BURNOUT IN THE WORKPLACE

Jacopo Trabona

Introduction

Burnout, as defined by Merriam-Webster, is the *exhaustion of physical or emotional strength or motivation usually as a result of prolonged stress or frustration.* In recent years, **burnout** has become a steady concern for companies and businesses, in that it appears to affect the mental health of a growing number of employees worldwide, and - accordingly - their productivity in the workplace.

In the context of the current pandemic, which is very likely to be exacerbating this phenomenon's grip, HackerEarth launched a machine learning competition, which focused on predicting the **burnout rate** in a given - and fairly large - sample of employees.

The **7 independent variables** that were included in the dataset (and which can be used for prediction) are date of joining, gender, company type, WFH setup availability, designation, resource allocation, and mental fatigue score. Most of their names are self-explanatory, aside from **Designation** (which essentially ranks employees' positions), **Resource Allocation**, which basically stands for working hours, and **WHF Setup Availability**, which specifies whether working from home is available for the observed employees.

In this report, we will be using HackerEarth dataset, not only to predict employees' **burnout rate**, but also to acquire a series of insights and other potentially influential components.

Finally, we will briefly propose an alternative use of the aforementioned dataset, namely that of predicting the **Mental Fatigue Score** of the surveyed employees.

Methods and Analysis

Install and Data Cleansing

We start by importing the dataset, which we previously downloaded and copied to our working directory.

```
#IMPORTING DATASET

#Identify data file
filename <- "train.csv"
dir <- "/Users/jacopotrabona/Downloads/burnout/upload"
fullpath <- file.path(dir, filename)

#Copy data file into project directory
file.copy(fullpath, "train.csv")

#Load data set
if(!require(tidyverse)) install.packages("tidyverse")
library(tidyverse)

dat <- read_csv("train.csv")
```

Then, we install and load the required packages.

```
#REQURED PACKAGES

#Install required packages
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(skimr)) install.packages("skimr")
if(!require(hrbrthemes)) install.packages("hrbrthemes")
if(!require(ggcorrplot)) install.packages("ggcorrplot")
if(!require(gam)) install.packages("gam")
if(!require(randomForest)) install.packages("randomForest")
if(!require(ggpubr)) install.packages("ggpubr")

#Load required packages
library(tidyverse)
library(caret)
library(skimr)
library(ggcorrplot)
library(randomForest)
library(hrbrthemes)
library(ggpubr)
```

Now we are ready to start exploring our dataset, which has **9** variables and **22750** observations.

For convenience, here we do not show the **Employee ID** variable, in that it won't be useful for the purpose of both our analysis and our machine learning task.

Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
2008-09-30	Female	Service	No	2	3	3.8	0.16
2008-11-30	Male	Service	Yes	1	2	5.0	0.36
2008-03-10	Female	Product	Yes	2	NA	5.8	0.49
2008-11-03	Male	Service	Yes	1	1	2.6	0.20
2008-07-24	Female	Service	No	3	7	6.9	0.52
2008-11-26	Male	Product	Yes	2	4	3.6	0.29

As we glance at the first rows, we promptly stumble upon several **missing values**, which we later understand to be slightly more than **20%** of the total entries.

Surprisingly, **1124** of them (which R codifies as **NAs**) are in the **Burn Rate** column, the dependent variable we eventually need to predict.

Thus, we **remove** these **missing values**.

```
#Remove observations with NAs in the Burn Rate column
newdat <- dat[!is.na(dat$`Burn Rate`),]
```

However, we are still left with **3223 NAs**, which amount to nearly **15%** of our data.

Additionally, only **187** of them spread across the same rows, implying that, were we to remove all of them, we would be left with about **77%** of our original data.

Can we afford that? Our dataset is not very large, so we may risk to produce a series of biased models, which may perform optimally only when compared to the observed outcomes in our dataset.

We thereby need to choose between three main options: we can **omit** the rows containing missing values altogether; we can **impute** all missing values using techniques such as Knn or bagging; or we can split our dataset so that to keep the test set devoid of missing values, while imputing the remaining missing values in the training set.

Prior to taking a definitive decision, we did some quick trials and noticed that both the 2nd and 3d option led to somewhat unstable models. This might be due to the fact that the remaining **NAs** are located in the **Resource Allocation** and **Mental Fatigue Score** columns, namely the most significant - and most correlated - variables for the purposes of our prediction.

```
#Columns with NAs
colSums(is.na(newdat))
```

```
##           Employee ID      Date of Joining              Gender
##                     0                    0                   0
##         Company Type  WFH Setup Available         Designation
##                     0                    0                   0
##  Resource Allocation Mental Fatigue Score           Burn Rate
##                 1278                 1945                   0
```

Furthermore, these approaches might perform better when applied to **classification** tasks, but, since what we are faced with is a **regression problem**, we will opt for the first option and use **repeated k-fold cross validation** to tackle the small size of our dataset.

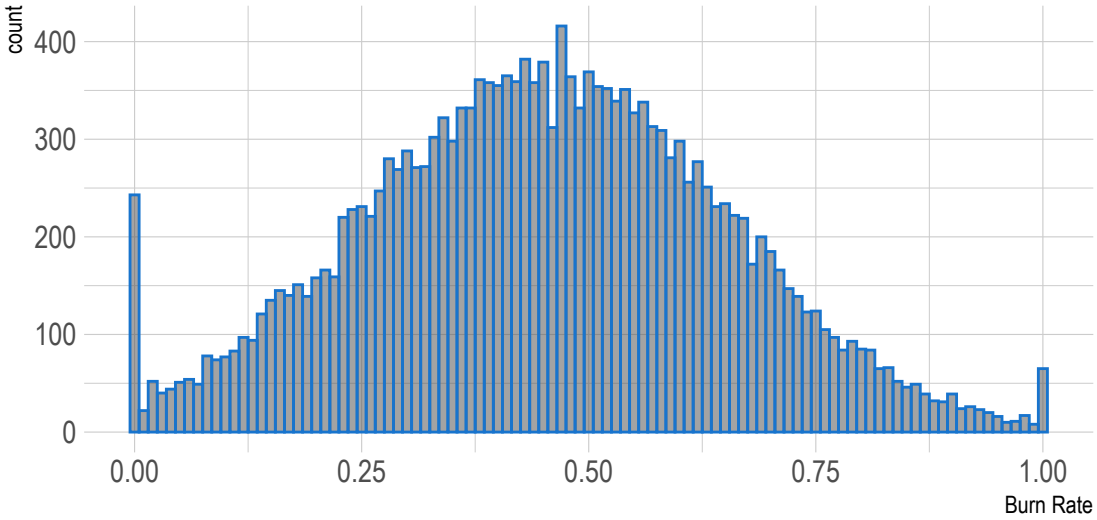
Thus, we **omit** all rows containing missing values, by using na.omit().

```
#Omit row containing missing values
newdat <- na.omit(newdat)
```

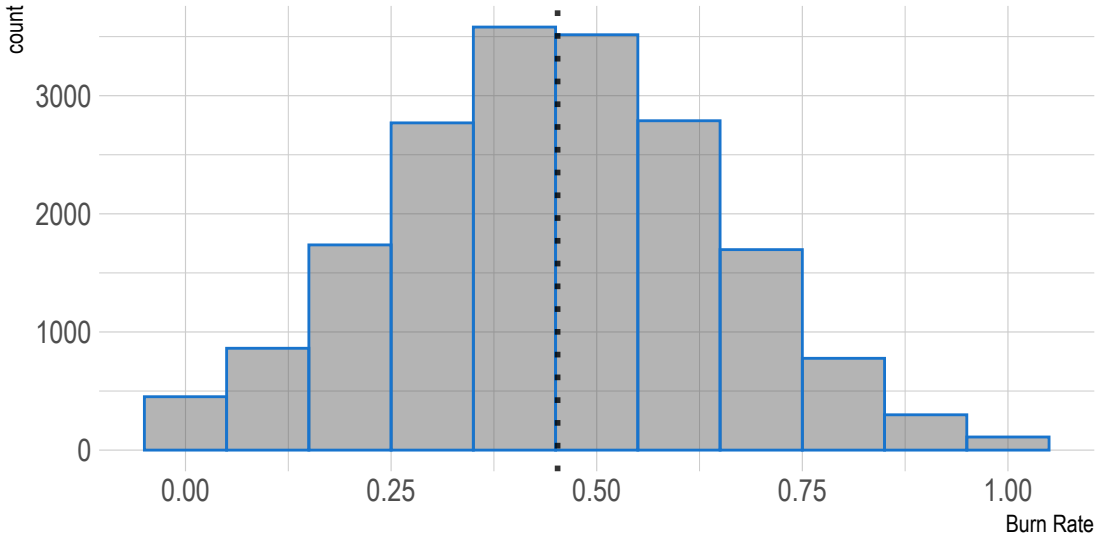
Exploratory Data Analysis (EDA)

Aside from the **Employee ID** variable, our dataset comprises **8 variables** we need to explore.

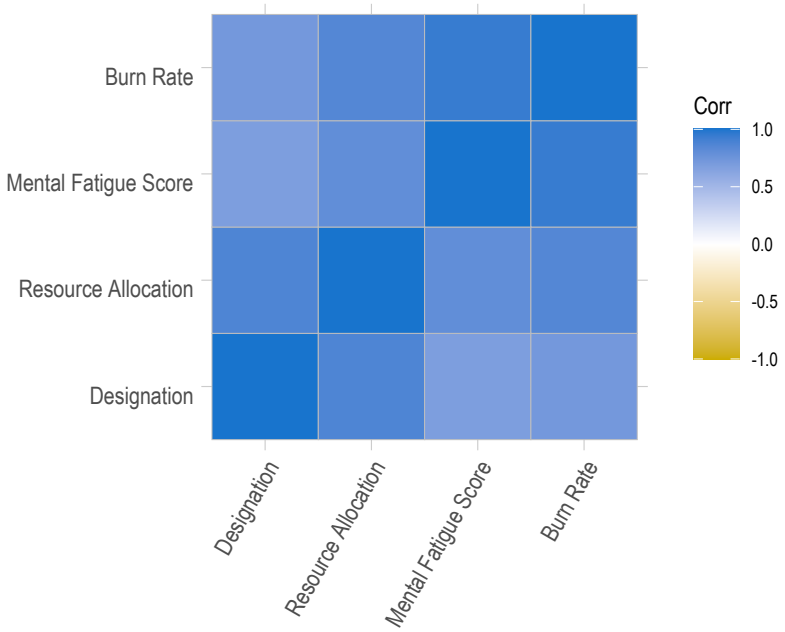
We start with **Burn Rate**, a numerical discrete variable where values span a range from 0.00 to 1.00.



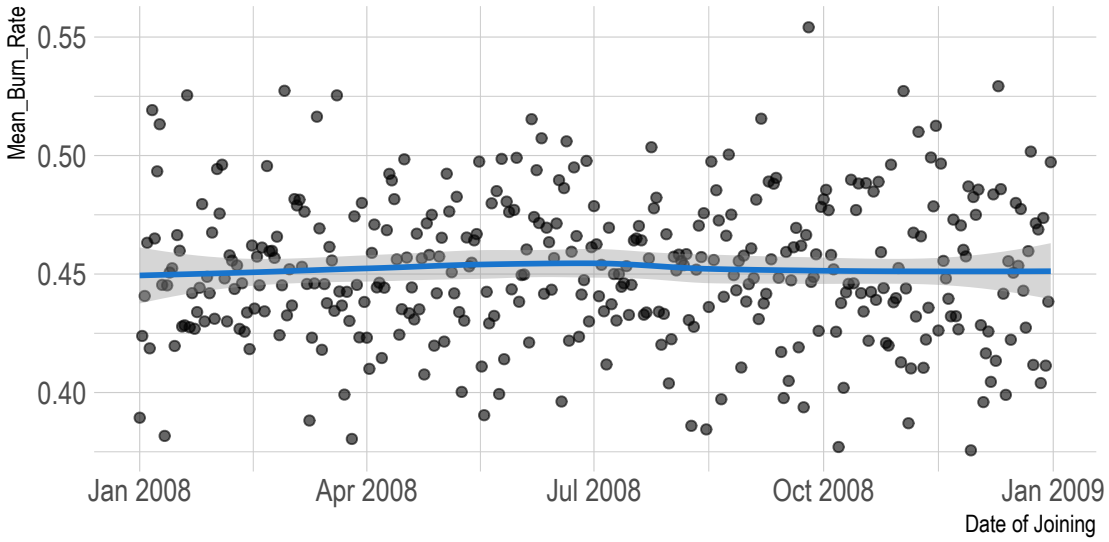
Now, by using a binwidth of 0.1, we plot another histogram to get a better understanding of **Burn Rate distribution**, which is approximately **normal**, with **mean 0.4524443** and **sd 0.1978476**.



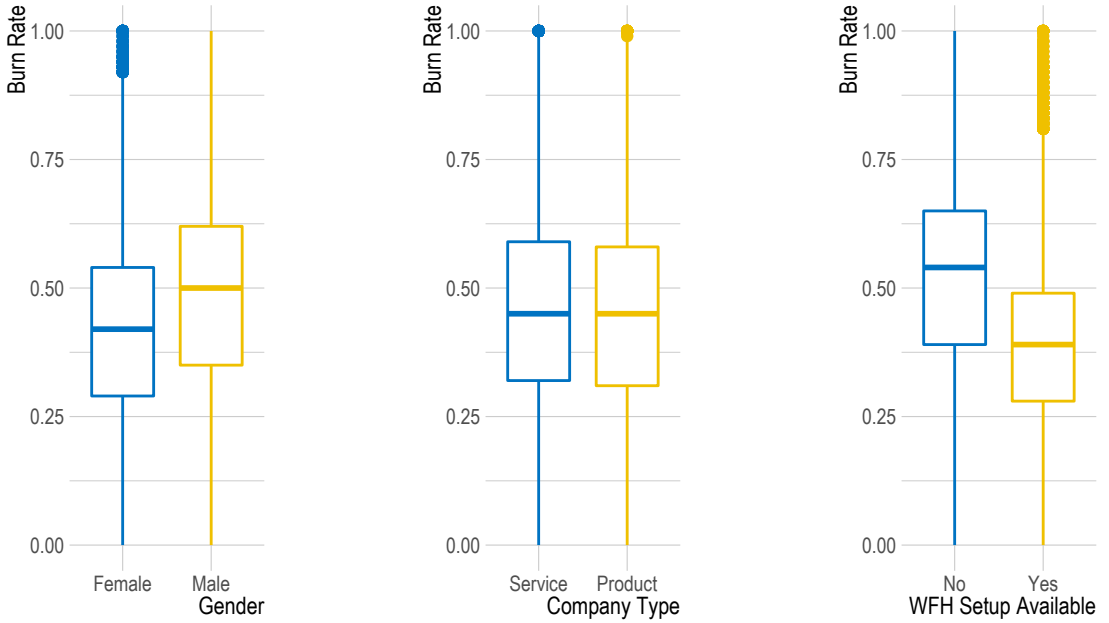
Then, we have our **independent variables**. Three of them, **Gender**, **Company Type** and **WFH Setup Available** are categorical, with only two levels. Three of them, **Designation**, **Resource Allocation**, and **Mental Fatigue Score**, are numerical with discrete values, and all are **positively correlated** with **Burn Rate**. Specifically, **Mental Fatigue Score**, has a remarkably high correlation coefficient, **0.9437024**.



The remaining variable, **Date of Joining**, has a very short range, implying that all observed employees were surveyed only in year 2008. Our assumption is that this is too short of a span to have a meaningful, causal relationship with employees burnout rate. We confirm this by graphing a scatter plot of means of **Burn Rate** values (grouped by Date of Joining) and by fitting a **loess** regression line through them.



Now we proceed by exploring the relationship of our **categorical variables** with **Burn Rate** by using a series of **box plots**.



We notice that both **Gender** and **WHF** appear to affect employees' burnout rate. The **Company Type** seems to be less relevant, and yet we will include it anyway, since we are going to use methods that are capable of accounting for its poor influence on our depended variable.

We also notice the presence of a series of **outliers** - as we did for our numerical variables in a series of box plots we decided not to display. Thus, we may be tempted to tackle them by using techniques such as **robust scaling**, which requires our variables to be converted to numerical. However, prior to writing this report, we did some trials using **one-hot encoding**, and the marginal improvements obtained in models such as linear regression were outrun by the drawbacks we obtained in other models, such as Random Forest.

Thus, additionally considering the short range of our values, we will thereby either use **standardization** for our numerical variables, or algorithms that are inherently more **robust** to the effect of outliers.

Preprocessing

Before we proceed to actually building our models, we need to exclude **Date of Joining** and re-code the names of the other columns, since they appear to be in a format that might prove problematic for some of the algorithms - Knn in particular.

```
#DATA FORMATTING

#Drop Id and Date of Joining variables
newdat_good <- newdat[,3:9] %>% as.data.frame()

#Re-code dataframe names
newnames <- c("Gender", "Company_Type",
              "WFH_Setup_Available", "Designation", "Resource_Allocation",
              "Mental_Fatigue_Score", "Burn_Rate")

oldnames <- c("Gender", "Company Type",
             "WFH Setup Available", "Designation", "Resource Allocation",
             "Mental Fatigue Score", "Burn Rate")

t <- newdat_good %>%
  rename_with(~ newnames[which(oldnames == .x)], .cols = oldnames)
```

Then, we are ready to **split our** data into **train** and **test** set. We use a conservative **20/80 %** split, since, despite ending up with a somewhat small train set, we still want to have a relevant portion of the data to properly evaluate the **flexibility** of our models. As previously stated, we will later use **repeated k-fold cross validation** on the train set to tackle the limitations imposed by its size.

```
#DATA SPLITTING

#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Create test index for data splitting
tst_index <- createDataPartition(t$Burn_Rate, times = 1, p = 0.2, list = FALSE)

#Create train set
train_set <- t %>% slice(-tst_index)

#Create test set
test_set <- t %>% slice(tst_index)
```

Now, we still have to preprocess our **categorical variables**. We could use one hot encoding for this specific purpose, but, as previously stated, we rather opt for leveraging R functionality by converting our categorical values to **factors**.

```
#PRE-PROCESSING

#Re-code categorical variables

#Show columns class
sapply(train_set, class)

#Convert characters to factors (train set)
temp <- map_df(train_set[,1:3], as.factor)

train_set[,1:3] <- temp[,1:3]

#Convert characters to factors (test set)
temp1 <- map_df(test_set[,1:3], as.factor)

test_set[,1:3] <- temp1[,1:3]
```

Models

1 Linear Regression

We are finally ready to build our models. Given the high correlation between our dependent variable and **Mental Fatigue Score** we decide to start with a simple **linear regression**. From now on, given the distribution of the values in our dependent numerical variable, we will be using **root-mean-square error (RMSE)** as our **evaluation metric**.

```
#Linear Regression

#Set seed
set.seed(1234, sample.kind = "Rounding")

#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Train linear model
lm_fit <- train(Burn_Rate ~ Mental_Fatigue_Score, method = "lm",
                preProcess = c("center", "scale"), data = train_set,
                trControl = control)

#Predict with linear model
lm_pred <- predict(lm_fit, test_set)

#Linear model RMSE
RMSE(lm_pred, test_set$Burn_Rate)

#Linear model CV results
lm_fit$results
```

We then create a **results table** that stores both the **RMSES** from our test set and the mean of those resulting from our **repeated 10 folds cross validation** on the train set. We also include a **Tune** column, which will help us keep track of the tuning process we will be undertaking.

```
#Create results table
results <- data.frame(Method = "1 Linear Regression", Tune = "/",
                      RMSE_CV = 0.06508212, RMSE_TEST = 0.06495063)

results %>% knitr::kable()
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506

For clarity, from now on, we will be showing only the lines of code we consider to be particularly relevant, as well as commenting only the coding and tuning methods that were useful in the process of building our final algorithm. For the full code, readers can refer to the **Appendix**.

2 Multivariate Linear Regression

Thus, we can proceed with a **multivariate linear regression**, for which - since not perfectly collinear between them - we use **all the predictors** at our disposal.

```
#Train lm multivariate
lm_fit_multivariate <- train(Burn_Rate ~ ., method = "lm",
                             preProcess = c("center", "scale"),
                             data = train_set, trControl = control)
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
2 Multivariate Linear Regression	/	0.0557647	0.0556402

Our **RMSE** did improve, but we can probably achieve better results by using more complex models.

3 K Nearest Neighbors

We start with **K Nearest Neighbors (KNN)**. Firstly, in order to get a rough idea of its behavior, we implement **knn** using **caret** default tuning and bootstrapping validation options. Additionally we **do not standardize** our data, since, for KNN, it led to worse results on both our test set and across our cross validation folds.

```
#Train KNN
knn_fit <- train(Burn_Rate ~ ., method = "knn", data = train_set)
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
2 Multivariate Linear Regression	/	0.0557647	0.0556402
3 K-nearest Neighbors	K = 9	0.0572474	0.0542313

4 K Nearest Neighbors (Optimized)

Building on the best tune of the previous model, we fit **knn** a second time, now testing values of **K** between **8** and **170** and repeating **10 fold cross validation 4 times** in order to pick the optimal one.

```
#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Specify tuning parameters
tunegrid <- data.frame(k = 8:170)

#Train KNN 2nd
knn_fit2 <- train(Burn_Rate ~ ., method = "knn", data = train_set,
                  tuneGrid = tunegrid, trControl=control)
```

From the plot above, we notice that the **RMSE** reached a minimum at **K = 91**, and then started to increase again. We thereby consider our **knn** algorithm to be optimal, and consequently add a new entry to our results table.

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
4 K-nearest Neighbors (Optimized)	K = 91	0.0545393	0.0543886

5 Locally weighted regression (Loess)

Now, before proceeding to employing more complex algorithms, such as Random Forests, we will approach a simpler method, which is known to perform well with small-range data as ours, namely **locally weighted regression**. Its functioning is not dissimilar from **KNN**, but it is much quicker to train. In caret, it is also easy to implement, so we run it via the argument method **loess** and tune it in order to pick the best value for **span** (which is specified as a proportion). We also scale and center our numerical data, since it leads to marginally improved results.

```
#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Specify tuning parameters
tunegrid <- expand.grid(span = c(0.2, 0.3, 0.4, 0.5, 0.6), degree = 1)

#Loess fit
loess_fit <- train(Burn_Rate ~ ., method = "gamLoess", tuneGrid = tunegrid,
                   preProcess = c("center", "scale"),
                   trControl = control, data = train_set)
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
4 K-nearest Neighbors (Optimized)	K = 91	0.0545393	0.0543886
5 Locally Weighted Regression	Span = 0.2	0.0545462	0.0543843

We can see how the results from **Loess** are virtually identical to those of **KNN**, despite it being much **quicker** to train and simpler to optimize.

6 Random Forests

Regression trees, generally speaking, might be one of the obvious choices for our task, since they are expected to perform well with variables that are both numerical and categorical, in most of the cases, capable of dealing with both highly collinear values and outliers.

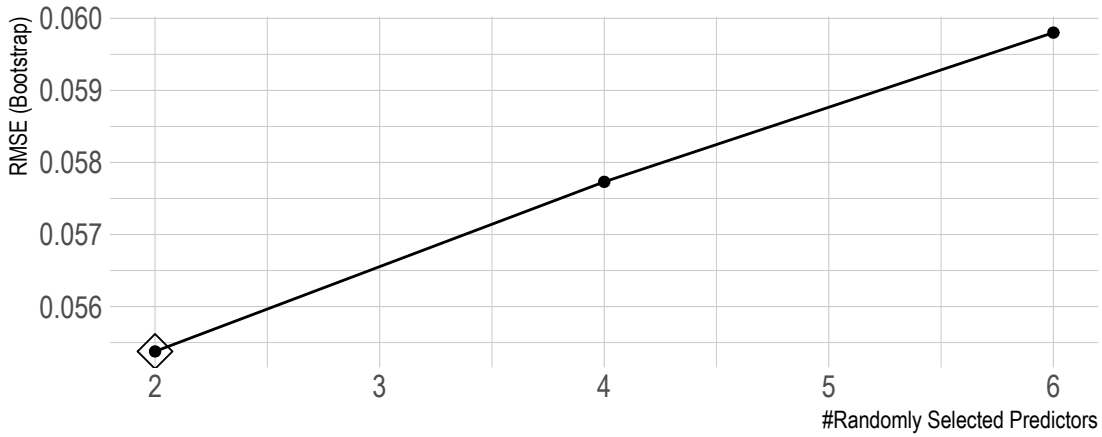
However, the main problem we are faced with is the small size of data set, where regression trees might quickly start to overfit, thereby producing highly biased models. **Random Forests**, on the other hand, are specifically designed to solve these very issues, and, therefore, will prove to be an effective tool that we are now going to implement.

As with **KNN**, we first start running caret's random forest **rf** with the default tuning and training control parameters.

```
#Train rf
rf_fit <- train(Burn_Rate ~ ., method = "rf", data = train_set)
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
4 K-nearest Neighbors (Optimized)	K = 91	0.0545393	0.0543886
5 Locally Weighted Regression	Span = 0.2	0.0545462	0.0543843
6 Random Forests	Mtry = 2, Ntree = 500	0.0553769	0.0550221

To our initial surprise, we see that **rf** didn't performed as well as either **knn** or **loess**. However, this model is still sub-optimal, and was tuned using caret bootstrapping default control, which considering, the size of our train set, might lead to misleading outcomes. Thus, we proceed by accessing the training results more in detail, visualizing components such as **variable importance** and **plotting** the resulting **RMSEs** against the mtry values that produced them.



Random Forests (Optimized)

Hence, we proceed by effectively tuning our **random forests** model. We apply 10 folds cross validation 4 times, train across all possible values of mtry, and increase the number of trees to grow to 1000.

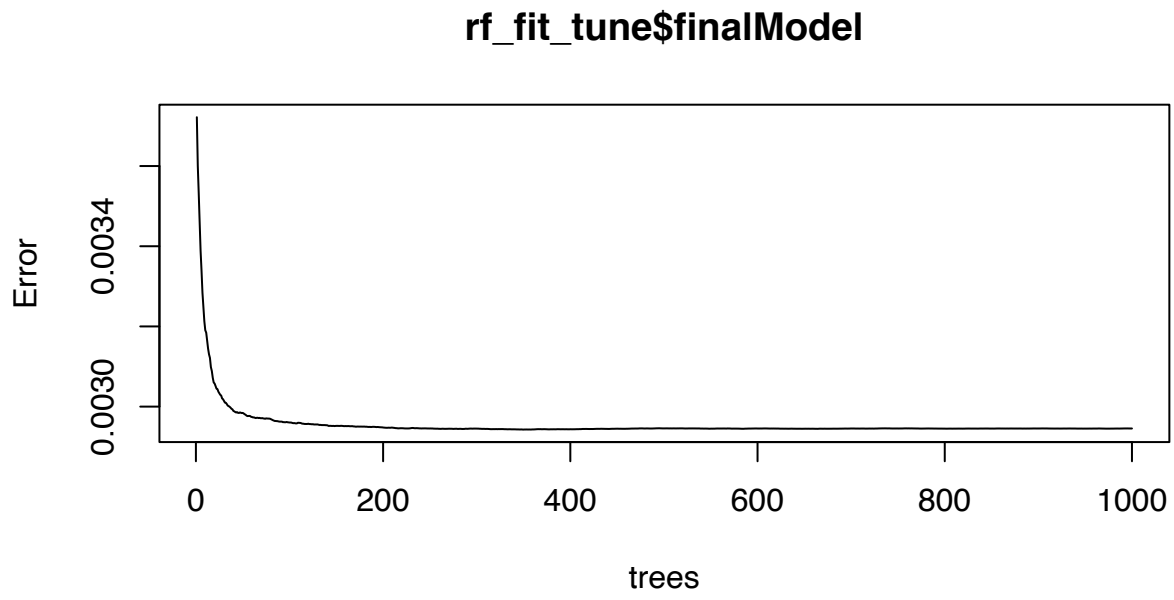
```
#Define train control
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10,
                        search = "grid")

#Define train grid
tunegrid <- data.frame(mtry = 1:6)

#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Train rf 2nd
rf_fit_tune <- train(Burn_Rate ~ ., method = "rf",
                  data = train_set, ntree = 1000, tuneGrid = tunegrid,
                  trControl=control)
```

```
#Predict with rf 2nd
rf_pred_tune <- predict(rf_fit_tune, test_set)
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
4 K-nearest Neighbors (Optimized)	K = 91	0.0545393	0.0543886
5 Locally Weighted Regression	Span = 0.2	0.0545462	0.0543843
6 Random Forests	Mtry = 2, Ntree = 500	0.0553769	0.0550221
7 Random Forests (Optimized)	Mtry = 3, Ntree = 1000	0.0543596	0.0537501

We can see that this last model did indeed outperform the previous one - but also, **knn** and **loess**. This is mainly due to the fact that, by implementing repeated **k-fold cross validation**, we were able to choose a better value for **mtry**, which eventually led to a **more flexible model**.

Eventually, we plot errors against Ntree and see that, by the time **rf** grew the specified number of trees (Ntree = 1000), errors did in fact **stabilize**.



We could try to grow an even larger number of trees, and use repeated k-fold cv to choose the best value among them, but the marginal improvements that we may - or may not - achieve, will be surely overshadowed by the cumbersome computational cost of such a process.

We therefore consider our **Random Forests** model to be optimized, and move on to create an **ensemble** of our best performing algorithms.

7 Ensembles

Ensemble algorithms, have gradually become a standard in many machine learning applications. The core idea is essentially that of **combining different ML methods** in order to improve the flexibility of the final

model, and - therefore - performances when applying it to unobserved new data. There are many techniques and methods for creating ensembles, which may vary significantly depending on the fields of application and the characteristics of the data they are confronted with.

In our case, given the low dimensionality of our data set, and the relatively simple task we are called upon to perform, we won't venture in complex ensemble methods. We will rather **average the predictions** of the algorithms we are going to combine.

However, in order to choose the **best performing ensemble**, it would be unwise to simply use our test set. We will then use **repeated k-fold cross validation** on the train set to do so, although that will require us to program a series of functions, specifically tailored to fulfill our task.

The code is rather convoluted, so we decide to show the full steps.

We start by creating **three functions** for each one of the combinations we are going to train. The first one comprises predictions from our three best-performing algorithms (**rf**, **knn** and **loess**). Then, we progressively add **multivariate linear regression** and **linear regression**.

```
#ENSEMBLES

#Create ensemble functions

ensemble1 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(loess_fit, ts)) / 3
}

ensemble2 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(lm_fit_multivariate, ts) +
     predict(loess_fit, ts)) / 4
}

ensemble3 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(lm_fit, ts) +
     predict(lm_fit_multivariate, ts) +
     predict(loess_fit, ts)) / 5
}
```

Then, we proceed by creating an **ensemble list**, which we will later feed to sapply().

```
#Create ensemble list
ensembles_list <- list(ensemble1, ensemble2, ensemble3)
```

And by programming a **repeated (4 times) 10 fold cross validation** function using our train set.

```
#Create repeated (4times) 10 fold cross validation function for our ensembles

repeatedcv_ensembles <- function(ensemblef) {
  mean(replicate(4, {

    folds <- createFolds(train_set$Burn_Rate, k = 10, list = TRUE,
                         returnTrain = TRUE)
    res <- vector("numeric", 10)
    for (i in 1:10) {
      tr <- train_set[folds[[i]],]
      ts <- train_set[-folds[[i]],]

      errors <- RMSE(ensemblef(ts), ts[,7])
      res[i] <- errors

    }
    mean(res)
  }
  )
  )
}
```

Now we are ready to set the seed and apply this function to all 3 ensembles we created.

```
#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Apply repeated cross validation function to ensembles
sapply(ensembles_list, repeatedcv_ensembles)
```

```
## [1] 0.05046548 0.05141923 0.05239122
```

The **first ensemble** - the one comprising rf, knn, and loess - was the best performing one.

Finally, we can re-run it on the test set and add the resulting **RMSE** to our final results table.

```
#Re-run ensemble one on the test set
ensemble1(test_set) %>% RMSE(., test_set$Burn_Rate)
```

```
## [1] 0.05296494
```

Method	Tune	RMSE_CV	RMSE_TEST
1 Linear Regression	/	0.0650821	0.0649506
4 K-nearest Neighbors (Optimized)	K = 91	0.0545393	0.0543886
5 Locally Weighted Regression	Span = 0.2	0.0545462	0.0543843
6 Random Forests	Mtry = 2, Ntree = 500	0.0553769	0.0550221
7 Random Forests (Optimized)	Mtry = 3, Ntree = 1000	0.0543596	0.0537501
8 Ensemble	Knn + Rf + Loess	0.0504655	0.0529649

Results

The last model we presented, namely the **ensemble** that combined our optimized **K-Nearest Neighbors** , **Random Forests**, and **Locally Weighted Regression** algorithms, is the one we pick as our **final model**. Of course, our decision is determined by the fact that it outperformed all the previous models we had built, on both the **test set** and across the folds of our **repeated k-fold cross validation**. However, some considerations are in order.

The first is that, despite achieving the best result, the final **RMSE** did not drop by a truly significant margin in our validation set. In other words, it did perform better then other models, but, considering the task at hands, it might not be a slight **improvement in the thousandths decimal place** to really make the difference between employees that are likely to burnout and employees who are not.

Employers might thus decide to opt for other algorithms, which are either easier to implement and interpret or simply **quicker** to run. In our final ensemble, for instance, an important role was played by an **rf** algorithm that took our Macbook Pro several hours to train. Other users might want to avoid such computational cost.

Accordingly, we think that a perfect candidate for such instances is **Locally Weighted Regression**, an algorithm that, in its caret implementation, was both quick and easy to train. Of course, future edits and applications will be limited to data that is very similar to ours. It is - in fact - primarily because of both the very low dimensionality and short range of our observed variables that **loess** performed very well. If, for instance, a survey with continuous and wider-range values for variables such as Mental Fatigue Score were to be conducted, then other methods would likely have to be considered.

Conclusion

Our final model achieved a low **RMSE**, namely **0.0529649**. However, the biggest role in driving up the quality of our results was played by an independent variable, **Mental Fatigue Score** which is very highly correlated with our outcome, **Burn Rate**. In fact, only by implementing a simple linear regression using the aforementioned variable we obtained an **RMSE** of 0.0649506.

Mental Fatigue Score is, indeed, a variable that, for the general public, might be almost tautologically associated with burnout, and, additionally, one that is not easy to report by employees themselves. Some employee, in fact, may not be able to identify it in a relevant manner, let alone rating it in a decimal scale.

Thus, a more interesting challenge might be that of predicting the **Mental Fatigue Score** of an employee given the variables in our dataset - **Burnout Rate**, obviously, excluded. We actually did a quick trial to get an idea of the feasibility of this approach, and, by implementing **KNN** with a swift sub-optimal tuning, we obtained an **RMSE** of **1.129071**. Considering that **Mental Fatigue Score** has a ten times bigger range of **Burnout Rate** (0 - 10 and 0 - 1, respectively) this tells us that we might be on the right track, and that, by making use of a better tuning and more effective algorithms, might actually achieve a series of satisfactory results.

In conclusion, if we were to actually employ **ML** as a practical tool to tackle the burnout in the workplace, two main actions would still be required. Firstly, a **much larger study** would need to be undertaken, in order to build a more relevant sample size and a more extensive data set - with a specific focus on constructing an actually reliable date of joining variable. Lastly, a **cutoff** value for **Burnout** (or **Mental Fatigue Score**) should be clearly established, in order to make employers able to implement practical policies, such as recommended holiday or mental health counseling. In this way, our **ML** task would become a **classification problem**, and we would be able to actually inform effective decision-making.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE,
                      eval = TRUE, error = FALSE, cache = TRUE,
                      dev = "cairo_pdf")
library(hrbrthemes)
library(ggplot2)
library(Cairo)
library(extrafont)
extrafont::loadfonts()
#IMPORTING DATASET

#Identify data file
filename <- "train.csv"
dir <- "/Users/jacopotrabona/Downloads/burnout/upload"
fullpath <- file.path(dir, filename)

#Copy data file into project directory
file.copy(fullpath, "train.csv")

#Load data set
if(!require(tidyverse)) install.packages("tidyverse")
library(tidyverse)

dat <- read_csv("train.csv")

#REQURED PACKAGES

#Install required packages
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(skimr)) install.packages("skimr")
if(!require(hrbrthemes)) install.packages("hrbrthemes")
if(!require(ggcorrplot)) install.packages("ggcorrplot")
if(!require(gam)) install.packages("gam")
if(!require(randomForest)) install.packages("randomForest")
if(!require(ggpubr)) install.packages("ggpubr")

#Load required packages
library(tidyverse)
library(caret)
library(skimr)
library(ggcorrplot)
library(randomForest)
library(hrbrthemes)
library(ggpubr)
#Head of the dataset
dat %>% select(!`Employee ID`) %>% head() %>% knitr::kable()


#Number of NAs across columns
#Identify NAs
is.na(dat)
```

```r
#Number of NAs
sum(is.na(dat))

#NAs are 20.31648 % of our data
(4622 * 100) / 22750

#Number of NAs across columns
colSums(is.na(dat))

#Remove observations with NAs in the Burn Rate column
newdat <- dat[!is.na(dat$`Burn Rate`),]
#Columns with NAs
colSums(is.na(newdat))
#Columns with NAs
colSums(is.na(newdat))

#Number of observations with NAs in both Resource Allocation and Mental Fatigue
is.na(newdat) %>%
  as.data.frame() %>%
  filter(`Resource Allocation` == TRUE & `Mental Fatigue Score` == TRUE) %>%
  nrow()

#Observations still containing missing values are approx. 14%
((3223 - 187) * 100) / 21626

#Correlation between Burn Rate and Mental Fatigue Score
cor(newdat$`Mental Fatigue Score`, newdat$`Burn Rate`, use = "complete.obs",
    method = "spearman")

#Correlation between Burn Rate and Resource Allocation
cor(newdat$`Resource Allocation`, newdat$`Burn Rate`, use = "complete.obs",
    method = "spearman")
#Omit row containing missing values
newdat <- na.omit(newdat)
#Dataset summary statistics
summary(newdat)

newdat %>% ggplot(aes(`Burn Rate`)) +
  geom_histogram(binwidth = 0.01, colour = "dodgerblue3",
                 alpha = 0.55) +
  theme_ipsum()
#Burn Rate distribution (0.1)
newdat %>% ggplot(aes(`Burn Rate`)) +
  geom_histogram(binwidth = 0.1, colour = "dodgerblue3", alpha = 0.45) +
  geom_vline(xintercept = 0.4524443, alpha = 0.8, size = 1,
             linetype = "dotted", color = "black") +
  theme_ipsum()

#Mental Fatigue Score
newdat %>% ggplot(aes(`Mental Fatigue Score`, `Burn Rate`)) +
  geom_boxplot(alpha=0.5) +
  geom_jitter(color="black", size = 0.4, alpha = 0.05) +
  theme_ipsum() +
```

```
  scale_fill_brewer(palette="Dark1") +
  theme(legend.position = "none")


#Resource Allocation
newdat %>% ggplot(aes(`Resource Allocation`, `Burn Rate`)) +
  geom_boxplot(alpha=0.5) +
  geom_jitter(color="black", size = 0.4, alpha = 0.05) +
  theme_ipsum() +
  scale_fill_brewer(palette="Dark1") +
  theme(legend.position = "none")


#Designation
newdat %>% ggplot(aes(Designation, `Burn Rate`)) +
  geom_boxplot(alpha=0.5) +
  geom_jitter(color="black", size = 0.4, alpha = 0.05) +
  theme_ipsum() +
  scale_fill_brewer(palette="Dark1") +
  theme(legend.position = "none")
#Study of correlations

#Correlations between continuous variables
correlations <- cor(newdat[,6:9], use = "pairwise.complete",
                    method = "spearman")
#Correlations Plot
ggcorrplot(corr = correlations, colors = c("gold3", "white",
                                           "dodgerblue3")) +
  theme_ipsum() +
  theme(axis.text.x = element_text(angle = 60, hjust = 1, vjust = 1, size = 9),
        axis.text.y = element_text(hjust = 1, vjust = 1, size = 9),
        legend.title = element_text(size = 9),
        legend.text = element_text(size = 7),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
#Date of Joining means
newdat %>% group_by(`Date of Joining`) %>%
  summarise(Mean_Burn_Rate = mean(`Burn Rate`)) %>%
  ggplot(aes(`Date of Joining`, Mean_Burn_Rate)) +
  geom_point(alpha = 0.6) +
  geom_smooth(color = "dodgerblue3") +
  theme_ipsum()
#Gender
gender_bxp <- ggboxplot(newdat, x = "Gender", y = "Burn Rate",
                        color = "Gender", palette = "jco") +
  theme_ipsum(base_size = 8, caption_size = 6) + theme(legend.position = "none")

gender_bxp

#Company type
company_bxp <- ggboxplot(newdat, x = "Company Type", y = "Burn Rate",
                        color = "Company Type", palette = "jco") +
  theme_ipsum(base_size = 8, caption_size = 6) + theme(legend.position = "none")
```

```r
company_bxp

#WFH
WFH_bxp <- ggboxplot(newdat, x = "WFH Setup Available", y = "Burn Rate",
                     color = "WFH Setup Available", palette = "jco") +
  theme_ipsum(base_size = 8, caption_size = 6) + theme(legend.position = "none")

WFH_bxp
ggarrange(gender_bxp, company_bxp, WFH_bxp + theme_ipsum(base_size = 8,
                                                         caption_size = 6),
          legend = "none",
          ncol = 3, nrow = 1)


#DATA FORMATTING

#Drop Id and Date of Joining variables
newdat_good <- newdat[,3:9] %>% as.data.frame()

#Re-code dataframe names
newnames <- c("Gender", "Company_Type",
              "WFH_Setup_Available", "Designation", "Resource_Allocation",
              "Mental_Fatigue_Score", "Burn_Rate")

oldnames <- c("Gender", "Company Type",
              "WFH Setup Available", "Designation", "Resource Allocation",
              "Mental Fatigue Score", "Burn Rate")

t <- newdat_good %>%
  rename_with(~ newnames[which(oldnames == .x)], .cols = oldnames)
#DATA SPLITTING

#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Create test index for data splitting
tst_index <- createDataPartition(t$Burn_Rate, times = 1, p = 0.2, list = FALSE)

#Create train set
train_set <- t %>% slice(-tst_index)

#Create test set
test_set <- t %>% slice(tst_index)

#PRE-PROCESSING

#Re-code categorical variables

#Show columns class
sapply(train_set, class)

#Convert characters to factors (train set)
temp <- map_df(train_set[,1:3], as.factor)
```

```r
train_set[,1:3] <- temp[,1:3]

#Convert characters to factors (test set)
temp1 <- map_df(test_set[,1:3], as.factor)

test_set[,1:3] <- temp1[,1:3]

#Linear Regression

#Set seed
set.seed(1234, sample.kind = "Rounding")

#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Train linear model
lm_fit <- train(Burn_Rate ~ Mental_Fatigue_Score, method = "lm",
                preProcess = c("center", "scale"), data = train_set,
                trControl = control)

#Predict with linear model
lm_pred <- predict(lm_fit, test_set)

#Linear model RMSE
RMSE(lm_pred, test_set$Burn_Rate)

#Linear model CV results
lm_fit$results
#Create results table
results <- data.frame(Method = "1 Linear Regression", Tune = "/",
                      RMSE_CV = 0.06508212, RMSE_TEST = 0.06495063)

results %>% knitr::kable()
#Multivariate Linear Regression

#Set seed
set.seed(1234, sample.kind = "Rounding")

#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)
#Train lm multivariate
lm_fit_multivariate <- train(Burn_Rate ~ ., method = "lm",
                             preProcess = c("center", "scale"),
                             data = train_set, trControl = control)
#Predict lm mutivariate  fit
lm_pred_multivariate <- predict(lm_fit_multivariate, test_set)

#Lm multivariate RMSE
RMSE(lm_pred_multivariate, test_set$Burn_Rate)

#CV results
lm_fit_multivariate$results
#Add row to results table
```

```r
results <- results %>% add_row(Method = "2 Multivariate Linear Regression",
                               Tune = "/", RMSE_CV = 0.05576466,
                               RMSE_TEST = 0.05564016)

results %>% knitr::kable()
#KNN (Default Tune)

#Set seed
set.seed(1234, sample.kind = "Rounding")
#Train KNN
knn_fit <- train(Burn_Rate ~ ., method = "knn", data = train_set)
#Predict with KNN
knn_pred <- predict(knn_fit, test_set)

#KNN RMSE
RMSE(knn_pred, test_set$Burn_Rate)

#Access model fit
knn_fit

#Plot k results
ggplot(knn_fit, highlight = TRUE) +
  theme_ipsum()

#CV results
knn_fit$results
#Add row to results table
results <- results %>% add_row(Method = "3 K-nearest Neighbors",
                               Tune = "K = 9", RMSE_CV = 0.05724738,
                               RMSE_TEST = 0.05423132)

results %>% knitr::kable()
#Optimized KNN

#Set seed
set.seed(1234, sample.kind = "Rounding")
#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Specify tuning parameters
tunegrid <- data.frame(k = 8:170)

#Train KNN 2nd
knn_fit2 <- train(Burn_Rate ~ ., method = "knn", data = train_set,
                  tuneGrid = tunegrid, trControl=control)
#Predict with KNN 2nd
knn_pred2 <- predict(knn_fit2, test_set)

#KNN 2nd RMSE
RMSE(knn_pred2, test_set$Burn_Rate)

#Access model results
knn_fit2$results
```

```r
knn_fit2$bestTune

knn_fit2$finalModel
load("~/projects/burnout/upload/burnout-code-final.RData")
#Plot k results
ggplot(knn_fit2, highlight = TRUE) +
  theme_ipsum()
#Add row to results table
results <- results %>% add_row(Method = "4 K-nearest Neighbors (Optimized)",
                               Tune = "K = 91", RMSE_CV = 0.05453934 ,
                               RMSE_TEST = 0.05438856)

results %>% knitr::kable()
#Locally Weighted Regression (loess)

#Set seed
set.seed(1234, sample.kind = "Rounding")
#Specify train controls
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10)

#Specify tuning parameters
tunegrid <- expand.grid(span = c(0.2, 0.3, 0.4, 0.5, 0.6), degree = 1)

#Loess fit
loess_fit <- train(Burn_Rate ~ ., method = "gamLoess", tuneGrid = tunegrid,
                   preProcess = c("center", "scale"),
                   trControl = control, data = train_set)
#Predict lm mutivariate  fit
lm_pred_multivariate <- predict(lm_fit_multivariate, test_set)

#Lm multivariate RMSE
RMSE(lm_pred_multivariate, test_set$Burn_Rate)

#CV results
lm_fit_multivariate$results
#Add row to results table
results <- results %>% add_row(Method = "5 Locally Weighted Regression",
                               Tune = "Span = 0.2", RMSE_CV = 0.05454619,
                               RMSE_TEST = 0.0543843)

results %>% knitr::kable()
#Random Forests (rf)

#Set.seed
set.seed(1234, sample.kind = "Rounding")
#Train rf
rf_fit <- train(Burn_Rate ~ ., method = "rf", data = train_set)
#Predict with rf
rf_pred <- predict(rf_fit, test_set)

#rf RMSE
RMSE(rf_pred, test_set$Burn_Rate)
#Add row to results table
```

```r
results <- results %>% add_row(Method = "6 Random Forests",
                               Tune = "Mtry = 2, Ntree = 500",
                               RMSE_CV = 0.05537689, RMSE_TEST = 0.05502211)

results %>% knitr::kable()
#Access model results
rf_fit$results

rf_fit$resample

rf_fit$bestTune

rf_fit$finalModel
#Access model's variable importance
varImp(rf_fit, scale = FALSE)
#Plot mtryies against RMSEs
ggplot(rf_fit, highlight = TRUE) +
  theme_ipsum()
#Optimized Random Forest
#Define train control
control <- trainControl(method = "repeatedcv", p = .2, repeats = 4, number = 10,
                        search = "grid")

#Define train grid
tunegrid <- data.frame(mtry = 1:6)

#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Train rf 2nd
rf_fit_tune <- train(Burn_Rate ~ ., method = "rf",
                 data = train_set, ntree = 1000, tuneGrid = tunegrid,
                 trControl=control)

#Predict with rf 2nd
rf_pred_tune <- predict(rf_fit_tune, test_set)

#rf 2nd RMSE
RMSE(rf_pred_tune, test_set$Burn_Rate)
#Add entry to results table
results <- results %>% add_row(Method = "7 Random Forests (Optimized)",
                               Tune = "Mtry = 3, Ntree = 1000",
                               RMSE_CV = 0.05435957, RMSE_TEST = 0.0537501)

results %>% knitr::kable()

quartzFonts(avenir = c("Avenir Book", "Avenir Black", "Avenir Book Oblique",
                       "Avenir Black Oblique"))

par(family = 'avenir')
plot(rf_fit_tune$finalModel)
#ENSEMBLES
```

```r
#Create ensemble functions

ensemble1 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(loess_fit, ts)) / 3
}

ensemble2 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(lm_fit_multivariate, ts) +
     predict(loess_fit, ts)) / 4
}

ensemble3 <- function(ts){
  (predict(rf_fit_tune, ts) +
     predict(knn_fit2, ts) +
     predict(lm_fit, ts) +
     predict(lm_fit_multivariate, ts) +
     predict(loess_fit, ts)) / 5
}
#Create ensemble list
ensembles_list <- list(ensemble1, ensemble2, ensemble3)
#Create repeated (4times) 10 fold cross validation function for our ensembles

repeatedcv_ensembles <- function(ensemblef) {
  mean(replicate(4, {

    folds <- createFolds(train_set$Burn_Rate, k = 10, list = TRUE,
                         returnTrain = TRUE)
    res <- vector("numeric", 10)
    for (i in 1:10) {
      tr <- train_set[folds[[i]],]
      ts <- train_set[-folds[[i]],]

      errors <- RMSE(ensemblef(ts), ts[,7])
      res[i] <- errors

    }
    mean(res)
  }
  )
  )
}
#Set.seed
set.seed(1234, sample.kind = "Rounding")

#Apply repeated cross validation function to ensembles
sapply(ensembles_list, repeatedcv_ensembles)
#Re-run ensemble one on the test set
ensemble1(test_set) %>% RMSE(., test_set$Burn_Rate)
#Result table
```

```r
results <- results %>% add_row(Method = "8 Ensemble",
                               Tune = "Knn + Rf + Loess",
                               RMSE_CV = 0.05046548, RMSE_TEST = 0.05296494)

results %>% knitr::kable()
```