17 predictors being selected from **importance** matrix and listed in **Table 7**.

Running Random Forest algorithm using top 17 predictors from **importance** matrix.

```
set.seed(1, sample.kind="Rounding")
fit_rf_imp_top <- randomForest(critical_temp ~ ., data = train_set[index_imp_top])
pred_rf <- predict(fit_rf_imp_top, newdata = test_set[index_imp_top])
rmse_rf_imp_top <- RMSE(test_set$critical_temp, pred_rf)
```

```
#  time required to fit Random Forest with 17 predictors
time_rf_imp_top <- system.time(randomForest(critical_temp ~ ., data = train_set[index_imp_top]))
```

The RMSEs estimated also for less predictors and the time required for 29 predictors (evaluated only once for train_set[index_corr]) and 17 predictors are summarized in **Table 8**.

Table 8: Root mean squared errors and time required to fit the data for various algorithms and different number of predictors.

method	number_of_predictors	predictors_selected_based_on	RMSE	time
Linear Regression	81	no features filtering	18.18404	0.06
k-Nearest Neighbors	81	no features filtering	11.78558	93.98
Random Forest	81	no features filtering	10.29903	208.97
Random Forest	29	correlations	10.45128	81.11
Random Forest	29	higher variance	10.64423	81.11
Random Forest	29	RF importance	10.45894	81.11
Random Forest	17	RF importance	10.62186	56.16

2.5. Choosing the best algorithm and predictors for the final model.

For the final fit Random Forest algorithm with default settings will be used, as it showed the best result.

First, fitting the data using all predictors and **sc** set, which is a combination of the **train** set and the **test** set, but does not include the **validation** set. The **Validation** set is used to evaluate the performance of the algorithm:

```
set.seed(1, sample.kind="Rounding")
fit <- randomForest(critical_temp ~ ., data = sc)
pred <- predict(fit, newdata = validation)
rmse_val_all <- RMSE(validation$critical_temp, pred)
rmse_val_all
```

```
## [1] 9.361706
```

Second, the top 17 predictors from **importance** matrix are used to fit **sc** set because it optimizes the code by reducing the time required to fit the data.

Moreover, if we look at the 17 predictors listed in **Table 7** we can see that less properties of the materials (such as atomic mass, electron affinity, etc.) needed to be used to find these predictors and can be extracted by removing the first part of the predictor names:

```
s <- "mean_|wtd_mean_|gmean_|wtd_gmean_|entropy_|wtd_entropy_|range_|wtd_range_|std_|wtd_std_"
prop_imp <- str_replace_all(table_7$rowname, s, "") %>% unique()
prop_imp
```

```
## [1] "ThermalConductivity"
## [2] "atomic_radius"
## [3] "Valence"
## [4] "atomic_mass"
## [5] "ElectronAffinity"
```

comparing this list with the one for all predictors shows that some groups of features can be dropped:

```
# Removing the first part of all predictor names result in the list of all
# properties used to extract the features:
prop_all <- str_replace_all(colnames(sc[1:81]), s, "") %>% unique()
# The difference in the lists of properties between all predictors and
```

```
# predictors listed in Table 7:
prop_diff <- setdiff(prop_all, prop_imp)
prop_diff
```

```
## [1] "number_of_elements"
## [2] "fie"
## [3] "Density"
## [4] "FusionHeat"
```

Thus, using only 17 predictors allows not only saving time on fitting data, but also reducing the number of material properties that need to be determined in the first place. The *number_of_elements* of elements is not hard to calculate, however for properties like "Density" and "FusionHeat" additional measurements or calculations may be required.

Running a Random Forest algorithm using top 17 predictors from the **importance** matrix:

```
set.seed(1, sample.kind="Rounding")
fit <- randomForest(critical_temp ~ ., data = sc[index_imp_top])
pred <- predict(fit, newdata = validation[index_imp_top])
rmse_val <- RMSE(validation$critical_temp, pred)
```

RMSE calculated for the **validation** set is:

```
rmse_val
```

```
## [1] 9.69941
```

Section 3. Results.

- The structure of the data set.

Superconductor set has data on **21263** superconductors in **82** columns. There are 81 features; 80 features are continuous variables and based on 8 properties of the materials (which reflected in the second part of the column name), and can be grouped, based on those properties :

```
prop_all
```

```
## [1] "number_of_elements"
## [2] "atomic_mass"
## [3] "fie"
## [4] "atomic_radius"
## [5] "Density"
## [6] "ElectronAffinity"
## [7] "FusionHeat"
## [8] "ThermalConductivity"
## [9] "Valence"
```

Resulting groups have 10 features. The first part of the column names in each group are *mean, wtd_mean, gmean, wtd_gmean, entropy, wtd_entropy, range, wtd_range, std, wtd_std*; also can be used to group features.

The first column is the integer, ranges from 1 to 9 number of elements. The variable we need to predict is in the last column, *critical_temp*, continuous variable.

15 features have correlation coefficient with critical temperature from **0.44** to **0.72** (absolute values), and **16** predictors have correlation coefficient with critical temperature more than 0.45 when statified by *number_of_elements*; and summarized in **Tables 2** and **4**.

The predictors with high variance belong mainly to the groups "Density", "fie" and "Thermal Conductivity" (**Fig. 10**).

- Choosing an algorithm.

Tree different algorithms were used to fit the data; results are summarized in **Table 8**. **Train** set contains 48% of the **Superconductor** data set was used to compare the performance of algorithms and all predictors were included.

Among the algorithms tested, Random Forest with default settings showed the best result (RMSE = **10.3**). k-Nearest Neighbors is also a good choice for this data, RMSE is about 1.5 K more and it is **11.79**. Another advantage is that fitting data using the k-Nearest Neighborss algorithm takes about half the time compared to using Random Forest. Linear Regression, as expected, did not perform well; the dependence of the critical temperature on variables have different shapes and is not linear. RMSE for this method is **18.18**, and about 77% higher than for Random Forest algorithm.

- Features Filtering.

Random Forest algorithm with default settings was used to compare approaches to filtering predictors. There are 81 predictors in **Superconductor** data set, so filtering out less important predictors can optimize the final algorithm.

Reducing the number of predictors from 81 to 29 increases the RMSE by **0.15** - **0.35** (depending on the filtering approach) and halves the time it takes to fit the data. Decreasing the number of predictors even further to 17 increases the RMSE by only **0.32** K, the time required to fit the data is about a quarter of that.

Although the use of correlations gave the best result in filtering predictors, the easiest way to select predictors is to use the output of the randomForest() function in the **importance** matrix, this simplifies the code, while the difference in RMSE is very small; the RMSE, if we use correlations to select 29 predictors, is **10.4513**, and for 29 predictors from the **importance** matrix, the RMSE is **10.4589**. Using 29 predictors with higher variance to fit the data results in RMSE = **10.6442**. This approach is also an easy way to filter out predictors, however it does not account for the correlations between the outcome and the predictors, moreover, calculating standard deviations may not give good results in this case, because the variables are not normally distributed.

- Fitting 80% of the original **Superconductor** data set using the Random Forest algorithm and all features result in prediction ± **9.36**, which is slightly lower than stated in [1], RMSE = 9.5. However, it is more rational to reduce the number of predictors to 17 and predict the critical temperature ± **9.7**.

Section 4. Conclusion.

3.1. Summary.

Random Forest algorithm with default settings can predict critical temperature of the superconducting material ± **9.7** using only 17 predictors or ± **9.36** using all 81 predictors. Given that the critical temperature ranges from 3.25×10^{-4} to 185 kelvin, 0.34 kelvin difference in RMSE is not very significant when we reduce the number of predictors from 81 to 17. The choice of the final settings depend on weather, we want the algorithm to be more accurate or more optimized (faster).

3.2 Future work.

The other approaches also could be considered for this data set, such as matrix factorization, for example principal component analysis.

The main limitation of this analysis is that the found relations for *critical temperature-vs-variables* were not used for the final model. Finding linear relationships when the data is stratified by *number_of_elements* and potentially using even fewer predictors can be seen as another way to find a better model.

In this analysis the second file, available for **Superconductor** data set, was not used; it contains chemical compositions and summarized formulas. This data can be used to determine the class of the superconductor, which can be very useful since the critical temperature depends on the class of the material. On the other hand, predicting the critical temperature regardless of the material class can be seen as a challenge for this project.

Citations.

[1] K. Hamidieh, *Computational Materials Science*, **2018**, 154, 346-354

[2] Yakhmi, Jatinder Vir. *Introduction to superconductivity, superconducting materials and their usefulness.* IOP Publishing Ltd, 2021