Fertilizer advice for farmers growning maize in Nigeria

Ivana ALEKSOVSKA

2024-06-09

Carob

Carob creates reproducible workflows that standardize primary agricultural research data from experiments and surveys. Standardization includes the use of a common file format, variable names, units and accepted values according to the terminag standard. Standardized data sets are aggregated into larger collections that can be used in further research. We do this by writing an *R* script for each individual dataset. See the website for more information.

Carob is an open access *Extract, Transform, and Load* (ETL) framework supported by CGIAR to support predictive analytics (machine learning, artifical intelligence) and other types of data analysis.

Get the data

Compiled versions of the dataset can be downloaded from carob-data.org and some will eventually be made available on the carob dataverse.

Purpose

The purpose of this study is to develop tailored and climate smart fertilizer advice for maize growners in Nigeria based on the available data, and provide a decision support tool to advice farmers about the usage of fertilizer. For that, consider the yield response to N,P and K in the data and the spatial variation in soil and weather in the region to predict site specific N, P and K rates.

Below we explain how we go about meeting this challenge, and explain each step of the way.

```
##########################################
# MAIN SCRIPT TO DESIGN ANALYTICAL WORKLOW
# Ivana Aleksovska
##########################################

# This part contains the packages and the libraries that will be used in the following

# Packages for machine learning and data analysis
# install.packages(caret)
# install.packages(factoextra)
# install.packages(Metrics)

# Packages for Rmarkdown generator
# install.packages('knitr')
# install.packages("/Users/moia/Downloads/knitr_1.47.tar.gz", repos = NULL,
↪  type="source")
```

```r
library(tidyverse)
library(caret)
library(party)
library(factoextra)
library(corrplot)
library(gridExtra)
library(grid)
library(ggplot2)
library(lattice)
library(Metrics)
# ------------------------------------------------------------------------------------------
# 1. DEFINE THE QUESTION
# Develop tailored and climate smart fertilizer advice for maize growners in Nigeria
# Consider the yield response to N,P and K in the data and the spatial variation in
# soil and weather in the region to predict site specific N, P and K rates.

# ------------------------------------------------------------------------------------------
# 2. DOWNLOAD AND PREPARE THE DATA FOR ML

# I) First get the data
# Follow the steps explained here to get the data
# https://carob-data.org/compile.html

# 1) online
#install.packages("remotes")
#remotes::install_github("reagro/carobiner", force = TRUE)
#remotes::install_github("reagro/carobiner")
#ff <- carobiner::make_carob("/Users/moia/Desktop/github/carob")
# carobiner::update_terms()
# 2) or zip format
# install.packages("/Users/moia/Downloads/geodata_0.5-9.tar.gz", repos = NULL,
↪  type="source")
# install.packages("/Users/moia/Desktop/github/carob/carobiner", repos = NULL,
↪  type="source")

# LOAD THE DATA IN THE RSTUDIO WORKING ENVIRONMENT
carob_fertilizer_terms <-
↪  readr::read_csv("data/compiled/carob_fertilizer_terms.csv",show_col_types = FALSE)
carob_fertilizer_metadata <-
↪  readr::read_csv("data/compiled/carob_fertilizer_metadata.csv",show_col_types = FALSE)
carob_fertilizer <- readr::read_csv("data/compiled/carob_fertilizer.csv",show_col_types =
↪  FALSE)

# SELECT THE DATA OF MAIZE THAT CORRESPONDS TO THE REGION OF NIGERIA
colnames(carob_fertilizer) # see all columns available in the data frame
```

```
##   [1] "dataset_id"          "trial_id"            "record_id"
##   [4] "country"             "adm1"                "adm2"
##   [7] "adm3"                "adm4"                "adm5"
##  [10] "location"            "site"                "reference"
##  [13] "longitude"           "latitude"            "elevation"
##  [16] "date"                "planting_date"       "emergence_date"
```

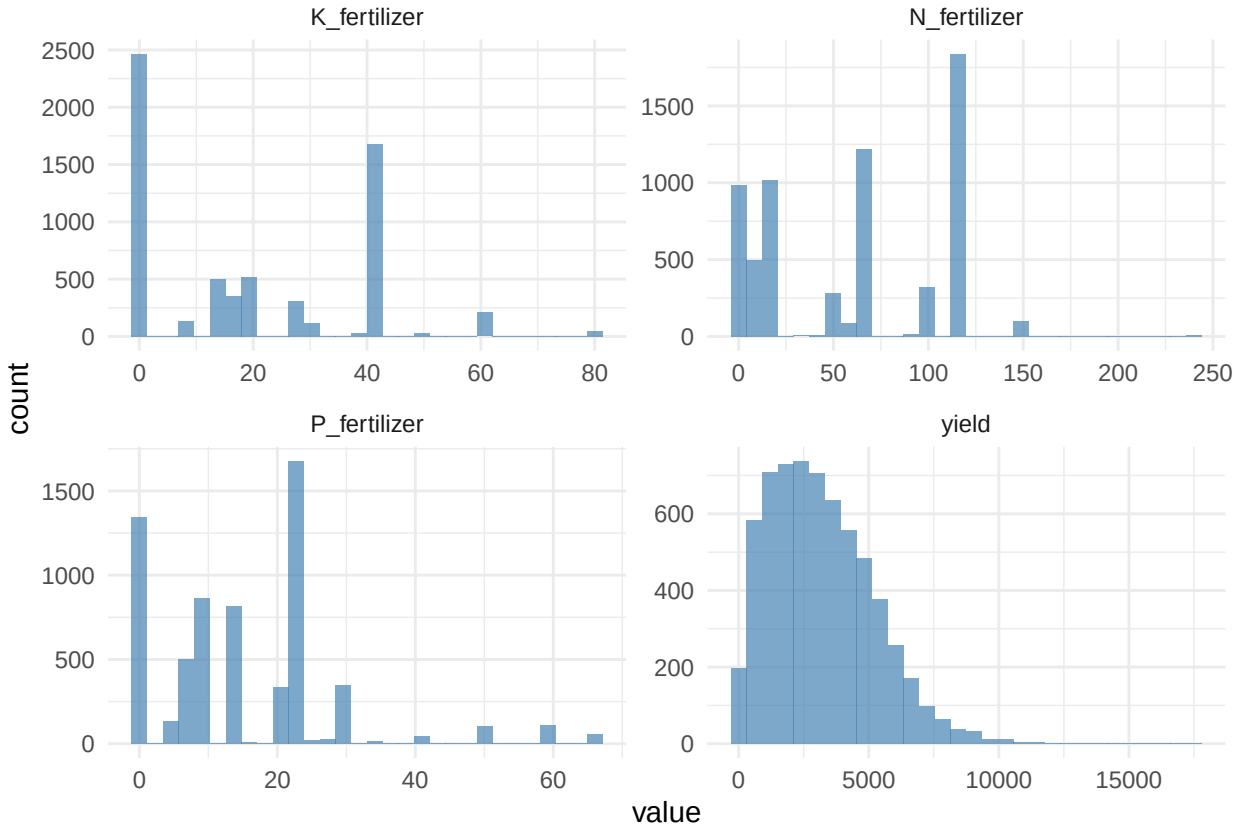
```
##  [19] "transplanting_date"   "flowering_date"       "maturity_date"
##  [22] "harvest_date"         "season"               "on_farm"
##  [25] "is_survey"            "treatment"            "rep"
##  [28] "crop"                 "variety"              "variety_code"
##  [31] "variety_type"         "intercrops"           "previous_crop"
##  [34] "crop_rotation"        "dmy_roots"            "dmy_stems"
##  [37] "dmy_storage"          "dmy_total"            "dmy_residue"
##  [40] "yield_part"           "yield"                "residue_yield"
##  [43] "grain_weight"         "fertilizer_type"      "N_fertilizer"
##  [46] "N_splits"             "P_fertilizer"         "K_fertilizer"
##  [49] "Zn_fertilizer"        "S_fertilizer"         "B_fertilizer"
##  [52] "Ca_fertilizer"        "Mg_fertilizer"        "Fe_fertilizer"
##  [55] "lime"                 "gypsum"               "OM_used"
##  [58] "OM_type"              "OM_amount"            "OM_N"
##  [61] "OM_P"                 "OM_K"                 "inoculated"
##  [64] "inoculant"            "irrigated"            "irrigation_source"
##  [67] "irrigation_number"    "irrigation_amount"    "irrigation_dates"
##  [70] "land_prep_method"     "planting_method"      "transplanting_method"
##  [73] "uncertainty"          "uncertainty_type"     "row_spacing"
##  [76] "plot_area"            "plant_spacing"        "plant_density"
##  [79] "rain"                 "emergence_days"       "heading_days"
##  [82] "flowering_days"       "anthesis_days"        "maturity_days"
##  [85] "harvest_days"         "plant_height"         "nodule_weight"
##  [88] "leaf_N"               "leaf_K"               "leaf_P"
##  [91] "leaf_Mg"              "leaf_Ca"              "leaf_Zn"
##  [94] "grain_N"              "grain_K"              "grain_P"
##  [97] "grain_Mg"             "grain_Ca"             "grain_Fe"
## [100] "grain_Zn"             "grain_Cu"             "grain_Mn"
## [103] "grain_Al"             "grain_protein"        "residue_N"
## [106] "residue_P"            "season_constraint"    "herbicide_times"
## [109] "weeding_times"        "weeding_dates"        "weed_biomass"
## [112] "insecticide_times"    "fungicide_times"      "diseases"
## [115] "crop_cut"             "soil_type"            "soil_pH"
## [118] "soil_pH_CaCl2"        "soil_SOC"             "soil_SOM"
## [121] "soil_sand"            "soil_clay"            "soil_silt"
## [124] "soil_texture"         "soil_N"               "soil_K"
## [127] "soil_P_total"         "soil_P_available"     "soil_P_Mehlich"
## [130] "soil_Ca"              "soil_Mg"              "soil_Cu"
## [133] "soil_Mn"              "soil_Fe"              "soil_Zn"
## [136] "soil_Na"              "soil_EC"              "soil_ECEC"
## [139] "soil_color"           "soil_depth"           "soil_constraint"
## [142] "soil_ex_acidity"
```

```
carob_fertilizer<- carob_fertilizer[ which(carob_fertilizer$country=="Nigeria" &
↪  carob_fertilizer$crop =="maize") , ]

# ------------------------------------------------------------------------------
# 3. DATA ANALYSIS (descriptive statistic and visualisation)
# a) Perform a comprehensive exploratory data analysis to summarize the main
↪  characteristics of the data)
# b) Use appropriate visualizations to illustrate your findings.

# In this context you should consider the yield response to N, P and K in the data and
↪  the spatial variation in soil and weather in the region to predict site specific N,
↪  P, and K rates.
```

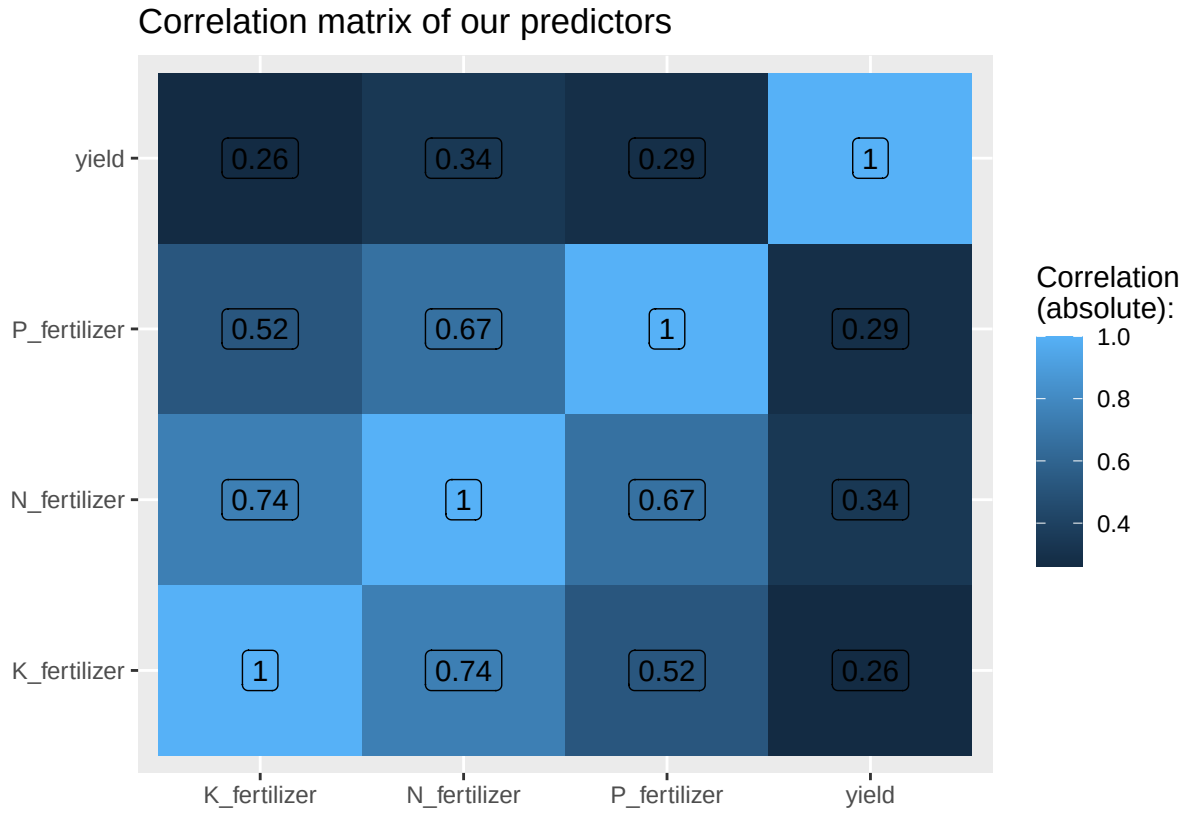
```r
# In short, fertilizers are labeled N, P or K to indicate their nutrient content in terms
↪  of nitrogen (N), phosphorus (P), and potassium (K). All three are important for plant
↪  growth.
# N_fertilizer: numeric (kg/ha) N applied in inorganic fertilizer
# P_fertilizer: numeric (kg/ha) P applied in inorganic fertilizer (as P, not P2O5)
# K_fertilizer: numeric (kg/ha) K applied in inorganic fertilizer (as K, not K2O)
# yield (kg/ha)

# ------
# Identify the important predictors X that influence the yield
predictors=c('N_fertilizer','P_fertilizer','K_fertilizer','yield')

# from the carob_fertilizer select only those columns that will be used into ML algo
carob_fertilizer_ML=carob_fertilizer[predictors]

# Before all clean the NA (to avoid missing values in the statistics) and the remove the
↪  duplicate values in the data frame (this comes usually from errors in savings, and it
↪  introduce bias since double)
carob_fertilizer_ML <-na.omit(carob_fertilizer_ML)
carob_fertilizer_ML <- carob_fertilizer_ML[!duplicated(carob_fertilizer_ML),]
carob_fertilizer_ML=as.data.frame(carob_fertilizer_ML) # convert to data frame

# ------

# Visual inspection / descriptive statistics
# Let's see the main statistics in the data. Here we are interested into mean, min, max
↪  values, and also for some quantiles.
summary(carob_fertilizer_ML)
```

```
##   N_fertilizer     P_fertilizer    K_fertilizer       yield
##  Min.   :  0.00   Min.   : 0.00   Min.   : 0.00   Min.   :    0
##  1st Qu.: 14.00   1st Qu.: 6.55   1st Qu.: 0.00   1st Qu.: 1607
##  Median : 66.00   Median :13.54   Median :17.43   Median : 2927
##  Mean   : 61.31   Mean   :15.03   Mean   :19.41   Mean   : 3174
##  3rd Qu.:120.00   3rd Qu.:21.85   3rd Qu.:41.50   3rd Qu.: 4451
##  Max.   :240.00   Max.   :66.00   Max.   :80.00   Max.   :17500
```

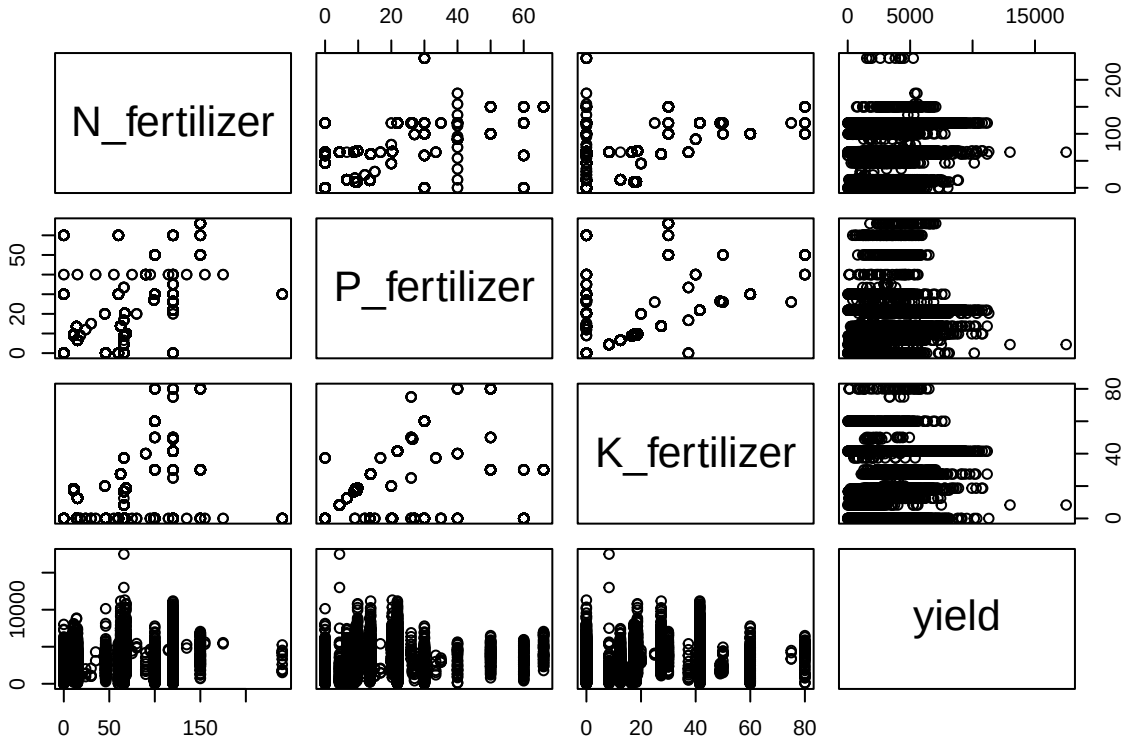
```r
# Histograms of all variables in our data set, the target, labeled "yield" and the
↪  predictors, K, N and P fertilizers. Here you can see the univariate distributions,
↪  one variable at a time. We can visually check the type of distribution, whether it's
↪  normal or not. In addition, we can check whether they need some form of normalization
↪  or discretization. All these variables are continuous variables.

# See the histograms for every variable (column)
carob_fertilizer_ML %>% gather() %>%
  ggplot(aes(x=value)) +
  geom_histogram(fill="steelblue", alpha=.7) +
  theme_minimal() +
  facet_wrap(~key, scales="free")
```




```
# Let's move on to bivariate statistics. We are plotting a correlation matrix, in order
↪  to :
# a) check if we have predictors that are highly correlated (which is problematic for
↪  some algorithms),
# b) get a first feeling about which predictors are correlated with the target.
# Strong correlation ~1.

corr_matrix <- cor(carob_fertilizer_ML %>% keep(is.numeric))
corr_matrix %>% as.data.frame %>% mutate(var2=rownames(.)) %>%
    pivot_longer(!var2, values_to = "value") %>%
    ggplot(aes(x=name,y=var2,fill=abs(value),label=round(value,2))) +
    geom_tile() + geom_label() + xlab("") + ylab("") +
    ggtitle("Correlation matrix of our predictors") +
    labs(fill="Correlation\n(absolute):")
```




```
#Let's also visualize the pairwise correlation graph using the pairs function, to see the
↪  correlation between each pair of available data, i.e. between the target and each
↪  input, but also to study each pair of inputs.
pairs(carob_fertilizer_ML)
```




```
# Principal Component Analysis (PCA) is a statistical method designed to reduce
↪  dimensionality. This is useful if we want to make a model (decision support tool)
↪  based on multiple predictors.PCA is used to extract meaningful information from a
↪  table of multivariate data, and to express this information as a set of a few new
↪  variables called principal components. The aim of PCA is to identify directions that
↪  can be visualized graphically with minimal loss of information. The first principal
↪  component can be equivalently defined as a direction that maximizes the variance of
↪  the projected data. Principal components are often analyzed via the covariance matrix
↪  decomposition of the data or the singular value decomposition (SVD) of the data
↪  matrix.
res.pca <- prcomp(carob_fertilizer_ML[,-4], scale = TRUE) # exclude the yield column, PCA
↪  is unsupervised machine learning
print(res.pca)
```

```
## Standard deviations (1, .., p=3):
## [1] 1.5139550 0.6959720 0.4728247
##
## Rotation (n x k) = (3 x 3):
##                      PC1        PC2        PC3
## N_fertilizer -0.6112712  0.1164667  0.7828046
## P_fertilizer -0.5470334 -0.7769695 -0.3115653
## K_fertilizer -0.5719283  0.6186711 -0.5386503
```

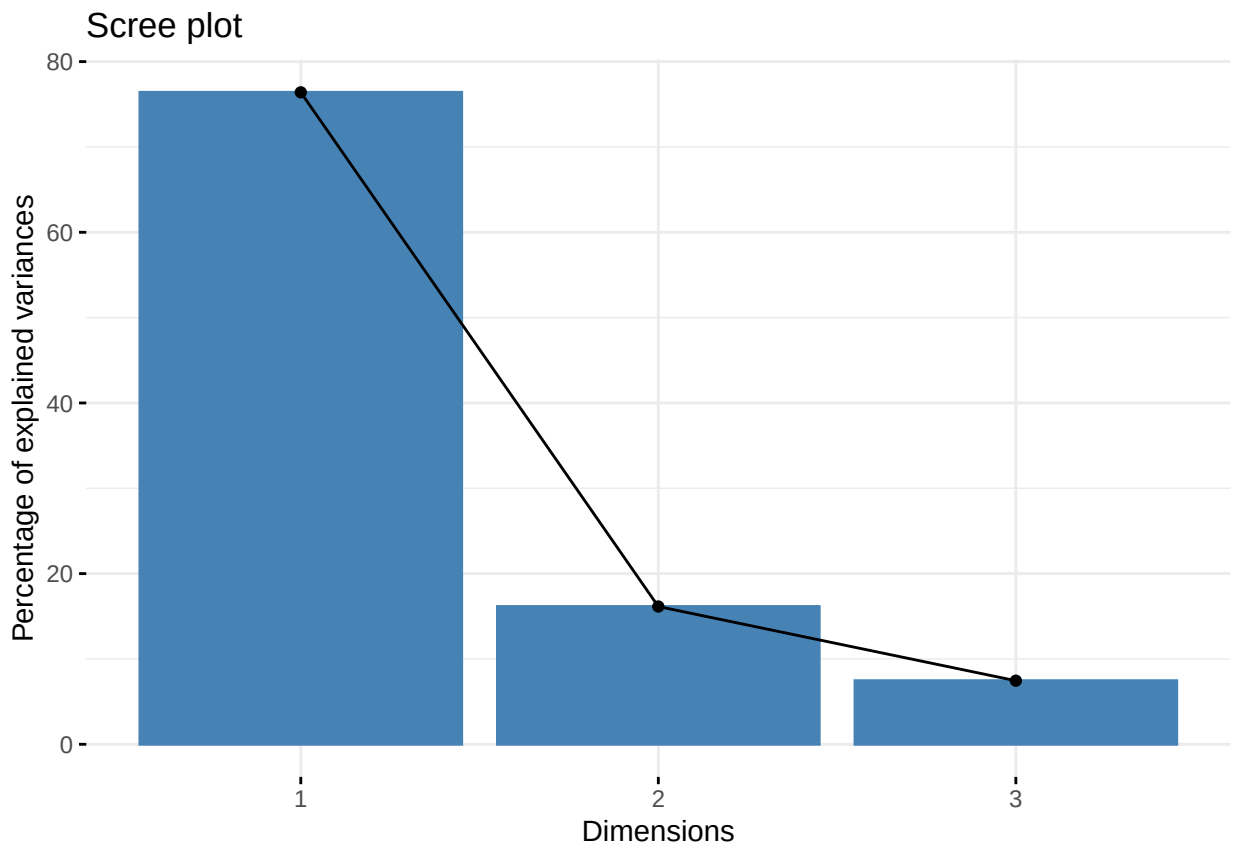
```
summary(res.pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3
## Standard deviation     1.514 0.6960 0.47282
## Proportion of Variance 0.764 0.1615 0.07452
## Cumulative Proportion  0.764 0.9255 1.00000
```

```
eig.val<-get_eigenvalue(res.pca)
eig.val
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.2920598        76.401995                    76.40199
## Dim.2  0.4843770        16.145900                    92.54789
## Dim.3  0.2235632         7.452105                   100.00000
```

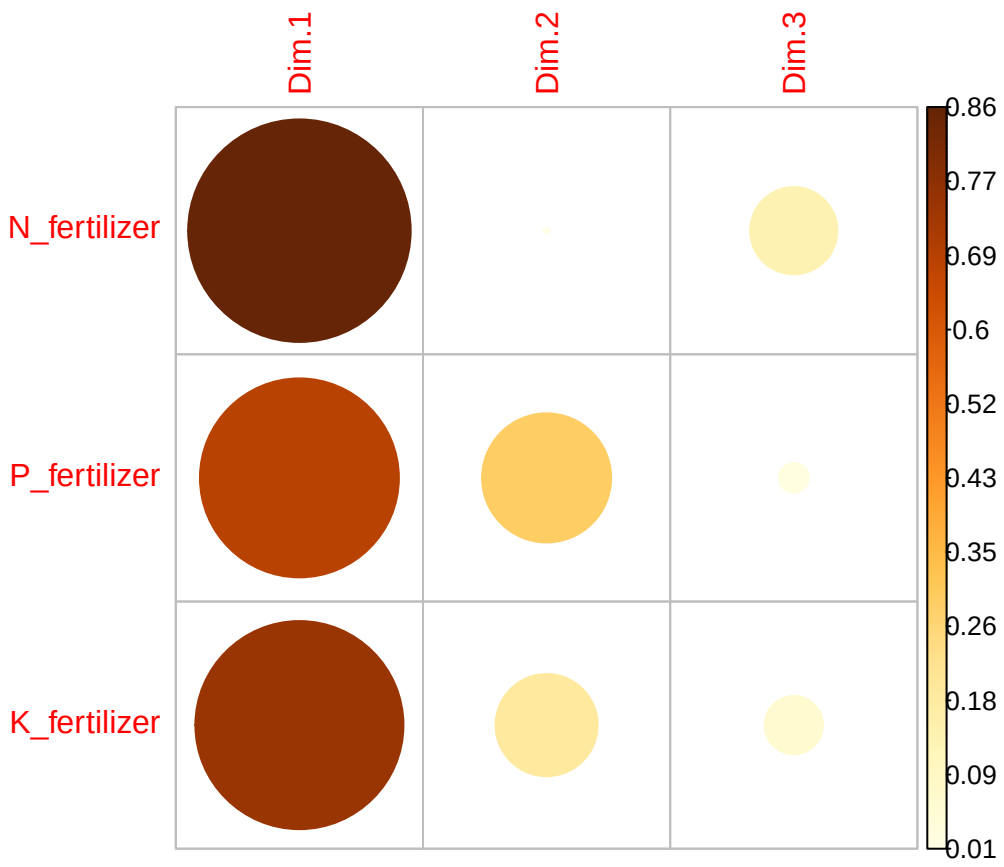
```
# On the basis of importance of components, is it visible that first two PC has the
↪  highest vales for proportion of variance. This statement is also proved by
↪  eigenvalues measure. They are large for the first PC and small for the subsequent
↪  PCs, which means that the first PC corresponds to the directions with the maximum
↪  amount of variation in the data set. As far as scatter plot is concerned, first
↪  eigenvalue explain more than 75% of the variation, second ~20%. Therefore, more than
↪  95% of the variation is explained by the first two eigenvalues together, which is a
↪  proper indicator for further analysis.
fviz_eig(res.pca, col.var="blue")
```



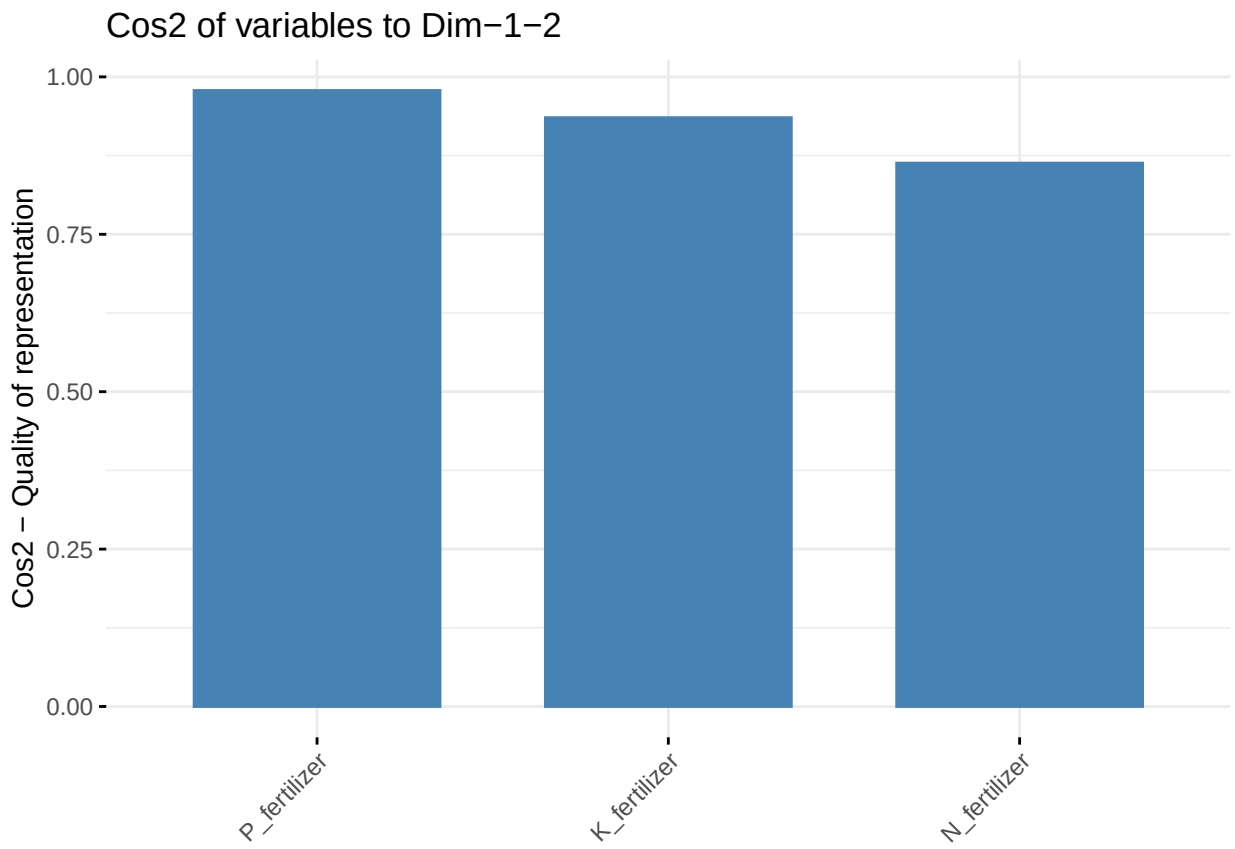

```
# PCA results can be assessed with regard to variables. Firstly, I will conduct
↪  extraction of results for variables. For that purpose get_pca_var() is used to
↪  provide a list of matrices containing all the results for the active variables
↪  (coordinates, correlation between variables and axes, squared cosine, and
↪  contributions).

var <- get_pca_var(res.pca)
var
```

```
## Principal Component Analysis Results for variables
##  ===================================================
##   Name       Description
## 1 "$coord"   "Coordinates for the variables"
## 2 "$cor"     "Correlations between variables and dimensions"
## 3 "$cos2"    "Cos2 for the variables"
## 4 "$contrib" "contributions of the variables"
```

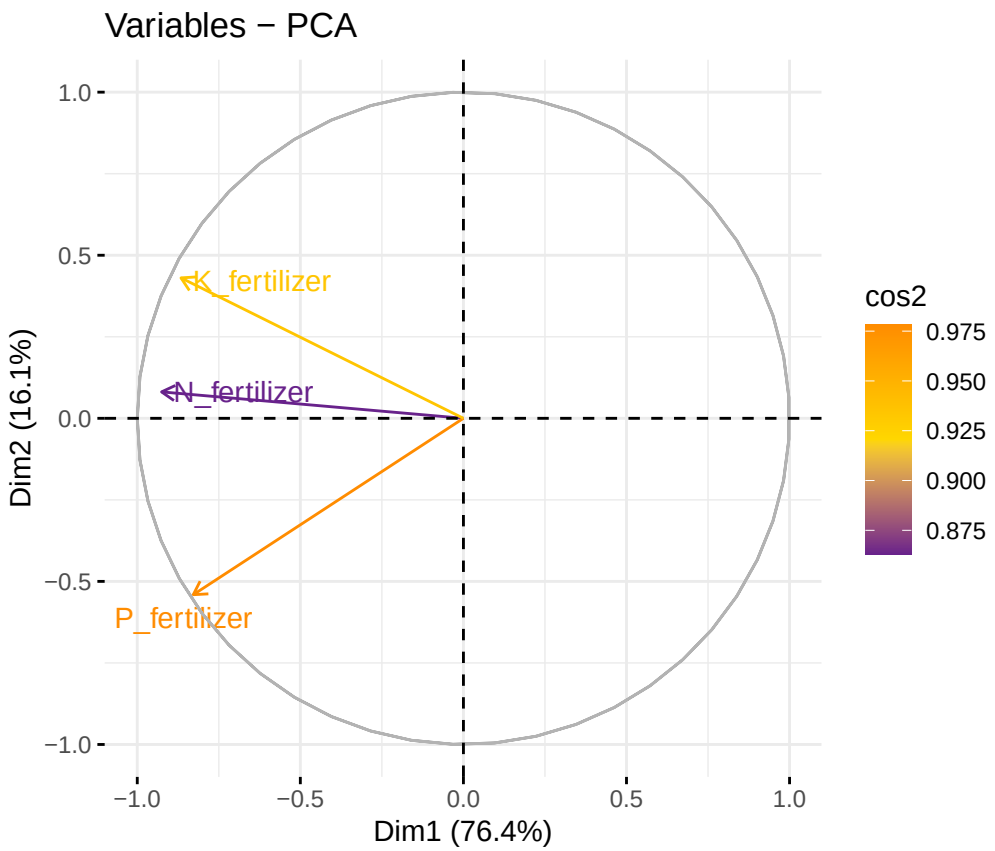
```
corrplot(var$cos2, is.corr=FALSE)
```



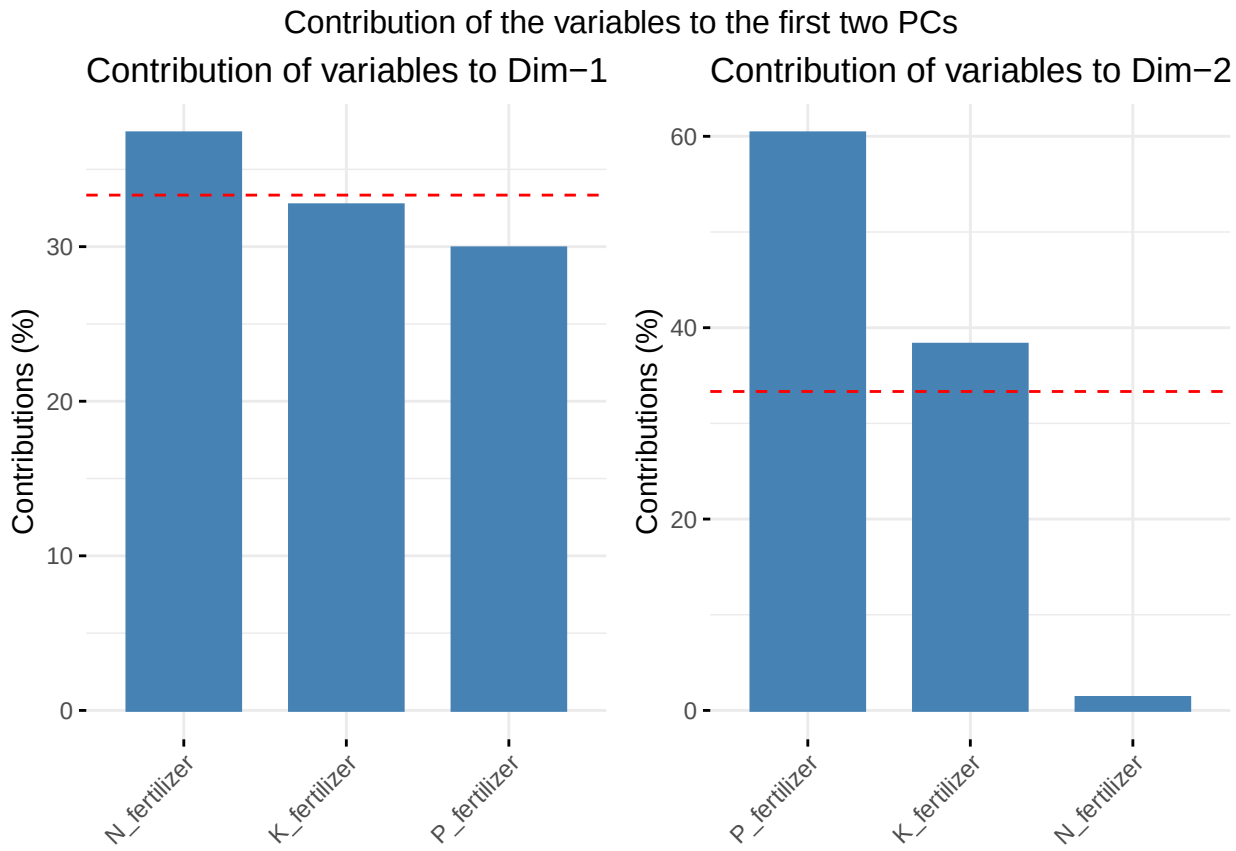

```
fviz_cos2(res.pca, choice = "var", axes = 1:2)
```




```
# Additionally, the quality of representation of variables can be draw on the factor map,
↪  where cos2 values differ by gradient colors. Variables with low cos2 values will be
↪  colored "darkorchid4", medium cos2 values - "gold", high co2 values - "darkorange".
↪  Positively correlated variables are grouped together, whereas negatively correlated
↪  variables are positioned on opposite sides of the plot origin. The distance between
↪  variables and the origin measures the quality of the variables on the factor map.
↪  Variables that are away from the origin are well represented on the factor map.

fviz_pca_var(res.pca,
             col.var = "cos2", # Color by the quality of representation
             gradient.cols = c("darkorchid4", "gold", "darkorange"),
             repel = TRUE
)
```

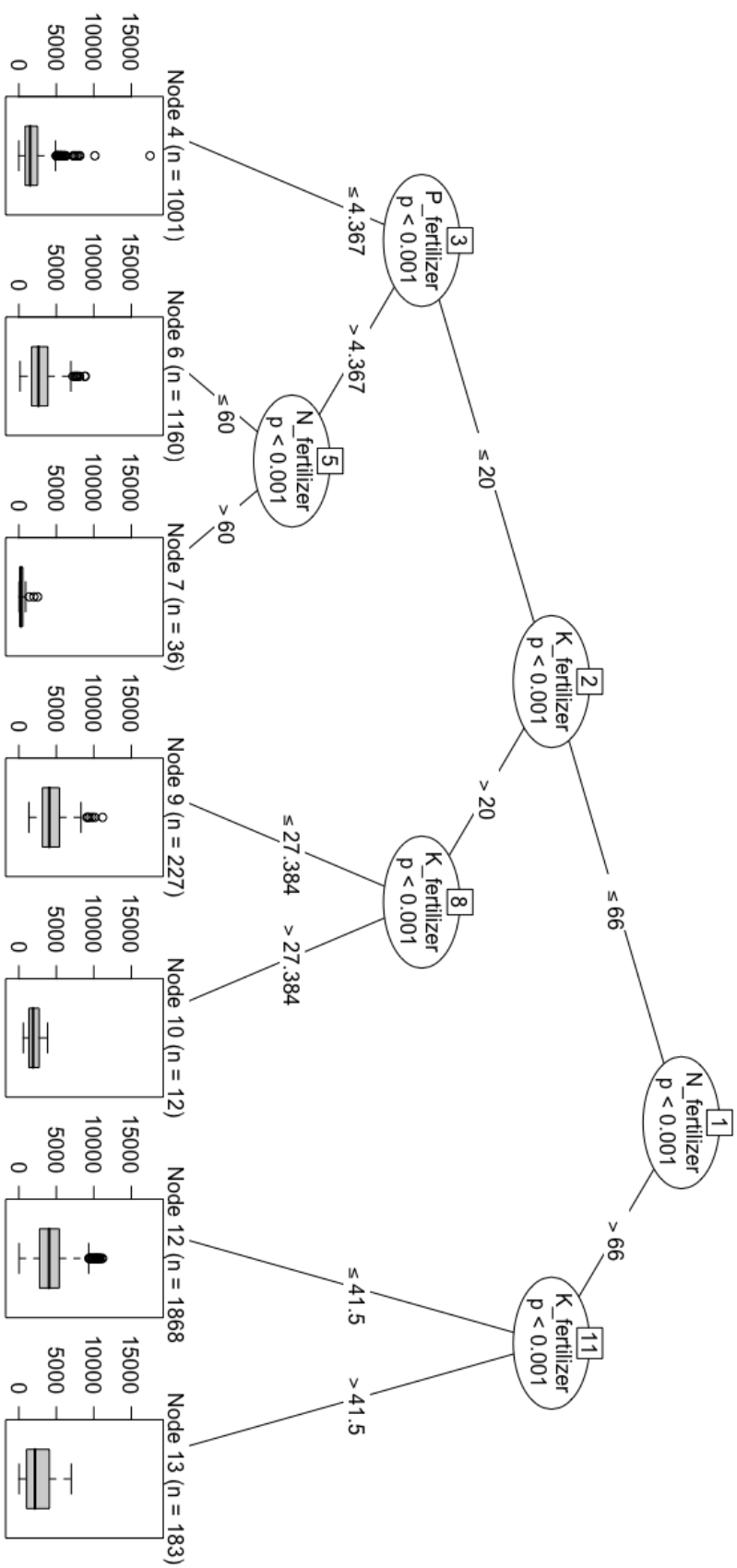



```
# P_fertilizer, K_fertilizer and N_fertilizer have very high cos2, which implies a good
↪  representation of the variable on the principal component. In this case variables are
↪  positioned close to the circumference of the correlation circle. N_fertilizer has
↪  slightly lowest cos2, which indicates that the variable is less good represented by
↪  the PCs. None of them is close to the center of the circle, so all are represented
↪  with the first components.

# Draw a bar plot of variable contributions for the most significant dimensions,
↪  therefore PC1 and PC2.

# Contributions of variables to PC1
a<-fviz_contrib(res.pca, choice = "var", axes = 1)
# Contributions of variables to PC2
b<-fviz_contrib(res.pca, choice = "var", axes = 2)
grid.arrange(a,b, ncol=2, top='Contribution of the variables to the first two PCs')
```




```
# Based on the pca analysis, we can reduce the dimension and build predictive statistics
↪  using machine learning models on the pca components (keeping the first most important
↪  ones). However, in this case, we've only taken 3 predictors, so we'll keep them for
↪  this exercise. In any case, I'm happy to do more advanced machine learning models,
↪  using the full list of predictors given in the data. In this case, PCA will be
↪  explored even further.
# ------------------------------------------------------------------------------------
# 4. MACHINE LEARNING (predictive statistics)

# I) Partition data into training and test sets (usually also validation set, but here
↪  for simplicity only 2). We use 70% of the data for training, and 30% for testing.

set.seed(2024)
split <- sample(1:nrow(carob_fertilizer_ML), as.integer(0.7*nrow(carob_fertilizer_ML)),
↪  F)
train <- carob_fertilizer_ML[split,]
test <- carob_fertilizer_ML[-split,]

# Here, we don't pre-process the data. These are continuous, non-categorical variables.
↪  What's more, since their units are the same kg/h, no normalization is required.
# From the train and test data, select the predictors x (fertilizers), and the target y
↪  (yield)
x_predictors=predictors[predictors!="yield"]

x_train=train[,x_predictors]
```

```r
x_test=test[,x_predictors]

y_train <- train[,"yield"]
y_test <- test[,"yield"]
# ------------------------------------------------------------------------------------
# II) Visualize exemplary algorithm

#Visualize exemplary algorithm. This step helps you understand what is going on when you
↪  subsequently train a more complex algorithm on your data. Let's strart with a
↪  decision tree here, because this is the foundation of more complex algorithms such as
↪  random forests or neural network. From the simple decision tree on our training data
↪  we can implement the following. Starting at the top, the most important predictor
↪  that can divide the data into the two most dissimilar subsets (in terms of expected
↪  yields) is "N_fertilizer", i.e. whether the amount applied is greater or less than 66
↪  kg/hr. If the quantity applied is greater (value > 66), we continue on the right-hand
↪  branch of the tree, otherwise we continue on the left-hand branch. If the quantity
↪  applied is greater (value > 66), we move on to the right-hand branch of the tree,
↪  otherwise we move on to the left-hand branch. For the right branch, the next
↪  important predictor is K_fertilizer, and the threshold to be considered here is:
↪  either <=41 (which leads to node 12 with a higher yield value), or >41.5 (which leads
↪  to node 13 with lower expected yield quantities). For the left branch, i.e.
↪  "N_fertilizer" <=66, the next predictor is K_fertilizer (with significant thresholds
↪  <=20 and >20 kg/h). When the K fertilizer applied is <=20, we must also consider P
↪  fertilizer as an option, with a significant threshold of 4.367 kg/h (a lower amount
↪  leads to node 4 and a higher amount leads to nodes 6 and 7). The combinations of
↪  different fertilizers with associated thresholds that lead to the worst yield
↪  prediction are associated with node 7. In the training data, there were 36 cases. The
↪  box-plots on the bottom of the chart show the statistics of the recored yields in the
↪  training data for the combination of different fertilizers and the corresponding
↪  applied quantitiy.


# DECISION TREE
set.seed(2024)
tree1 <- party::ctree(y_train ~ ., data=cbind(x_train, y_train),
                      controls = ctree_control(minsplit=12, mincriterion = .999))

#plot(tree1)

knitr::include_graphics("DecTree.png")
```

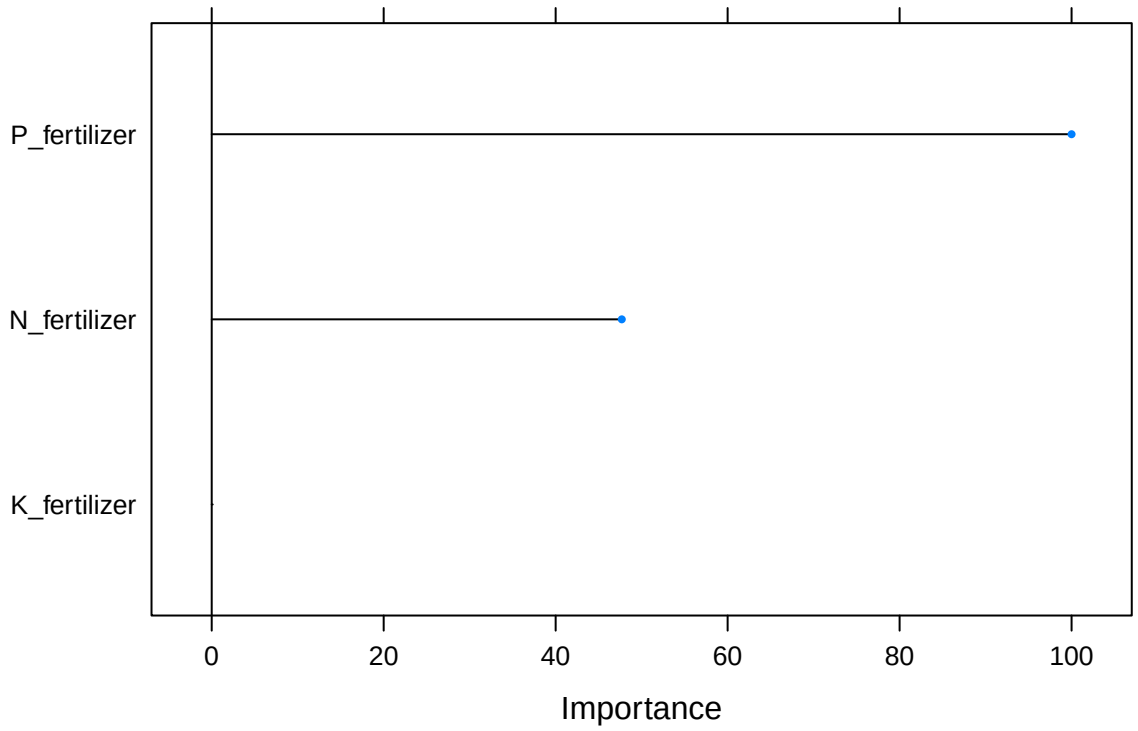

N_fertilizer
p < 0.001
1
≤ 66
> 66
K_fertilizer
p < 0.001
2
≤ 20
> 20
P_fertilizer
p < 0.001
3
≤ 4.367
> 4.367
K_fertilizer
p < 0.001
8
≤ 27.384
> 27.384
N_fertilizer
p < 0.001
5
≤ 60
> 60
K_fertilizer
p < 0.001
11
≤ 41.5
> 41.5
Node 4 (n = 1001)
Node 6 (n = 1160)
Node 7 (n = 36)
Node 9 (n = 227)
Node 10 (n = 12)
Node 12 (n = 1868)
Node 13 (n = 183)
15000
10000
5000
0

```r
# ------------------------------------------------------------------------------------
# III) Machine learning models
# a) RANDOM FOREST

#First we will run random forest which is basically an ensemble of many trees as the one
↪  we built in the previous section. The trick is that each tree is grown with only a
↪  random subset of all predictors considered at each node, and in the end all trees
↪  take a vote how to classify a specific patient. Taking a subset of all predictors at
↪  each run ensures that the trees are less correlated, i.e. not all of them use the
↪  same rules as the example tree shown above. If there are a few dominant predictors
↪  (N_fertilizer), then there will be some trees in our forest grown without these
↪  dominant predictors. These trees will be better able to classify the subgroup of
↪  predictors, for whatever reasons. Our first tree in the previous section would be
↪  confused about what to predict for yields associated for those predictors. However,
↪  in the random forest, there are trees that understand these cases as well. Thus, an
↪  ensemble of learners such as a random forest most often outperforms a single learner.

# Let's set the hyperparamter via the tuneGrid argument. The "mtry" parameter specifies
↪  how many of the predictors to consider at each split. We have 3 predictors in our
↪  training dataset, so if you set mtry == 3, then it's not a random forest any more,
↪  because all predictors are used and no random selection is applied. If you set mtry
↪  == 1, then the trees will be totally different from each other, but most ones will
↪  perform poorly because they are forced to use certain variables at the top split
↪  which are maybe not useful. The lower mtry, the more decorrelated the trees are, and
↪  the higher the value, the more predictors each tree can consider and thus the better
↪  the performance of a single tree. Somewhere between 1 and 3 is the optimal value, and
↪  there is no theoretical guidance as to which value should be taken. It depends on
↪  your data at hand, how correlated the predictors are, whether there are distinct
↪  sub-groups where the causal structure of the predictors works differently, and so on.
↪  The ideal will be to use mtry=2. Since for this exercise we have only 3 predictors, I
↪  will use mtry = c(1,2,3), which means I will make random forecast, from random trees
↪  that have all possible combinatons of predictors. Finding the best performance for
↪  different match for tuning hyperparameters is out of scope for this exercices.

# In "trainControl" we will specify "method = 'cv'" which stands for "cross validation".
↪  This means we will create another random split, splitting the training data into
↪  training and validation sets for the purpose of determining which algorithm works
↪  best on the training data. We set "number = 5", the function creates a validation set
↪  of size 1/5 of x_train and takes 4/5 of the data for training. "Cross validation"
↪  therefore repeats this training process and changes the validation set to another
↪  fifth of the data. This is done 5 times in total, so that all parts of the data
↪  served as validation set once, and then the results are averaged. This lets us use
↪  all of the training data and have train/validation splits to avoid overfitting.

set.seed(2024)
garbage <- capture.output(mod_rf <- caret::train(x_train, y_train, method="rf",
                  tuneGrid = expand.grid(mtry = seq(1,ncol(x_train),by=1)),
                  trControl = trainControl(method="cv", number=5, verboseIter = T),
                  verbose = FALSE))

#There are 3 predictors, 4487 samples. Three values were used for the hyperparameter
↪  "mtry". With each of them, 5-fold cross validation was performed. All values for mtry
↪  gives the same performances. On average (of the five runs using cross-validation),
↪  for each mtry value, the accuracy was obtained with a train/validation split into the
↪  training data, and the performance measured via RMSE, R2 and MAE.
```

```
mod_rf
```

```
## Random Forest
##
## 4487 samples
##    3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3589, 3588, 3590, 3591, 3590
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   1     1688.090  0.2608211  1311.828
##   2     1685.900  0.2625926  1307.014
##   3     1686.075  0.2624607  1306.730
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
#Predictor importance plot tells which of the variables were most often used as the
↪  important splits at the top of the trees. Unlike the single decision tree on all of
↪  the training data, where "N_fertilizer" was the most important predictor, across an
↪  ensemble of different trees, it's actually "P_fertilizer".

plot(varImp(mod_rf), main="Predictor importance of random forest model on training
↪  data",cex=0.5)
```

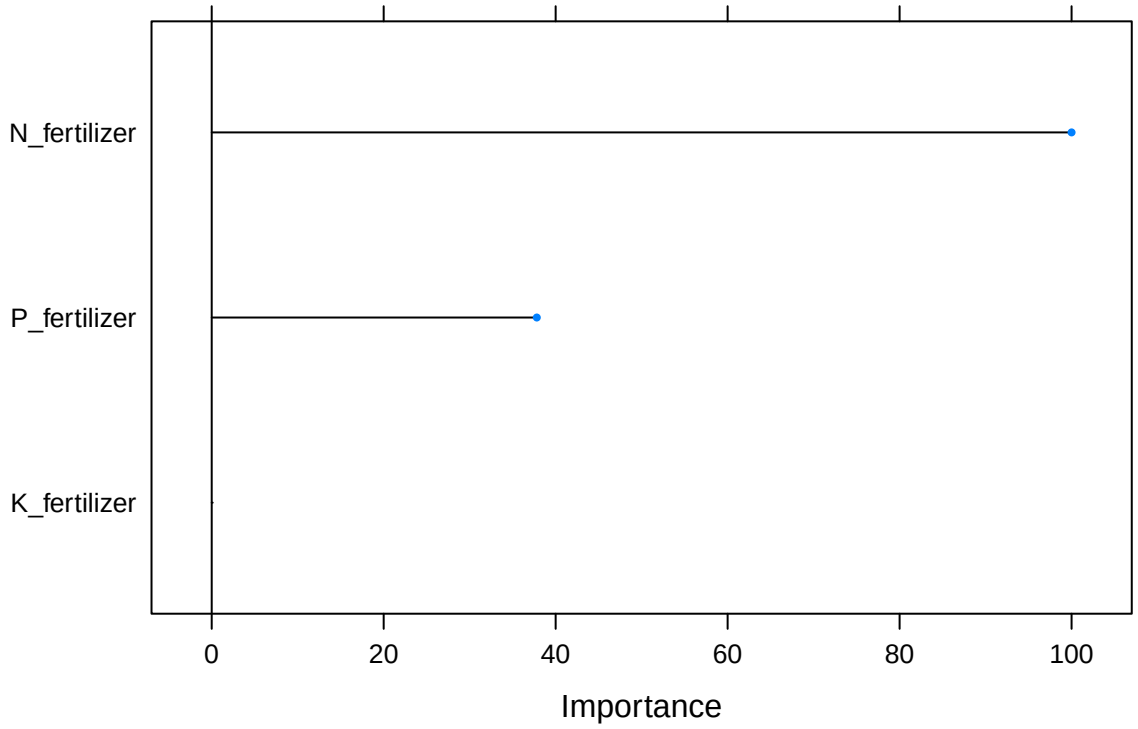



```
# b) NEURAL NETWORK
# Let's try out a neural network. Those deep learning methods can be useful for a complex
↪  relationship between predictors and targets.

#Here we use the pre-processing steps of centering and scaling the data because,neural
↪  networks are optimized more easily if the predictors have similar numerical ranges.
↪  Near-zero variance ("nzv") means that we disregard predictors where almost all
↪  fertilizers have the same value. Tree-based methods such as random forests are not as
↪  sensitive to these issues.

#We have a few more tuning parameters here. "Size" refers to the number of nodes in the
↪  hidden layer. Our network has an input layer of 3 nodes (i.e. the number of
↪  predictors) and an output layer with one node (target yield) and in between, a hidden
↪  layer where interactions between the predictors and non-linear transformations can be
↪  learned. As with other hyperparameters, the optimal size of the hidden layer(s)
↪  depend on the data, so we just try the same set of values like in RF. Decay is a
↪  regularization parameter that causes the weights of our nodes to decrease a bit after
↪  each round of updating the values after backpropagation (i.e. the opposite of what
↪  the learning rate does wich is used in other implementations of neural networks).
↪  What this means is, roughly speaking, we don't want the network to learn too
↪  ambitiously with each step of adapting its parameters to the evidence, in order to
↪  avoid overfitting. We have passed 3 different values for "size" to consider, 4 values
↪  for "decay", and two for "bag" (true or false, specifying how to aggregate several
↪  networks' predictions with various random number seeds, which is what the avNNet
↪  classifier does, bagging = bootstrap aggregating), so we have 3*4*2 = 24 combinations
↪  to try out.
```

```r
set.seed(2024)
garbage <- capture.output(mod_nn <- caret::train(x_train, y_train, method="avNNet",
                     preProcess = c("center", "scale", "nzv"),
                     tuneGrid = expand.grid(size = seq(1,3,by=1), decay=c(1e-03, 0.01,
                     ↪  0.1,0),bag=c(T,F)),
                     trControl = trainControl(method="cv", number=5, verboseIter = T),
                     importance=T))
mod_nn
```

```
## Model Averaged Neural Network
##
## 4487 samples
##    3 predictor
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3589, 3588, 3590, 3591, 3590
## Resampling results across tuning parameters:
##
##   size  decay  bag    RMSE      Rsquared     MAE
##   1     0.000  FALSE  3694.261          NaN  3129.53
##   1     0.000   TRUE  3694.261          NaN  3129.53
##   1     0.001  FALSE  3694.261          NaN  3129.53
##   1     0.001   TRUE  3694.261          NaN  3129.53
##   1     0.010  FALSE  3694.261          NaN  3129.53
##   1     0.010   TRUE  3694.261  0.103201353  3129.53
##   1     0.100  FALSE  3694.261          NaN  3129.53
##   1     0.100   TRUE  3694.261          NaN  3129.53
##   2     0.000  FALSE  3694.261          NaN  3129.53
##   2     0.000   TRUE  3694.261          NaN  3129.53
##   2     0.001  FALSE  3694.261          NaN  3129.53
##   2     0.001   TRUE  3694.261          NaN  3129.53
##   2     0.010  FALSE  3694.261  0.052619884  3129.53
##   2     0.010   TRUE  3694.261          NaN  3129.53
##   2     0.100  FALSE  3694.261          NaN  3129.53
##   2     0.100   TRUE  3694.261          NaN  3129.53
##   3     0.000  FALSE  3694.261          NaN  3129.53
##   3     0.000   TRUE  3694.261          NaN  3129.53
##   3     0.001  FALSE  3694.261          NaN  3129.53
##   3     0.001   TRUE  3694.261          NaN  3129.53
##   3     0.010  FALSE  3694.261          NaN  3129.53
##   3     0.010   TRUE  3694.261  0.006916439  3129.53
##   3     0.100  FALSE  3694.261          NaN  3129.53
##   3     0.100   TRUE  3694.261          NaN  3129.53
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1, decay = 0.1 and bag = FALSE.
```

```r
#As before we use predictor importance plot. Here, "N_fertilizer" was the most often used
↪  predictor.

plot(varImp(mod_nn), main="Feature importance of neural network classifier on training
↪  data",cex=0.5)
```

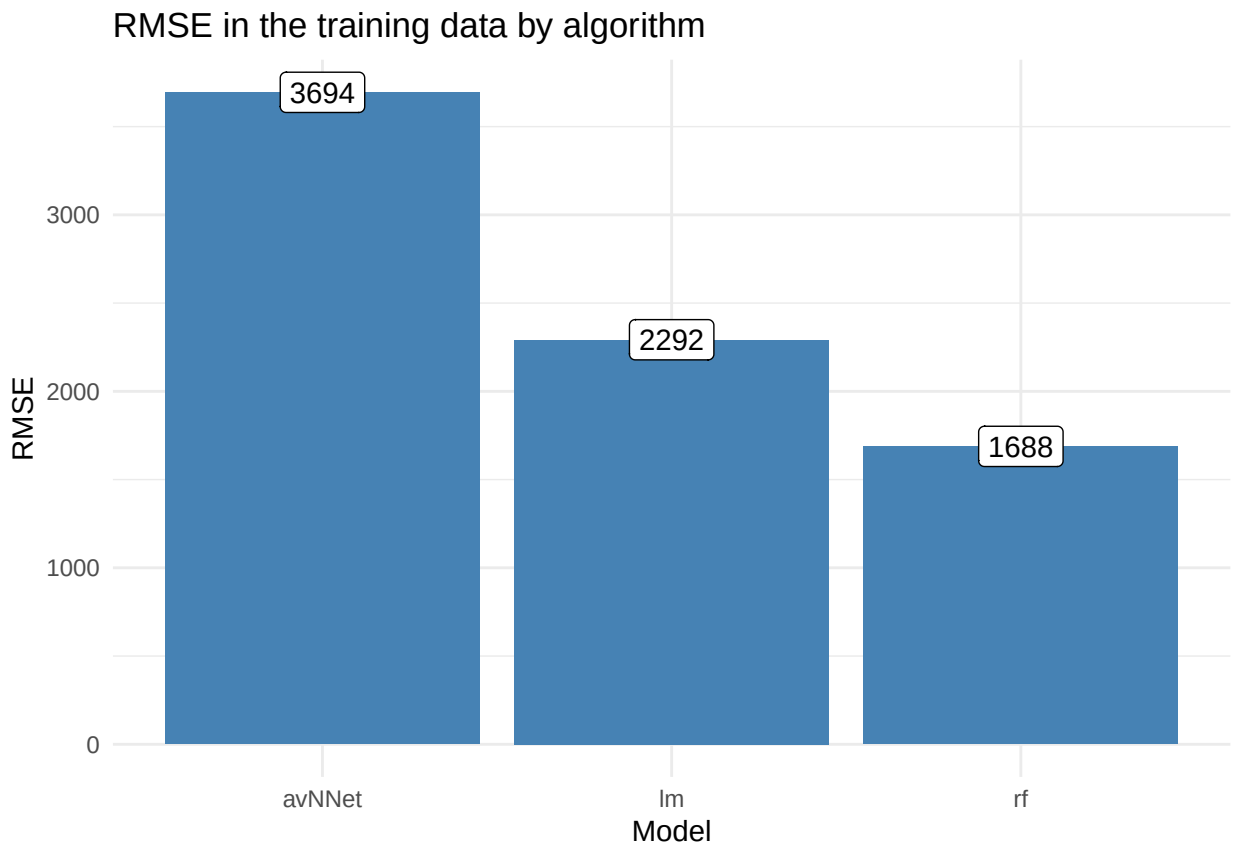



```
# C) LINEAR REGRESSION
lm.fit=lm(yield~N_fertilizer+P_fertilizer+K_fertilizer, data=train)
summary(lm.fit)
```

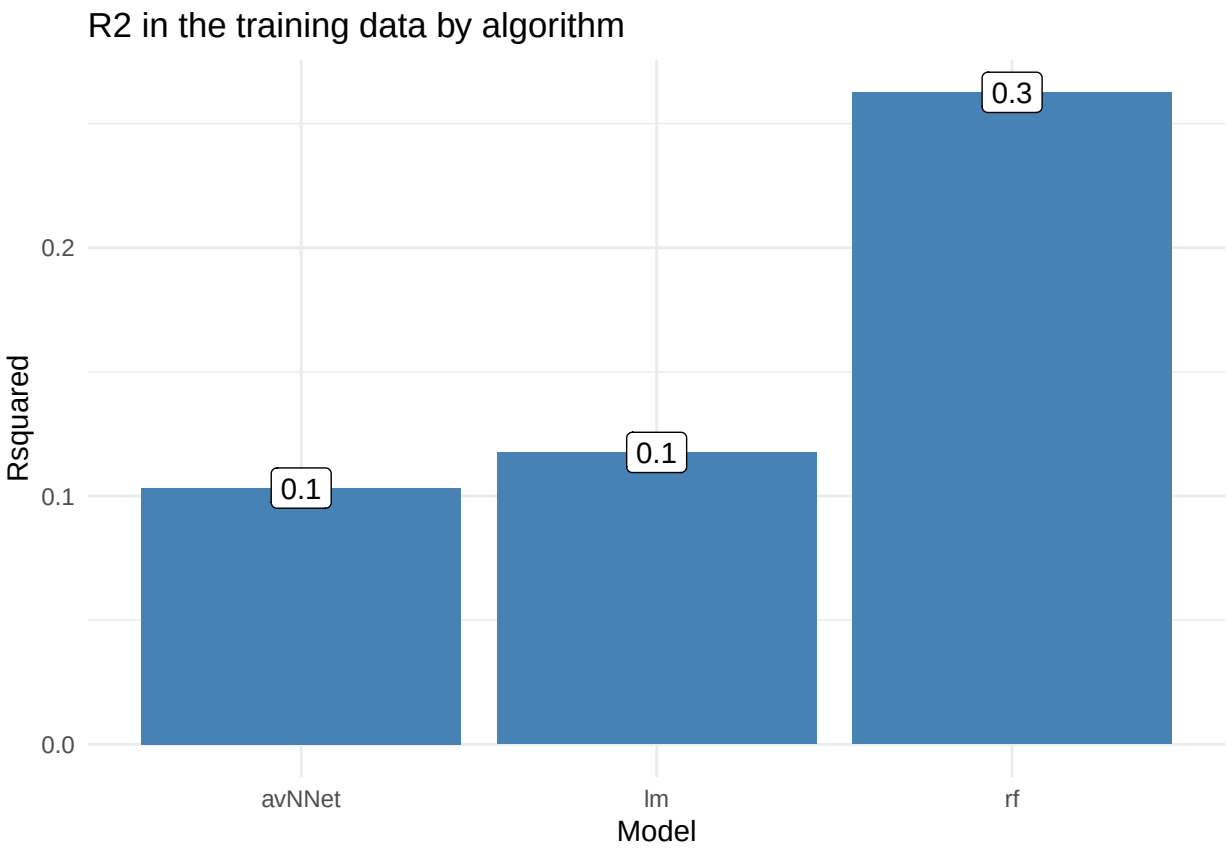
```
##
## Call:
## lm(formula = yield ~ N_fertilizer + P_fertilizer + K_fertilizer,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4069.1 -1308.8  -181.4  1081.8 10000.1
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2248.353     45.531  49.381  < 2e-16 ***
## N_fertilizer   10.243      0.963  10.637  < 2e-16 ***
## P_fertilizer   16.793      2.808   5.981 2.39e-09 ***
## K_fertilizer    0.258      2.120   0.122    0.903
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1845 on 4483 degrees of freedom
```

```
## Multiple R-squared:  0.1176, Adjusted R-squared:  0.117
## F-statistic: 199.1 on 3 and 4483 DF,  p-value: < 2.2e-16
```

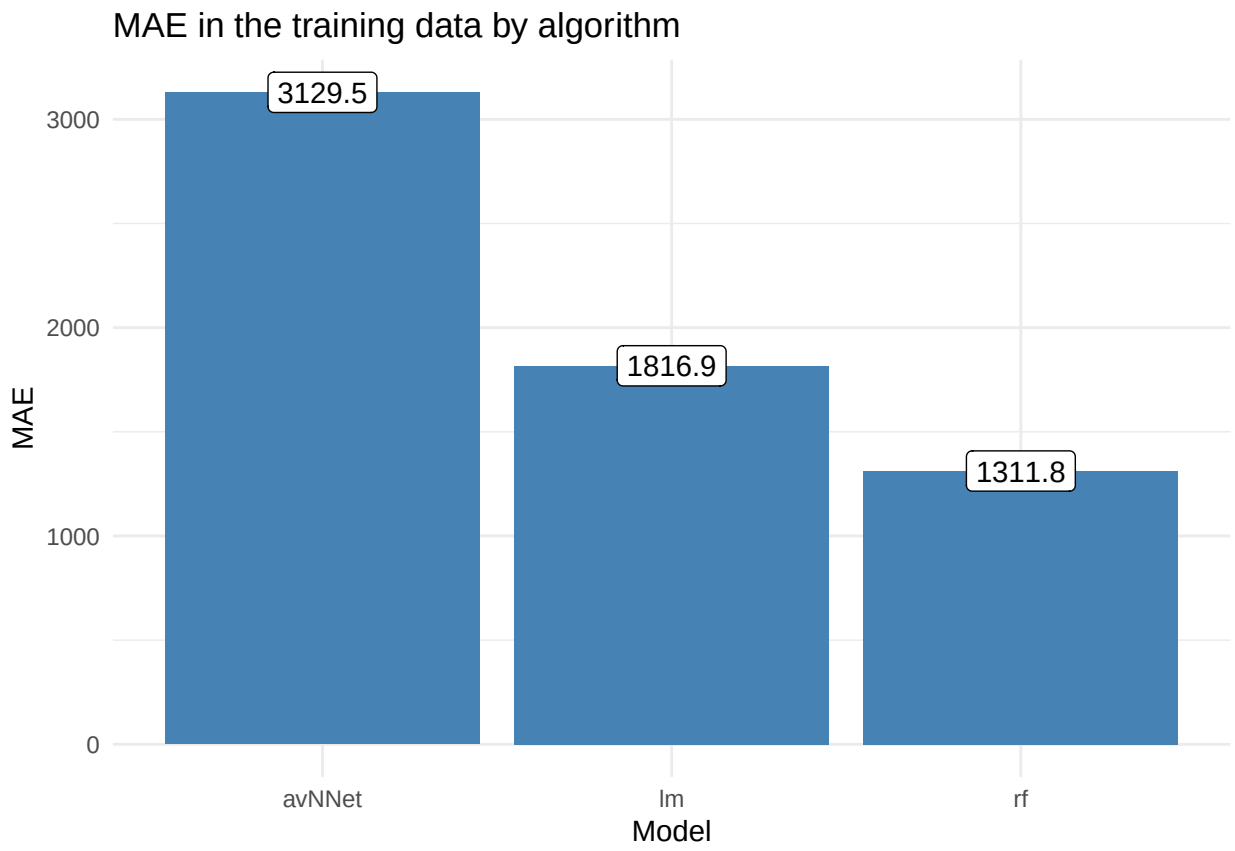
```
regressControl  <- trainControl(method="repeatedcv",
                                number = 4,
                                repeats = 5
)

mod_lm <- train(yield~N_fertilizer+P_fertilizer+K_fertilizer,
                data = train,
                method  = "lm",
                trControl = regressControl,
                tuneGrid  = expand.grid(intercept = FALSE))

# ------------------------------------------------------------------------------------------
# IV) Compare the performance of the three algorithms:

#Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction
↪  errors). Residuals are a measure of how far from the regression line data points are;
↪  RMSE is a measure of how to spread out these residuals are. In other words, it tells
↪  you how concentrated the data is around the line of best fit. RMSE value with zero
↪  indicates that the model has a perfect fit. The lower the RMSE, the better the model
↪  and its predictions. A higher RMSE indicates that there is a large deviation from the
↪  residual to the ground truth. From the histogram we can see that the RF outperform NN
↪  and LM.

results <- data.frame(Model = c(mod_rf$method,mod_nn$method, mod_lm$method),
                      RMSE = c(max(mod_rf$results$RMSE), max(mod_nn$results$RMSE),
                       ↪  max(mod_lm$results$RMSE)))
results %>% ggplot(aes(x=Model, y=RMSE, label=paste(round(RMSE)))) +
  geom_col(fill="steelblue") + theme_minimal() + geom_label() +
  ggtitle("RMSE in the training data by algorithm")
```




```
#R-Squared (R² or the coefficient of determination) is a statistical measure in a
↪  regression model that determines the proportion of variance in the dependent variable
↪  that can be explained by the independent variable. The standards for a good R-squared
↪  is to be close to 1. From the histogram we can see that the RF outperform NN and LM.

results <- data.frame(Model = c(mod_rf$method,mod_nn$method, mod_lm$method),
                      Rsquared = c(max(mod_rf$results$Rsquared),
                      ↪  max(na.omit(mod_nn$results$Rsquared)),
                      ↪  max(mod_lm$results$Rsquared)))
results %>% ggplot(aes(x=Model, y=Rsquared, label=paste(round(Rsquared,1)))) +
  geom_col(fill="steelblue") + theme_minimal() + geom_label() +
  ggtitle("R2 in the training data by algorithm")
```




```
# MAE stands for mean absolute error, which is a measure of how close your predictions
↪  are to the actual values in a regression problem. It is calculated by taking the
↪  average of the absolute differences between the predicted and the actual values for
↪  each observation in your data set. The closer MAE is to 0, the more accurate the
↪  model is. From the histogram we can see that the RF outperform NN and LM.

results <- data.frame(Model = c(mod_rf$method,mod_nn$method, mod_lm$method),
                      MAE = c(max(mod_rf$results$MAE), max(mod_nn$results$MAE),
                      ↪  max(mod_lm$results$MAE)))
results %>% ggplot(aes(x=Model, y=MAE, label=paste(round(MAE,1)))) +
  geom_col(fill="steelblue") + theme_minimal() + geom_label() +
  ggtitle("MAE in the training data by algorithm")
```




```
# REMARK! Even if RF shows better performance than NN and LM, I would still look for a
↪  model that gives better accuracy on the training data, i.e. R2 closer to 1 and RMSE
↪  and MAE closer to 0. This means that we should either include more predictors in the
↪  machine learning models, or think about another model that will be more appropriate
↪  for the data.

#V) EVALUATION ON "TEST" DATA
#Model evaluation against the test data. We now compare our model's prediction against
↪  the reserved test dataset. We use the random forest to predict the test data, and we
↪  compare the predictions against the actual outcomes.

predictions <- predict(mod_rf, newdata = x_test,na.action = na.pass)
RMSE=sqrt(mean((y_test - predictions)^2))
RMSE
```

```
## [1] 1720.687
```

```
MAE=mae(y_test, predictions)
MAE
```

```
## [1] 1330.36
```

```r
rss <- sum((predictions - y_test) ^ 2)  ## residual sum of squares
tss <- sum((y_test - mean(y_test)) ^ 2)  ## total sum of squares
R2 <- 1 - rss/tss
R2
```

```
## [1] 0.2832214
```

```r
#The evaluation metrics RMSE, MAE and R2 shows similar behaviour on test data as on
↪  training data.

# ------------------------------------------------------------------------------------
# VI) MODEL PREDICTIONS

# Once we have a good fitted model, we can make predictions for any set of predictors.
↪  Suppose a farmer is interested in predicting yields and plans to apply the following
↪  amounts of fertilizer.

new_data=data.frame(N_fertilizer=120,P_fertilizer=35,K_fertilizer=50)
predictions <- predict(mod_rf, newdata = new_data[1,],na.action = na.pass)
predictions # Estimated yield kg/h if applied 120 kg/h (N), 35kg/h (P) and 50kg/h (K)
↪  ferlilizer.
```

```
##        1
## 2061.901
```

Future work

This analysis is carried out for the first part of the exercises, i.e. it considers the yield response to N, P and K in the data, once we have the N, P and K values. This is advice for farmers, to estimate what maize yield will be. However, another model should be built to advise the farmer on how much N, P and K to use for a given location and under what weather conditions. To this end, we will take into account spatial variations as predictors (latitude, longitude) and spatial variations in soil and weather conditions in the region to predict site-specific N, P and K rates (multiple target results). In addition, variations in the relevant columns of soil and weather data need to be discussed with experts. As far as I understand from the data description, the following columns should be used as predictors (in addition to latitude and longitude): `Soil_N`, `Soil_P_total`,`Soil_K`,`rain`,`OM_N`,`OM_P`,`OM_K`, as well as columns that contains information about the irrigation and probably `leaf_N`,`leaf_P`,`leaf_K` and `grain_N`,`grain_P`,`grain_K`. This requires discussions with experts to better understand how to define the problem and select the most relevant predictors in the columns indicated in `colnames(carob_fertilizer)`. This is because some of the predictors are categoretic, such as yes/no rain or yes/no irrigation. Moreover, this is crucial for the choice of certain machine learning models, as well as for data pre-processing in the case of categoric inputs.