Machine Learning with R-tidymodels: recipe optimization

Lausanne, 26.06.2022

30 min reading

Introduction

This article investigates how to optimize a Concrete recipe to obtain maximal compressive strength. It provides an opportunity to test statistical learning techniques for modeling the relation between the output (Compression Strength) and various inputs (e.g. Cement, Water) and to assess the feasibility of a simple search approach for finding an optimal result.

The language used is R and several well tested libraries for data science and modelling (tidyverse and tidymodels).

This article is based on the case study *Compressive Strength of Concrete* from the book Applied Predictive Modeling by Max Kuhn (2021). The data for this analysis is obtained for the companion R package of the book.

Methods

Setup

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------------------------------------- tidyverse 1.3.
```

```
v tibble  3.1.6     v stringr 1.4.0
v purrr   0.3.4
```

```
-- Conflicts ------------------------------------------------------------------------ tidyverse_conflicts()
x scales::col_factor() masks readr::col_factor()
x purrr::discard()     masks scales::discard()
x dplyr::filter()      masks stats::filter()
x dplyr::lag()         masks stats::lag()
```

```
library(tidymodels)
```

```
Registered S3 method overwritten by 'tune':
  method                   from
  required_pkgs.model_spec parsnip
```

```
-- Attaching packages --------------------------------------------------------------------- tidymodels 0.1.4
```

```
v broom        0.8.0      v rsample      0.1.0
v dials        0.0.10     v tune         0.1.6
v infer        1.0.0      v workflows    0.2.4
v modeldata    0.1.1      v workflowsets 0.1.0
v parsnip      0.1.7      v yardstick    0.0.8
v recipes      0.1.17
```

```
-- Conflicts ------------------------------------------------------------------------ tidymodels_conflicts()
x purrr::discard()  masks scales::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
* Learn how to get started at https://www.tidymodels.org/start/
```

```
library(vip)
```

```
Attaching package: 'vip'

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

    vi
```

```
library(GGally)
```

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

```
tidymodels_prefer()
doParallel::registerDoParallel()
```

Data

Our dataset has the name concrete and we made a local copy that we now load. The outputs below provides important information on the dataset such as its size and also some statistics on the variables distributions.

Summary

```
concrete <- read_csv("data/concrete.csv")
```

```
Rows: 1030 Columns: 9
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (9): cement, blast_furnace_slag, fly_ash, water, superplasticizer, coarse_aggregate, fine_aggregate

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
summary(concrete)
```

```
     cement      blast_furnace_slag    fly_ash           water       superplasticizer coarse_aggregate
 Min.   :102.0   Min.   :  0.0      Min.   :  0.00   Min.   :121.8   Min.   : 0.000   Min.   : 801.0
 1st Qu.:192.4   1st Qu.:  0.0      1st Qu.:  0.00   1st Qu.:164.9   1st Qu.: 0.000   1st Qu.: 932.0
 Median :272.9   Median : 22.0      Median :  0.00   Median :185.0   Median : 6.400   Median : 968.0
 Mean   :281.2   Mean   : 73.9      Mean   : 54.19   Mean   :181.6   Mean   : 6.205   Mean   : 972.9
 3rd Qu.:350.0   3rd Qu.:142.9      3rd Qu.:118.30   3rd Qu.:192.0   3rd Qu.:10.200   3rd Qu.:1029.4
 Max.   :540.0   Max.   :359.4      Max.   :200.10   Max.   :247.0   Max.   :32.200   Max.   :1145.0
 fine_aggregate       age         compressive_strength
 Min.   :594.0   Min.   :  1.00   Min.   : 2.33
 1st Qu.:731.0   1st Qu.:  7.00   1st Qu.:23.71
 Median :779.5   Median : 28.00   Median :34.45
 Mean   :773.6   Mean   : 45.66   Mean   :35.82
 3rd Qu.:824.0   3rd Qu.: 56.00   3rd Qu.:46.13
 Max.   :992.6   Max.   :365.00   Max.   :82.60
```

A first step in our data treatment is to calculate the proportion of each recipe component from the total weight:

```
concrete_mixture <- concrete |>
  mutate(total_weight = water + cement + blast_furnace_slag + fly_ash + superplasticizer + coarse_aggreg
  mutate(water_prop = water / total_weight,
```

```
          cement_prop = cement / total_weight,
          blast_prop = blast_furnace_slag / total_weight,
          fly_prop = fly_ash / total_weight,
          superplast_prop = superplasticizer / total_weight,
          coarse_agg_prop = coarse_aggregate / total_weight,
          fine_agg_prop = fine_aggregate / total_weight)
```

A check can be done to confirm that ingredients always add up to one:

```
concrete_mixture <- concrete_mixture |>
  mutate(prop_control = water_prop + cement_prop + blast_prop + fly_prop + superplast_prop + coarse_agg

prop_control <- unique(concrete_mixture$prop_control) |> round(8)
prop_control
```

```
[1] 1 1 1 1
```

From the book also we learn that the input factor Age has a strong influence on the outcome and we assume that it has a strong impact on cost. This is also the age subgroup with the highest number of datapoints. As in the text for illustrative purposes, we're now fixing this input factor in our study at 28 days and filtering the data accordingly:

```
concrete_mixture_28 <- concrete_mixture %>%
  filter(age == 28)
```

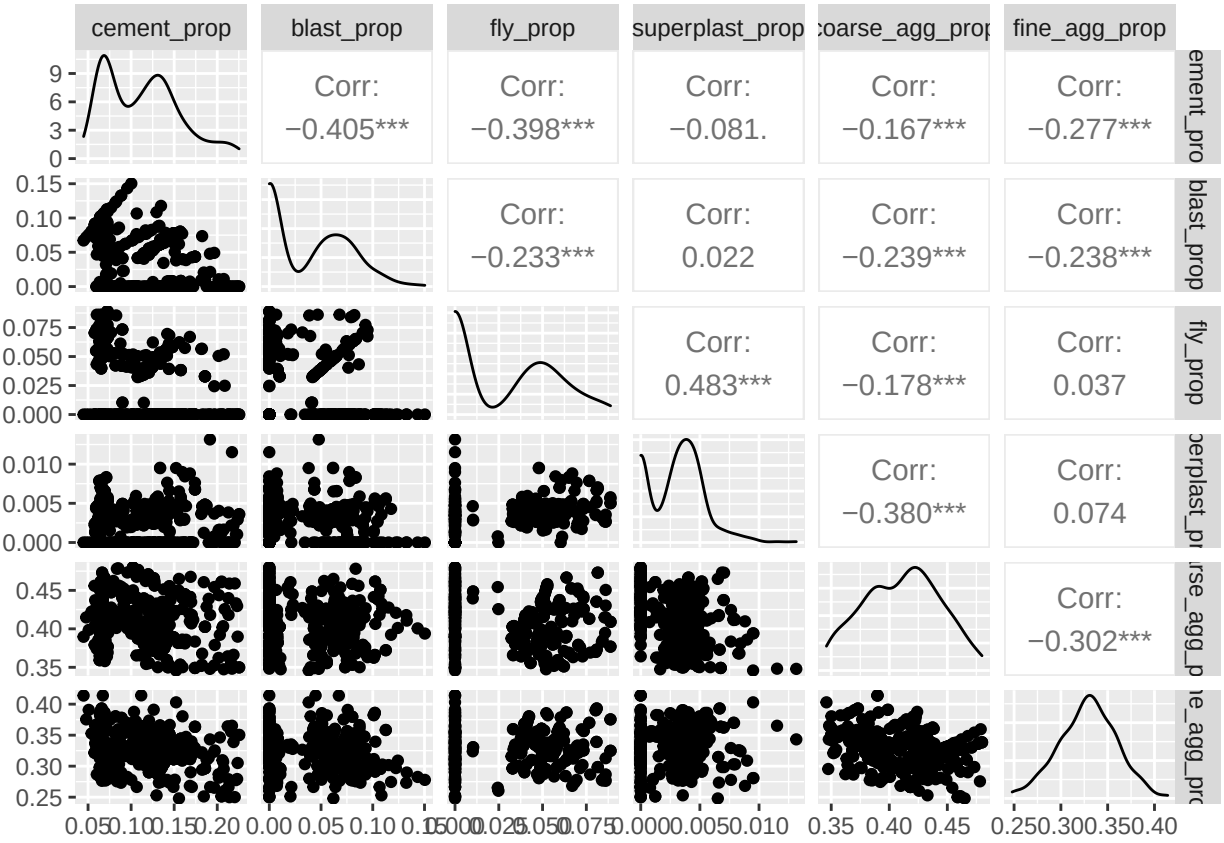
Correlation plot

We can now have a look at the input variables, their distributions and their relationship with each other:

```
ggpairs(concrete_mixture_28[,12:17])
```

```
plot: [1,1] [=>-------------------------------------------------------------------------------] 3% est: 0s
plot: [1,2] [===>-----------------------------------------------------------------------------] 6% est: 1s
plot: [1,3] [=====>---------------------------------------------------------------------------] 8% est: 1s
plot: [1,4] [========>------------------------------------------------------------------------] 11% est: 1s
plot: [1,5] [==========>----------------------------------------------------------------------] 14% est: 1s
plot: [1,6] [============>--------------------------------------------------------------------] 17% est: 1s
plot: [2,1] [==============>------------------------------------------------------------------] 19% est: 1s
plot: [2,2] [================>----------------------------------------------------------------] 22% est: 1s
plot: [2,3] [==================>--------------------------------------------------------------] 25% est: 1s
plot: [2,4] [====================>------------------------------------------------------------] 28% est: 1s
plot: [2,5] [======================>----------------------------------------------------------] 31% est: 1s
plot: [2,6] [========================>--------------------------------------------------------] 33% est: 1s
plot: [3,1] [==========================>------------------------------------------------------] 36% est: 1s
plot: [3,2] [============================>----------------------------------------------------] 39% est: 1s
plot: [3,3] [==============================>--------------------------------------------------] 42% est: 1s
plot: [3,4] [================================>------------------------------------------------] 44% est: 1s
plot: [3,5] [==================================>----------------------------------------------] 47% est: 1s
plot: [3,6] [====================================>--------------------------------------------] 50% est: 1s
plot: [4,1] [======================================>------------------------------------------] 53% est: 1s
plot: [4,2] [========================================>----------------------------------------] 56% est: 1s
plot: [4,3] [==========================================>--------------------------------------] 58% est: 1s
plot: [4,4] [============================================>------------------------------------] 61% est: 1s
plot: [4,5] [==============================================>----------------------------------] 64% est: 1s
plot: [4,6] [================================================>--------------------------------] 67% est: 1s
plot: [5,1] [==================================================>------------------------------] 69% est: 0s
```

```
plot: [5,2] [=============================================================>----------------------] 72% est: 0s
plot: [5,3] [==============================================================>---------------------] 75% est: 0s
plot: [5,4] [================================================================>-------------------] 78% est: 0s
plot: [5,5] [=================================================================>------------------] 81% est: 0s
plot: [5,6] [===================================================================>----------------] 83% est: 0s
plot: [6,1] [======================================================================>-------------] 86% est: 0s
plot: [6,2] [=======================================================================>------------] 89% est: 0s
plot: [6,3] [=========================================================================>----------] 92% est: 0s
plot: [6,4] [==========================================================================>---------] 94% est: 0s
plot: [6,5] [============================================================================>-------] 97% est: 0s
plot: [6,6] [==============================================================================]100% est: 0s
```



We can observe that the correlation is low to moderate in most cases. The distribution is useful to get some expert insight into the proportions and will help later in interpreting the results from the optimization.

Modeling

Sample

As explained in the previous articles on modeling with tidymodels, we here use functions from the sample package to split the dataset into train and test datasets.

```
set.seed(123)
concrete_split <- initial_split(concrete_mixture_28, prop = 0.8)
concrete_train <- training(concrete_split)
concrete_test <- testing(concrete_split)
```

Recipe

In our recipe we're adding all Cement recipe components but not the age as decided before:

```
tree_recipe <- recipe(
  formula = compressive_strength ~ cement_prop + blast_prop + fly_prop + superplast_prop + coarse_agg_p
```

The next steps follow also the tidymodels approach.

Model

```
model_rf <-
  rand_forest() |>
  set_engine('ranger', importance = "impurity") |>
  set_mode('regression')
```

Workflow

```
rf_workflow <-
  workflow() |>
  add_recipe(tree_recipe) |>
  add_model(model_rf)
```

Fit

```
fit_rf <-
  rf_workflow |>
  fit(data = concrete_train)

fit_rf_test <-
  rf_workflow |>
  fit(data = concrete_test)
```

Predict

```
predict_rf <-
  fit_rf |>
  predict(new_data = concrete_train) |>
  bind_cols(concrete_train)

predict_rf_test <-
  fit_rf |>
  predict(new_data = concrete_test) |>
  bind_cols(concrete_test)
```

Metrics

```
metrics_model_rf <- predict_rf |>
  metrics(truth = compressive_strength, estimate = .pred)
metrics_model_rf_test <- predict_rf_test |>
  metrics(truth = compressive_strength, estimate = .pred)
```

We can here check our model performance on the train dataset:

```
metrics_model_rf
```

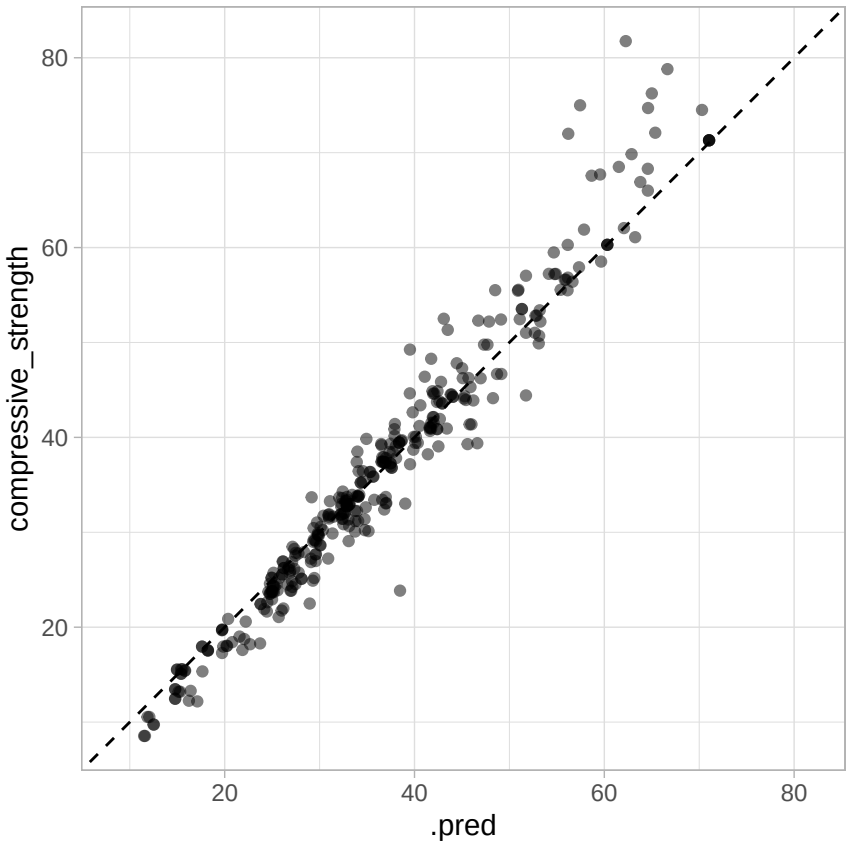
```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
```

```
1 rmse    standard       3.34
2 rsq     standard       0.957
3 mae     standard       2.12
```

The fit for the train dataset is extremely high with a value above 0.90 and the Mean Absolute Error is very low, slightly above 2. We can see in the plot below, real vs predict values for the train dataset: specially until values of 60, points are well on top of the plot diagonal.

Fit plot

```
predict_rf |>
    ggplot(aes(x = .pred, y = compressive_strength)) +
    geom_abline(lty = 2) +
    geom_point(alpha = 0.5) +
    coord_obs_pred(ratio = 1) + # Scale and size the x- and y-axis uniformly:
    theme_light()
```



For the test dataset the RSQ is still above 0.70. We can now we expect this from new real data.

```
metrics_model_rf_test
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       8.20
2 rsq     standard       0.740
3 mae     standard       5.99
```

Variable importance

Now that we have a model, we're listing the ranking of explanatory variables:

```
fit_rf |>
  extract_fit_parsnip() |>
  vip::vi(num_features = 20)
```

```
# A tibble: 6 x 2
  Variable        Importance
  <chr>                <dbl>
1 cement_prop         26236.
2 coarse_agg_prop     11423.
3 fine_agg_prop        8929.
4 blast_prop           8019.
5 fly_prop             7822.
6 superplast_prop      7328.
```

Optimizing

To find a recipe with maximal compressive strength we're going to prepare a table with possible combinations of the input factors exploring a space slightly beyond the original dataset.

Search space

We start by seeing the statistics of components proportions (as reminder the initial statistics presented were on absolute values).

```
summary(concrete_mixture_28[11:18])
```

```
   water_prop       cement_prop        blast_prop         fly_prop       superplast_prop
 Min.   :0.05139   Min.   :0.04482   Min.   :0.00000   Min.   :0.00000   Min.   :0.000000
 1st Qu.:0.07340   1st Qu.:0.07080   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000000
 Median :0.07894   Median :0.11456   Median :0.04049   Median :0.02479   Median :0.003302
 Mean   :0.07895   Mean   :0.11362   Mean   :0.03724   Mean   :0.02728   Mean   :0.003007
 3rd Qu.:0.08450   3rd Qu.:0.14010   3rd Qu.:0.06873   3rd Qu.:0.05121   3rd Qu.:0.004486
 Max.   :0.11222   Max.   :0.22541   Max.   :0.15034   Max.   :0.08884   Max.   :0.013149
 coarse_agg_prop  fine_agg_prop     prop_control
 Min.   :0.3459   Min.   :0.2480   Min.   :1
 1st Qu.:0.3875   1st Qu.:0.3092   1st Qu.:1
 Median :0.4132   Median :0.3297   Median :1
 Mean   :0.4111   Mean   :0.3288   Mean   :1
 3rd Qu.:0.4354   3rd Qu.:0.3503   3rd Qu.:1
 Max.   :0.4798   Max.   :0.4141   Max.   :1
```

These statistics give us insight on which ranges to vary each component:

```
concrete_new <- expand.grid(
  cement_prop = seq(0.04, 0.4, 0.02),
  superplast_prop = seq(0, 0.02, 0.01),
  fine_agg_prop = seq(0.1, 0.5, 0.1),
  coarse_agg_prop = seq(0.2, 1, 0.1),
  blast_prop = seq(0, 0.3, 0.05),
  fly_prop = seq(0, 1, 0.1))

concrete_new$age <- 28
```

The expand.grid function is naturally going to create combinations for which the sum of the proportions is greater than 1. From the summary statistics we could see that the water proportion can go up to 0.11 . As the water is added when all other ingredients are fixed we can use this fact to remove all recipes that don't leave room to add water.

We add a variable to calculate the sum of the proportions:

```
concrete_new <- concrete_new |>
  mutate(total_prop = cement_prop + superplast_prop + fine_agg_prop + coarse_agg_prop + blast_prop + fl
```

and we retain only combinations that are lower that 1 - 0.15 in order to leave 15% for water (a bit more than the current max of water):

```
concrete_optim <- concrete_new %>%
  filter(total_prop < 1 - 0.15)
```

Predict new

Now we feed the Search space dataset to our predictive function and calculate Compressive strengths for these potential recipes.

```
predict_rf_new <-
  fit_rf |>
  predict(new_data = concrete_optim) |>
  bind_cols(concrete_optim)
```

Results

Find optimal

Our optimal recipe with this approach is the one with the highest prediction which we obtain by sorting the prediction output table:

```
predict_rf_new |>
  arrange(desc(.pred)) |>
  select(.pred, cement_prop, coarse_agg_prop, fine_agg_prop, blast_prop, fly_prop, superplast_prop) |>
  head(5)
```

```
# A tibble: 5 x 7
  .pred cement_prop coarse_agg_prop fine_agg_prop blast_prop fly_prop superplast_prop
  <dbl>       <dbl>           <dbl>         <dbl>      <dbl>    <dbl>           <dbl>
1  63.8        0.18             0.2           0.3       0.15        0            0.01
2  63.7        0.16             0.2           0.3       0.15        0            0.02
3  63.7        0.16             0.2           0.3       0.15        0            0.01
4  63.7        0.18             0.2           0.3       0.1         0            0.02
5  63.6        0.18             0.2           0.3       0.1         0            0.01
```

Clearly the first values correspond to to an increase in the cement proportion which is the factor that has the higher variable importance. Interestingly, comparing with the original dataset we see we had recipes with higher compressive strength:

```
concrete_mixture_28 |>
  arrange(desc(compressive_strength)) |>
  select(compressive_strength, cement_prop, coarse_agg_prop, fine_agg_prop, blast_prop, fly_prop, superp
  head(5)
```

```
# A tibble: 5 x 7
  compressive_strength cement_prop coarse_agg_prop fine_agg_prop blast_prop fly_prop superplast_prop
                 <dbl>       <dbl>           <dbl>         <dbl>      <dbl>    <dbl>           <dbl>
```

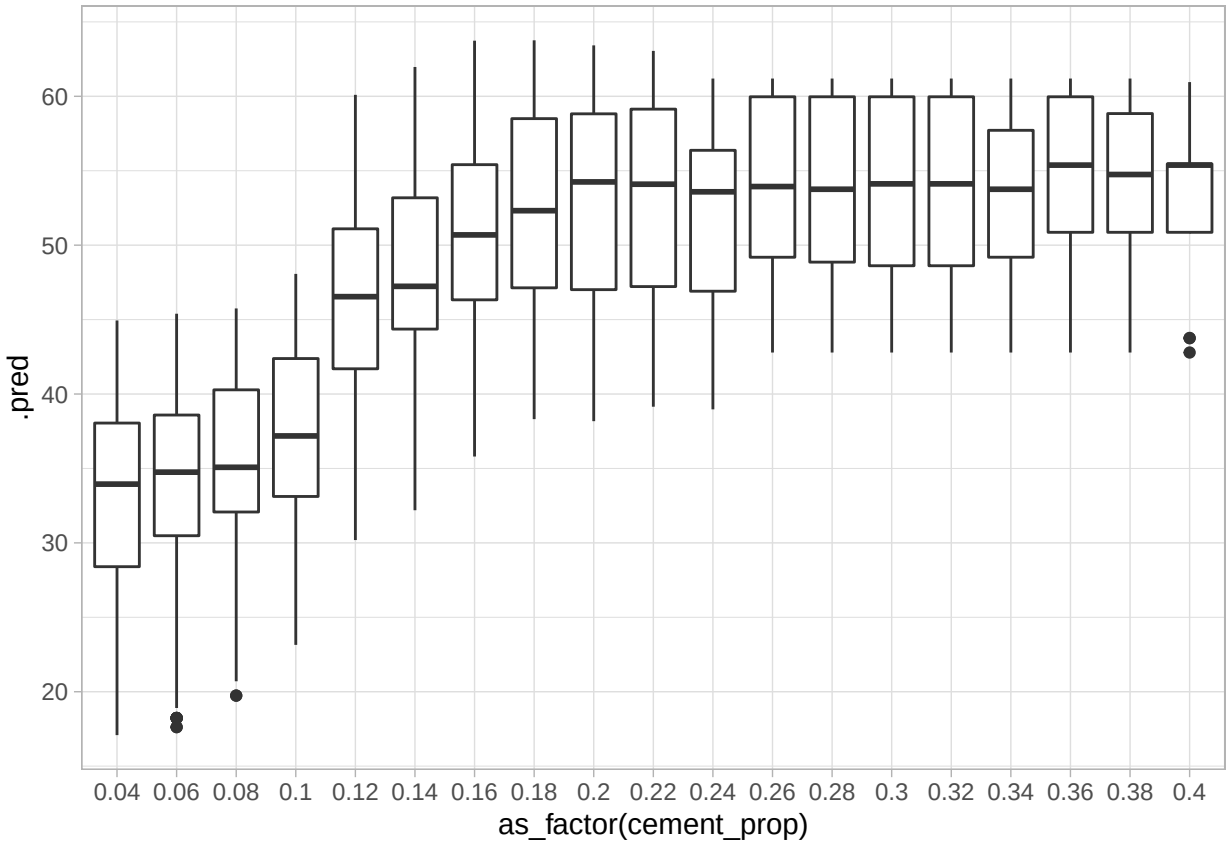
1	81.8	0.127	0.456	0.301	0.0553	0	0.00238
2	80.0	0.223	0.430	0.279	0	0	0.00103
3	78.8	0.188	0.429	0.310	0	0	0.00470
4	76.2	0.117	0.354	0.327	0.0768	0.0512	0.00444
5	75.0	0.212	0.364	0.364	0	0	0

Looking back at the Fit Plot we see that above 60 the model is under predicting the values. Now this opens for the important discussion between using a model and interpreting it instead of just doing random trials and considering that the input factors can be generalized.

In the next plots we look at how the main influencing factor relates with the Compressive Strength both in reality and in the model.

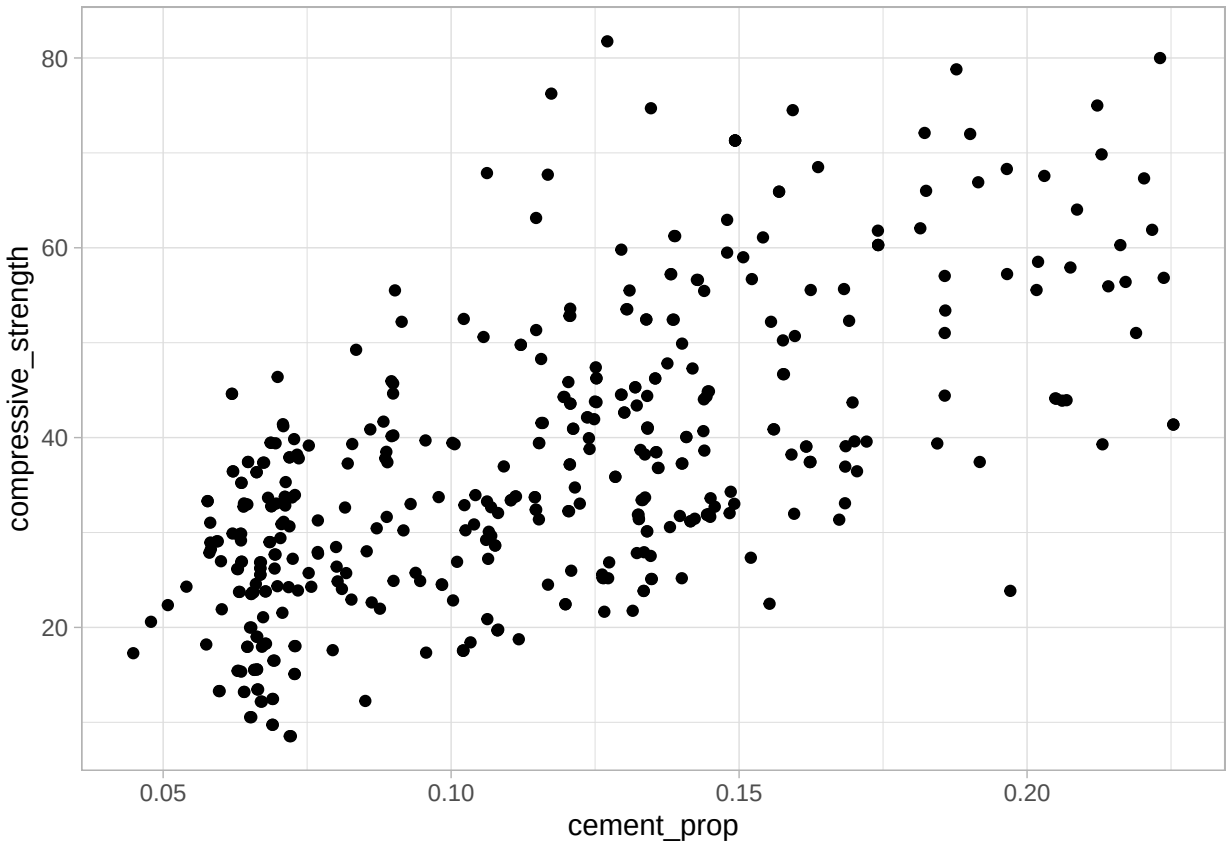
Plot Strenght ~ Cement

```
predict_rf_new |>
  ggplot(aes(x = as_factor(cement_prop), y = .pred)) +
  geom_boxplot() +
  theme_light()
```



The previous plot shows that according to the model it is enough to use 0.18 for cement to maximize strength. We also see visually that the variability of this value is approximately $\pm$ 5 which is close to the metrics calculated on the test dataset.

```
concrete_mixture_28 |>
  ggplot(aes(x = cement_prop, y = compressive_strength)) +
  geom_point()+
  theme_light()
```



The previous plot shows that on the original dataset increasing cement proportion leads always to an increase in strength but the variability is much higher (visually around $\pm$ 10).

A similar analysis for the remaining input factors would certainly provide other valuable insights in the selection of the optimal recipe.

Conclusions

An optimal recipe that maximizes compression strength has been identified. This recipe corresponds to a maximization of the cement which is the input variable with the highest importance. Interestingly some recipes from the original dataset have an even higher compressive strength but the variability of the results in the compressive strength is much higher thus less reliable.

In these cases the next is step is naturally to confirm the predictions with real experiments to confirm the output and variability, finetune the model and eventually get to an improved recipe.

References

Max Kuhn, Kjell Johnson. 2021. *Applied Predictive Modeling.* 2nd ed. Springer. appliedpredictivemodeling.com.