Statistical Learning Project

Contents

Author: Andrea Ierardi

Repository : Code link

Assignment

The exam consists in two assignments, one on the first part(regression, tree, neural nets) and the second part (unsupervised learning). For both you must prepare a writing report using one or more techniques and comparing their performance on one or more data set chosen by the student.

Libraries

```
library(knitr)
library(ggplot2)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(png)
library(ggpubr)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v tibble  3.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks plotly::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caTools)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## The following object is masked from 'package:plotly':
##
##     select
```

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ranger)
```

```
##
## Attaching package: 'ranger'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(tuneRanger)
```

```
## Loading required package: mlrMBO
```

```
## Loading required package: mlr
```

```
## Loading required package: ParamHelpers
## 'mlr' is in maintenance mode since July 2019. Future development
## efforts will go into its successor 'mlr3' (<https://mlr3.mlr-org.com>).
##
## Attaching package: 'mlr'
## The following object is masked from 'package:caret':
##
##     train
## Loading required package: smoof
## Loading required package: checkmate
## Loading required package: parallel
## Loading required package: lubridate
##
## Attaching package: 'lubridate'
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
## Loading required package: lhs
```

```
library(keras)

library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
library(clustMixType)
library(cluster)
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.13.4
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:ggpubr':
##
##     rotate
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(readr)

library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(FactoMineR)


library(PCAmixdata)
```

Dataset

Link of the dataset

```
ds = read.csv("AB_NYC_2019.csv")
```

Data Inspection

```
head(ds)
```

```
##     id                                              name host_id   host_name
## 1 2539               Clean & quiet apt home by the park    2787        John
## 2 2595                            Skylit Midtown Castle    2845    Jennifer
## 3 3647              THE VILLAGE OF HARLEM....NEW YORK !    4632   Elisabeth
## 4 3831                  Cozy Entire Floor of Brownstone    4869 LisaRoxanne
## 5 5022 Entire Apt: Spacious Studio/Loft by central park    7192       Laura
## 6 5099        Large Cozy 1 BR Apartment In Midtown East    7322       Chris
##   neighbourhood_group neighbourhood latitude longitude       room_type price
## 1            Brooklyn    Kensington 40.64749 -73.97237    Private room   149
## 2           Manhattan       Midtown 40.75362 -73.98377 Entire home/apt   225
## 3           Manhattan        Harlem 40.80902 -73.94190    Private room   150
## 4            Brooklyn  Clinton Hill 40.68514 -73.95976 Entire home/apt    89
## 5           Manhattan   East Harlem 40.79851 -73.94399 Entire home/apt    80
## 6           Manhattan   Murray Hill 40.74767 -73.97500 Entire home/apt   200
##   minimum_nights number_of_reviews last_review reviews_per_month
## 1              1                 9  2018-10-19              0.21
## 2              1                45  2019-05-21              0.38
## 3              3                 0                          NA
## 4              1               270  2019-07-05              4.64
## 5             10                 9  2018-11-19              0.10
```

```
## 6              3                74  2019-06-22              0.59
##   calculated_host_listings_count availability_365
## 1                              6              365
## 2                              2              355
## 3                              1              365
## 4                              1              194
## 5                              1                0
## 6                              1              129
```

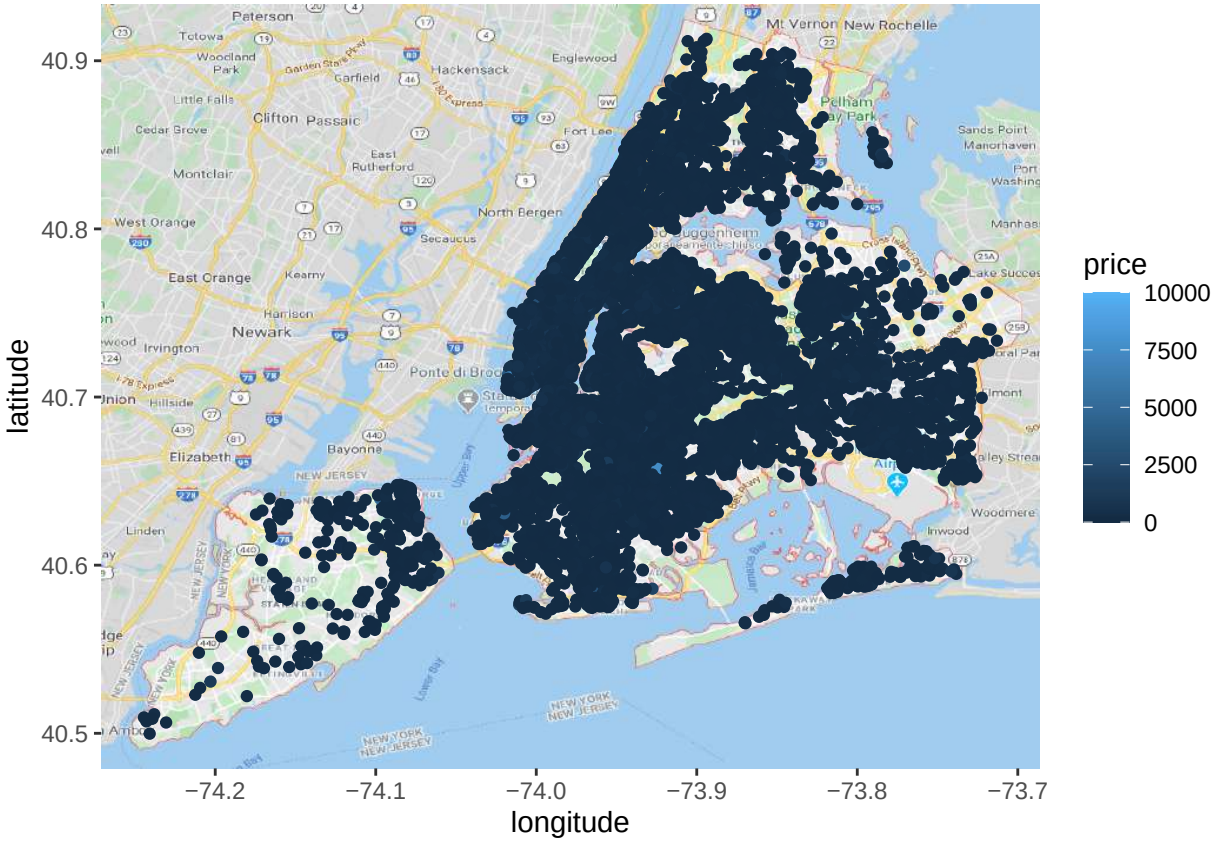
```
summary(ds)
```

```
##        id                 name              host_id           host_name
##  Min.   :    2539   Length:48895       Min.   :     2438   Length:48895
##  1st Qu.: 9471945   Class :character   1st Qu.:  7822033   Class :character
##  Median :19677284   Mode  :character   Median : 30793816   Mode  :character
##  Mean   :19017143                      Mean   : 67620011
##  3rd Qu.:29152178                      3rd Qu.:107434423
##  Max.   :36487245                      Max.   :274321313
##
##  neighbourhood_group neighbourhood         latitude       longitude
##  Length:48895        Length:48895       Min.   :40.50   Min.   :-74.24
##  Class :character    Class :character   1st Qu.:40.69   1st Qu.:-73.98
##  Mode  :character    Mode  :character   Median :40.72   Median :-73.96
##                                         Mean   :40.73   Mean   :-73.95
##                                         3rd Qu.:40.76   3rd Qu.:-73.94
##                                         Max.   :40.91   Max.   :-73.71
##
##   room_type             price         minimum_nights    number_of_reviews
##  Length:48895       Min.   :    0.0   Min.   :   1.00   Min.   :  0.00
##  Class :character   1st Qu.:   69.0   1st Qu.:   1.00   1st Qu.:  1.00
##  Mode  :character   Median :  106.0   Median :   3.00   Median :  5.00
##                     Mean   :  152.7   Mean   :   7.03   Mean   : 23.27
##                     3rd Qu.:  175.0   3rd Qu.:   5.00   3rd Qu.: 24.00
##                     Max.   :10000.0   Max.   :1250.00   Max.   :629.00
##
##  last_review        reviews_per_month calculated_host_listings_count
##  Length:48895       Min.   : 0.010    Min.   :  1.000
##  Class :character   1st Qu.: 0.190    1st Qu.:  1.000
##  Mode  :character   Median : 0.720    Median :  1.000
##                     Mean   : 1.373    Mean   :  7.144
##                     3rd Qu.: 2.020    3rd Qu.:  2.000
##                     Max.   :58.500    Max.   :327.000
##                     NA's   :10052
##  availability_365
##  Min.   :  0.0
##  1st Qu.:  0.0
##  Median : 45.0
##  Mean   :112.8
##  3rd Qu.:227.0
##  Max.   :365.0
##
```

```
cat("Shape of the dataset:" ,dim(ds))
```

```
## Shape of the dataset: 48895 16
```

```
img <- readPNG("map.png")


map_ds = ggplot() + background_image(img)+ geom_point(data = ds,  aes(y=latitude,x = longitude, color =
map_ds
```



Data cleaning

Check for NA and NULL values

```
apply(ds,2,function(x) sum(is.na(x)))
```

```
##                             id                           name
##                              0                              0
##                        host_id                      host_name
##                              0                              0
##           neighbourhood_group                  neighbourhood
##                              0                              0
##                       latitude                      longitude
##                              0                              0
```

```
##                       room_type                           price
##                               0                               0
##                  minimum_nights               number_of_reviews
##                               0                               0
##                     last_review              reviews_per_month
##                               0                           10052
## calculated_host_listings_count                availability_365
##                               0                               0
```

Variable selection

```
dataset = ds  %>% dplyr::select(neighbourhood_group,latitude, longitude, room_type,price)

head(dataset)
```

```
##   neighbourhood_group latitude longitude       room_type price
## 1            Brooklyn 40.64749 -73.97237    Private room   149
## 2           Manhattan 40.75362 -73.98377 Entire home/apt   225
## 3           Manhattan 40.80902 -73.94190    Private room   150
## 4            Brooklyn 40.68514 -73.95976 Entire home/apt    89
## 5           Manhattan 40.79851 -73.94399 Entire home/apt    80
## 6           Manhattan 40.74767 -73.97500 Entire home/apt   200
```

Variable scaling

```
scale_data = function(df)
{
  df = df %>%filter( price >= 15  & price <= 500)

  numerical = c("price")
  numerical2 = c("latitude","longitude")
  categorical = c("room_type", "neighbourhood_group")

  for( cat in categorical )
  {
    df[cat] = factor(df[[cat]],
                     level = unique(df[[cat]]),
                     labels = c(1:length(unique(df[[cat]])) ))
  }

  df[numerical] = as.numeric(scale(df[numerical]))


  df2 = df
  df2[numerical2] = as.numeric(scale(df2[numerical2]))
  df3 = list()
  df3$df2 = df2
  df3$df = df
```

```
  return(df3)


}
dataframe = scale_data(dataset)

dataset = dataframe$df

data = dataframe$df2

head(dataset)
```

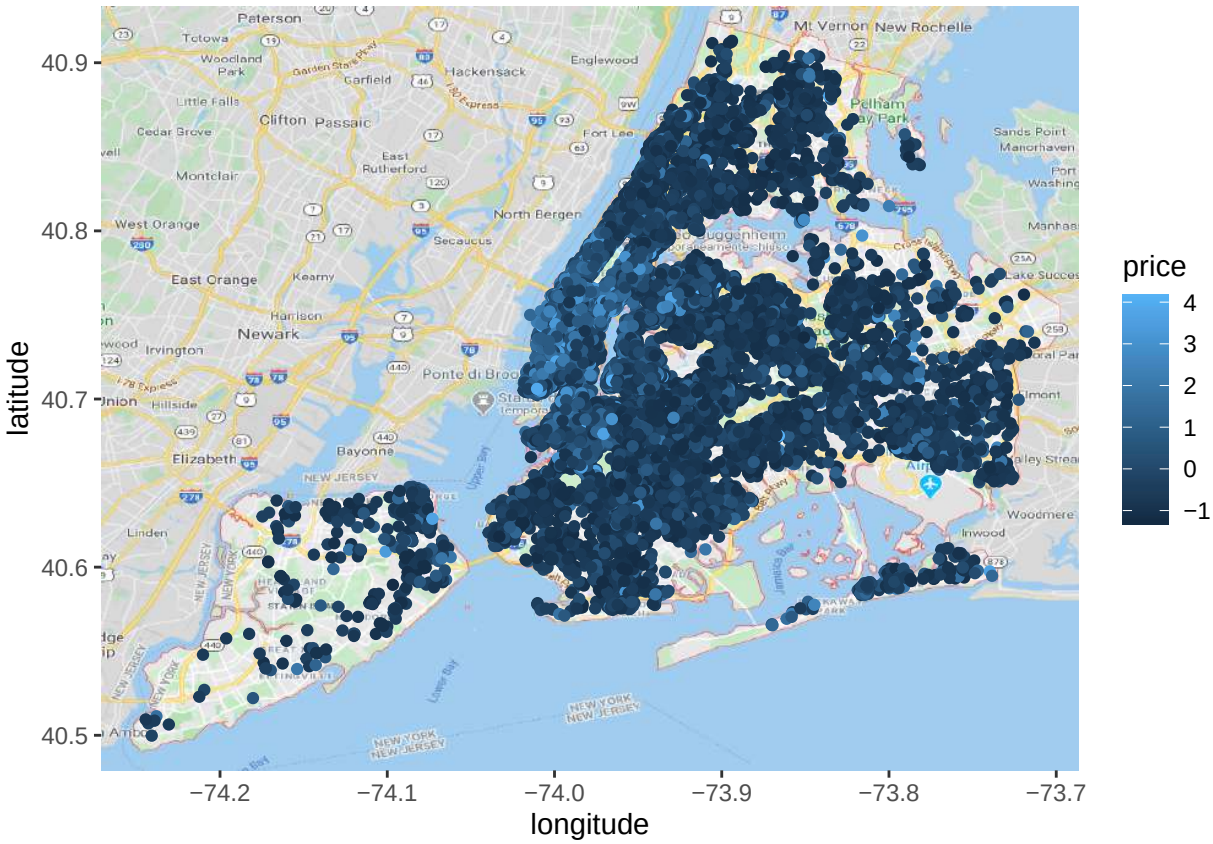
```
##   neighbourhood_group latitude longitude room_type      price
## 1                   1 40.64749 -73.97237         1  0.1973858
## 2                   2 40.75362 -73.98377         2  1.0607252
## 3                   2 40.80902 -73.94190         1  0.2087456
## 4                   1 40.68514 -73.95976         2 -0.4841979
## 5                   2 40.79851 -73.94399         2 -0.5864354
## 6                   2 40.74767 -73.97500         2  0.7767320
```

```
summary(dataset)
```

```
##  neighbourhood_group    latitude       longitude      room_type
##  1:19858             Min.   :40.50   Min.   :-74.24   1:22167
##  2:20877             1st Qu.:40.69   1st Qu.:-73.98   2:24503
##  3: 5632             Median :40.72   Median :-73.96   3: 1145
##  4:  366             Mean   :40.73   Mean   :-73.95
##  5: 1082             3rd Qu.:40.76   3rd Qu.:-73.94
##                      Max.   :40.91   Max.   :-73.71
##      price
##  Min.   :-1.3248
##  1st Qu.:-0.7228
##  Median :-0.3479
##  Mean   : 0.0000
##  3rd Qu.: 0.4587
##  Max.   : 4.1847
```

Data visualisation after the scaling

```
mappa = ggplot() + background_image(img)+ geom_point(data = dataset,  aes(y=latitude,x = longitude, col
mappa
```



Data split

Split data in subsets for each neighbourhood_group and room_type

```
neighbourhoods = unique(dataset$neighbourhood_group)

rooms = unique(dataset$room_type)

clust_data = vector("list")

lis_n = vector("list")

for (n in neighbourhoods)
{
  tmp = dataset %>%filter( neighbourhood_group == n)
  lis_n[[n]] = tmp[-1]

  tmp2 = data %>%filter( neighbourhood_group == n)
  clust_data[[n]] = tmp2[-1]
  clust_data[[n]]$room_type = factor(clust_data[[n]]$room_type , level = unique(clust_data[[n]]$room_ty
}
```

```
lis_r_n= vector("list")
for (n in neighbourhoods)
{
  for(r in rooms)
  {
    tmp = dataset %>%
      filter( room_type == r  & neighbourhood_group == n)
    lis_r_n[[paste0("n",n,"-","r",r)]]= tmp[-1][-3]


    tmp2 = data %>%
      filter( room_type == r  & neighbourhood_group == n)
    clust_data[[paste0("n",n,"-","r",r)]]= tmp2[-1][-3]

  }
}


data$neighbourhood_group = factor(data$neighbourhood_group  , level = unique(data$neighbourhood_group ) )
data$room_type = factor(data$room_type  , level = unique(data$room_type ) , labels= unique(ds$room_type)

clust_data[["all"]] = data
```

Split in train and test for each subset

```
trains = vector("list")
tests = vector("list")
datas = vector("list")

for (i in names(lis_n))
{
  sample = sample.split(lis_n[[i]], SplitRatio = .75)
  train = subset(lis_n[[i]], sample == TRUE)
  test  = subset(lis_n[[i]], sample == FALSE)
  trains[[i]]=  train
  tests[[i]] = test
  datas[[i]] = lis_n[[i]]
}

for (i in names(lis_r_n))
{
  sample = sample.split(lis_r_n[[i]], SplitRatio = .75)
  train = subset(lis_r_n[[i]], sample == TRUE)
  test  = subset(lis_r_n[[i]], sample == FALSE)
  trains[[i]]=  train
  tests[[i]] = test
  datas[[i]] = lis_r_n[[i]]

}
```

```
sample = sample.split(dataset, SplitRatio = .75)
train = subset(dataset, sample == TRUE)
test  = subset(dataset, sample == FALSE)

trains[["all"]] = train
tests[["all"]] = test
datas[["all"]] = dataset
```

MODELS TRAIN

```
model_lis = vector("list")
```

LINEAR REGRESSION

```
lin_reg = vector("list")

for (sub in names(trains))
{
  lin_reg[[sub]]$fit =lm.fit = lm(price~., data = trains[[sub]])
  lin_reg[[sub]]$summary = summary(lm.fit)

  lin_reg[[sub]]$pred  = pr.lm = predict(lm.fit,tests[[sub]])

  lin_reg[[sub]]$MSE = sum((pr.lm - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  print(paste0("========== ",sub, " =========="))
  print(summary(lm.fit))
  cat("\n\n")
}
```

```
## [1] "========== 1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8323 -0.3481 -0.1192  0.1750  5.3768
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -573.61243   20.31625 -28.234  < 2e-16 ***
## latitude       5.38922    0.20629  26.124  < 2e-16 ***
## longitude     -4.78267    0.22513 -21.244  < 2e-16 ***
## room_type2     0.96362    0.01128  85.441  < 2e-16 ***
## room_type3    -0.18322    0.03931  -4.661 3.17e-06 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6717 on 14888 degrees of freedom
## Multiple R-squared:  0.3869, Adjusted R-squared:  0.3867
## F-statistic:  2348 on 4 and 14888 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== 2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4058 -0.5348 -0.1896  0.2846  4.7774
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -845.59998   58.81126 -14.378  < 2e-16 ***
## latitude      -0.46594    0.35605  -1.309    0.191
## longitude    -11.68449    0.62124 -18.808  < 2e-16 ***
## room_type2     1.01872    0.01503  67.766  < 2e-16 ***
## room_type3    -0.26290    0.04863  -5.406 6.52e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8837 on 15653 degrees of freedom
## Multiple R-squared:  0.3343, Adjusted R-squared:  0.3341
## F-statistic:  1965 on 4 and 15653 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== 3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3742 -0.2988 -0.1270  0.1391  5.0902
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.94589   12.53702  -0.634  0.52625
## latitude    -0.67106    0.28710  -2.337  0.01947 *
## longitude   -0.46762    0.20025  -2.335  0.01958 *
## room_type2   0.80929    0.01951  41.478  < 2e-16 ***
## room_type3  -0.17440    0.05165  -3.377  0.00074 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6028 on 4219 degrees of freedom
```

```
## Multiple R-squared:  0.3044, Adjusted R-squared:  0.3037
## F-statistic: 461.5 on 4 and 4219 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== 4 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9120 -0.3348 -0.1459  0.2172  3.6564
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -30.688887 144.852858  -0.212    0.832
## latitude      0.734021   1.511517   0.486    0.628
## longitude    -0.001178   1.331628  -0.001    0.999
## room_type2    0.739498   0.078599   9.408   <2e-16 ***
## room_type3    0.159594   0.292213   0.546    0.585
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6387 on 269 degrees of freedom
## Multiple R-squared:  0.2493, Adjusted R-squared:  0.2382
## F-statistic: 22.34 on 4 and 269 DF,  p-value: 6.126e-16
##
##
##
## [1] "========== 5 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9669 -0.2906 -0.1359  0.1124  4.9772
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 62.09595   75.12511   0.827    0.409
## latitude    -0.46076    0.89539  -0.515    0.607
## longitude    0.59638    0.72879   0.818    0.413
## room_type2   0.67380    0.04732  14.238   <2e-16 ***
## room_type3  -0.15156    0.09529  -1.591    0.112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6266 on 806 degrees of freedom
## Multiple R-squared:  0.2199, Adjusted R-squared:  0.216
## F-statistic: 56.79 on 4 and 806 DF,  p-value: < 2.2e-16
##
##
```

```
##
## [1] "========== n1-r1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7175 -0.2367 -0.0890  0.1093  5.1328
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -351.5686    19.4645  -18.06   <2e-16 ***
## latitude       2.8562     0.1986   14.38   <2e-16 ***
## longitude     -3.1736     0.2130  -14.90   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4244 on 6721 degrees of freedom
## Multiple R-squared:  0.0483, Adjusted R-squared:  0.04802
## F-statistic: 170.6 on 2 and 6721 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== n1-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9631 -0.5724 -0.2106  0.2963  4.4458
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -764.9500    39.6271  -19.30   <2e-16 ***
## latitude       8.2186     0.4068   20.20   <2e-16 ***
## longitude     -5.8264     0.4454  -13.08   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8599 on 6238 degrees of freedom
## Multiple R-squared:  0.07359,    Adjusted R-squared:  0.07329
## F-statistic: 247.7 on 2 and 6238 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== n1-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -0.43709 -0.22328 -0.13609  0.05886  2.72639
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -322.3727    94.7421  -3.403 0.000769 ***
## latitude       2.8413     0.7927   3.584 0.000401 ***
## longitude     -2.7841     1.0335  -2.694 0.007504 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4409 on 270 degrees of freedom
## Multiple R-squared:  0.05102,    Adjusted R-squared:  0.04399
## F-statistic: 7.258 on 2 and 270 DF,  p-value: 0.0008505
##
##
##
## [1] "========== n2-r1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2007 -0.3766 -0.1682  0.1281  4.7297
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -673.3536    81.3847  -8.274   <2e-16 ***
## latitude      -0.5113     0.4675  -1.094    0.274
## longitude     -9.3809     0.8670 -10.820   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6845 on 5252 degrees of freedom
## Multiple R-squared:  0.1039, Adjusted R-squared:  0.1036
## F-statistic: 304.5 on 2 and 5252 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== n2-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4464 -0.6834 -0.2428  0.4327  3.8755
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -888.5411    89.7697  -9.898   <2e-16 ***
## latitude      -0.9338     0.5683  -1.643      0.1
## longitude    -12.5366     0.9398 -13.339   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9986 on 8343 degrees of freedom
## Multiple R-squared:  0.07902,    Adjusted R-squared:  0.0788
## F-statistic: 357.9 on 2 and 8343 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== n2-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8824 -0.3980 -0.2476  0.0510  4.5630
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1089.024    345.497  -3.152 0.001778 **
## latitude        3.921      2.098   1.869 0.062580 .
## longitude     -12.554      3.676  -3.415 0.000723 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7959 on 313 degrees of freedom
## Multiple R-squared:  0.04358,    Adjusted R-squared:  0.03747
## F-statistic: 7.132 on 2 and 313 DF,  p-value: 0.0009361
##
##
##
## [1] "========== n3-r1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5259 -0.2303 -0.0979  0.1135  4.6236
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.6830    12.6014  -1.483   0.1383
## latitude     -0.2314     0.2962  -0.781   0.4346
## longitude    -0.3707     0.1963  -1.888   0.0591 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4295 on 2239 degrees of freedom
## Multiple R-squared:  0.001641,   Adjusted R-squared:  0.0007496
## F-statistic: 1.841 on 2 and 2239 DF,  p-value: 0.159
##
##
##
```

```
## [1] "========== n3-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3548 -0.5229 -0.2095  0.2175  4.1989
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -14.6167    28.3789  -0.515   0.6066
## latitude     -1.2073     0.6411  -1.883   0.0599 .
## longitude    -0.8644     0.4822  -1.793   0.0732 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8256 on 1381 degrees of freedom
## Multiple R-squared:  0.003026,   Adjusted R-squared:  0.001582
## F-statistic: 2.096 on 2 and 1381 DF,  p-value: 0.1233
##
##
##
## [1] "========== n3-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4426 -0.2340 -0.1254  0.0081  5.0332
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -113.4214    81.6764  -1.389    0.167
## latitude       2.9879     2.0923   1.428    0.156
## longitude      0.1248     1.2873   0.097    0.923
##
## Residual standard error: 0.5843 on 126 degrees of freedom
## Multiple R-squared:  0.0226, Adjusted R-squared:  0.007089
## F-statistic: 1.457 on 2 and 126 DF,  p-value: 0.2368
##
##
##
## [1] "========== n4-r1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4399 -0.2616 -0.1008  0.1382  2.7047
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  41.8452   140.4584   0.298    0.766
## latitude     -1.3294     1.5633  -0.850    0.397
## longitude    -0.1532     1.3589  -0.113    0.910
##
## Residual standard error: 0.4341 on 123 degrees of freedom
## Multiple R-squared:  0.008143,   Adjusted R-squared:  -0.007985
## F-statistic: 0.5049 on 2 and 123 DF,  p-value: 0.6048
##
##
##
## [1] "========== n4-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9076 -0.5490 -0.2720  0.2208  3.6389
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   40.350    296.277   0.136    0.892
## latitude       1.207      2.991   0.403    0.687
## longitude      1.207      2.710   0.445    0.657
##
## Residual standard error: 0.8287 on 110 degrees of freedom
## Multiple R-squared:  0.009854,   Adjusted R-squared:  -0.008149
## F-statistic: 0.5474 on 2 and 110 DF,  p-value: 0.58
##
##
##
## [1] "========== n4-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##        1        3        4        6        7
##  0.05210  0.05313  0.06896  0.03662 -0.21082
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1780.567   3072.654  -0.579    0.621
## latitude       26.744     13.141   2.035    0.179
## longitude      -9.363     35.224  -0.266    0.815
##
## Residual standard error: 0.1675 on 2 degrees of freedom
## Multiple R-squared:  0.8503, Adjusted R-squared:  0.7006
## F-statistic:  5.68 on 2 and 2 DF,  p-value: 0.1497
##
##
##
## [1] "========== n5-r1 =========="
```

```
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4612 -0.2341 -0.0916  0.1272  4.9178
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  88.3541    73.8984   1.196   0.2325
## latitude     -1.9505     0.8424  -2.315   0.0211 *
## longitude     0.1284     0.7459   0.172   0.8634
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4251 on 429 degrees of freedom
## Multiple R-squared:  0.01321,    Adjusted R-squared:  0.00861
## F-statistic: 2.872 on 2 and 429 DF,  p-value: 0.05769
##
##
##
## [1] "========== n5-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1899 -0.4834 -0.2201  0.1913  4.2329
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -100.40099  164.61869  -0.610    0.542
## latitude       2.57087    2.05784   1.249    0.213
## longitude      0.06421    1.54657   0.042    0.967
##
## Residual standard error: 0.8279 on 248 degrees of freedom
## Multiple R-squared:  0.007378,   Adjusted R-squared:  -0.0006266
## F-statistic: 0.9217 on 2 and 248 DF,  p-value: 0.3992
##
##
##
## [1] "========== n5-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40836 -0.14435 -0.05595  0.06429  1.04848
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   77.152    186.703   0.413    0.682
## latitude       1.712      2.084   0.822    0.417
## longitude      2.004      1.707   1.174    0.248
##
## Residual standard error: 0.2919 on 36 degrees of freedom
## Multiple R-squared:  0.1093, Adjusted R-squared:  0.05986
## F-statistic:  2.21 on 2 and 36 DF,  p-value: 0.1244
##
##
##
## [1] "========== all =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2177 -0.4433 -0.1399  0.2223  5.0722
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -2.038e+02  1.411e+01 -14.446  < 2e-16 ***
## neighbourhood_group2  4.855e-01  1.608e-02  30.189  < 2e-16 ***
## neighbourhood_group3  2.352e-01  1.963e-02  11.979  < 2e-16 ***
## neighbourhood_group4 -8.552e-01  5.697e-02 -15.012  < 2e-16 ***
## neighbourhood_group5  2.821e-01  3.838e-02   7.350 2.04e-13 ***
## latitude             -1.413e+00  1.382e-01 -10.227  < 2e-16 ***
## longitude            -3.524e+00  1.587e-01 -22.204  < 2e-16 ***
## room_type2            9.844e-01  9.582e-03 102.736  < 2e-16 ***
## room_type3           -2.466e-01  3.070e-02  -8.032 9.94e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7784 on 28680 degrees of freedom
## Multiple R-squared:  0.3889, Adjusted R-squared:  0.3888
## F-statistic:  2282 on 8 and 28680 DF,  p-value: < 2.2e-16
```

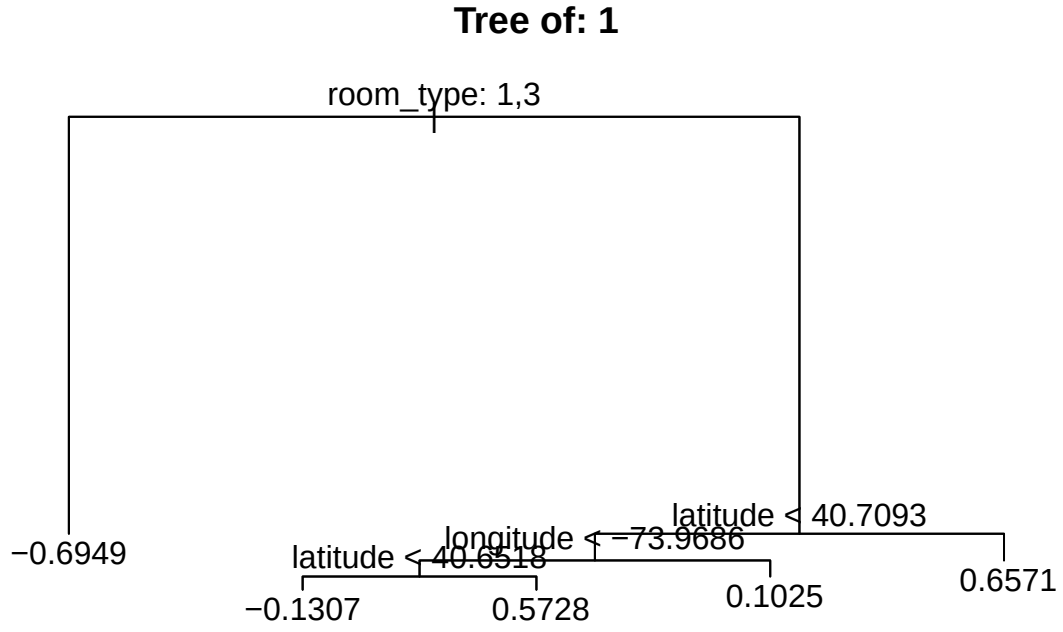
```
lin_reg$name = "Linear Regression"
model_lis$linear_regression=  lin_reg
```

DECISION TREE

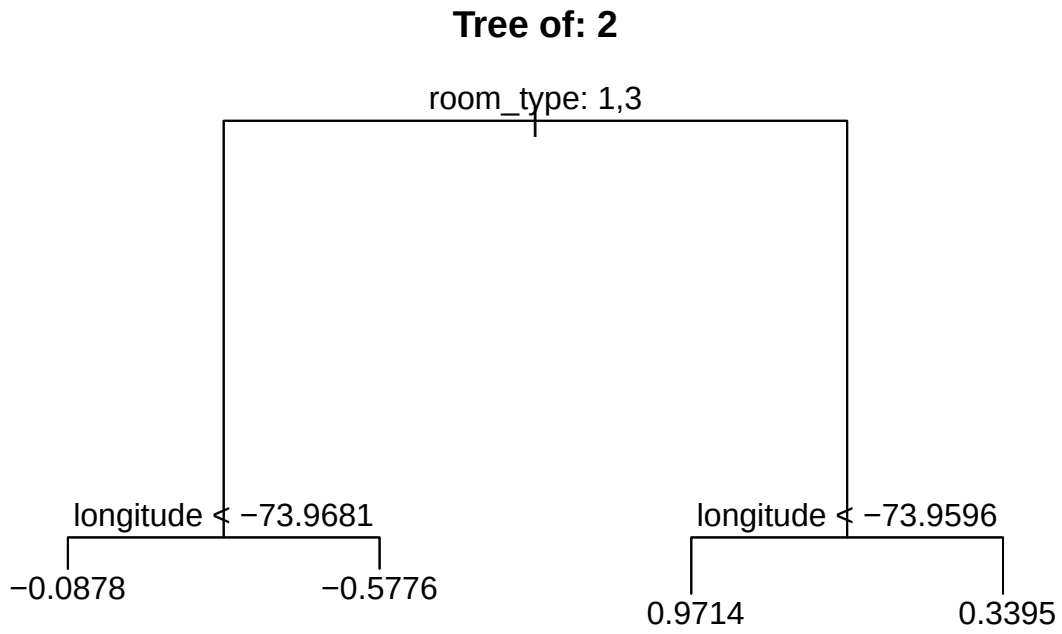
```
dec_tree = vector("list")

for (sub in names(trains))
{
  dec_tree[[sub]]$fit = tree_res=tree(price~., data = trains[[sub]])
  dec_tree[[sub]]$summary = sum = summary(tree_res)

  print(paste0("========== ",sub, " =========="))
```

```
  print(sum)

  if(sum$size > 1 )
  {
    plot(tree_res)
    text(tree_res,pretty=0)
    title(paste0("Tree of: ",sub))


  }
  else
  {
    cat("Not possible to plot tree: ", sub)
  }

  dec_tree[[sub]]$pred  = pred = predict(tree_res,tests[[sub]])

  dec_tree[[sub]]$MSE = mse =  sum((pred - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  print(mse)
  cat("\n\n")
}
```

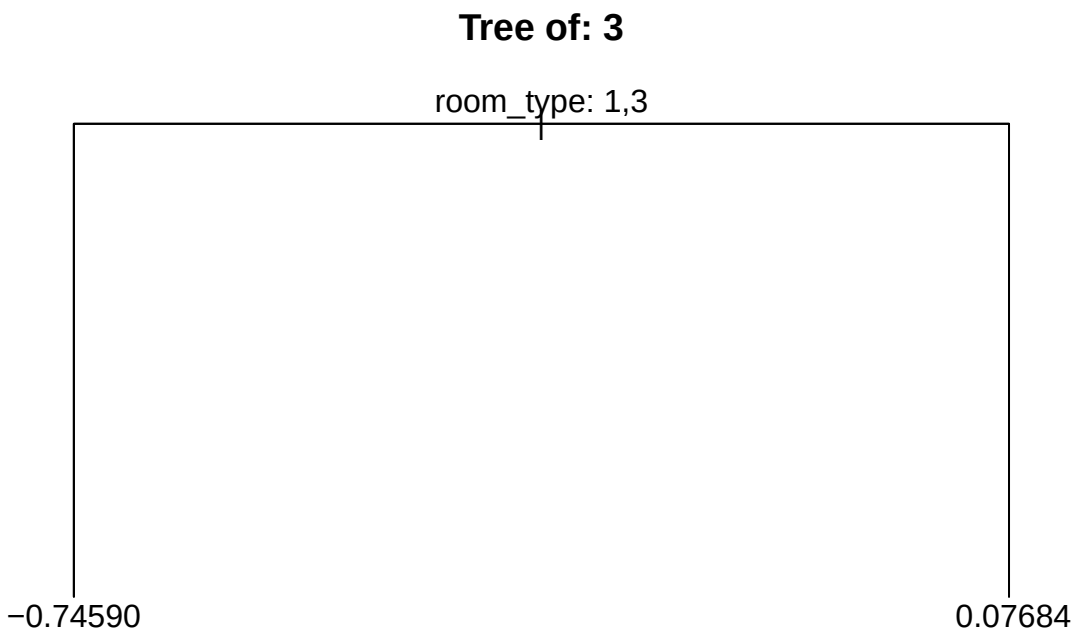
```
## [1] "========== 1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  5
## Residual mean deviance:  0.4464 = 6645 / 14890
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9250 -0.3460 -0.1188  0.0000  0.1652  4.8800
```



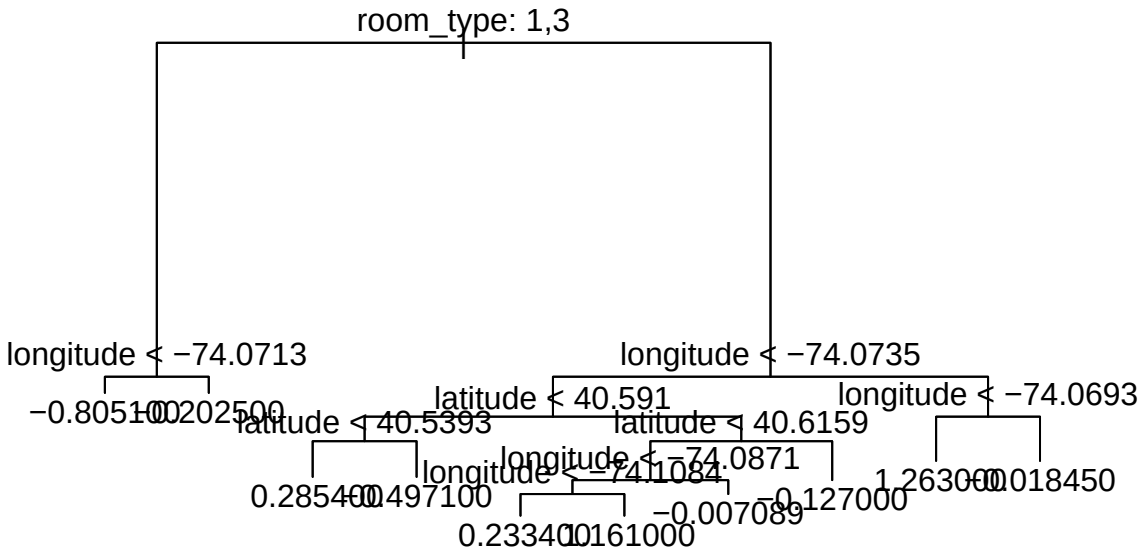

```
## [1] 0.4607817
##
##
## [1] "========== 2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type" "longitude"
## Number of terminal nodes:  4
## Residual mean deviance:  0.7869 = 12320 / 15650
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.2850 -0.5200 -0.1947  0.0000  0.2965  4.7620
```



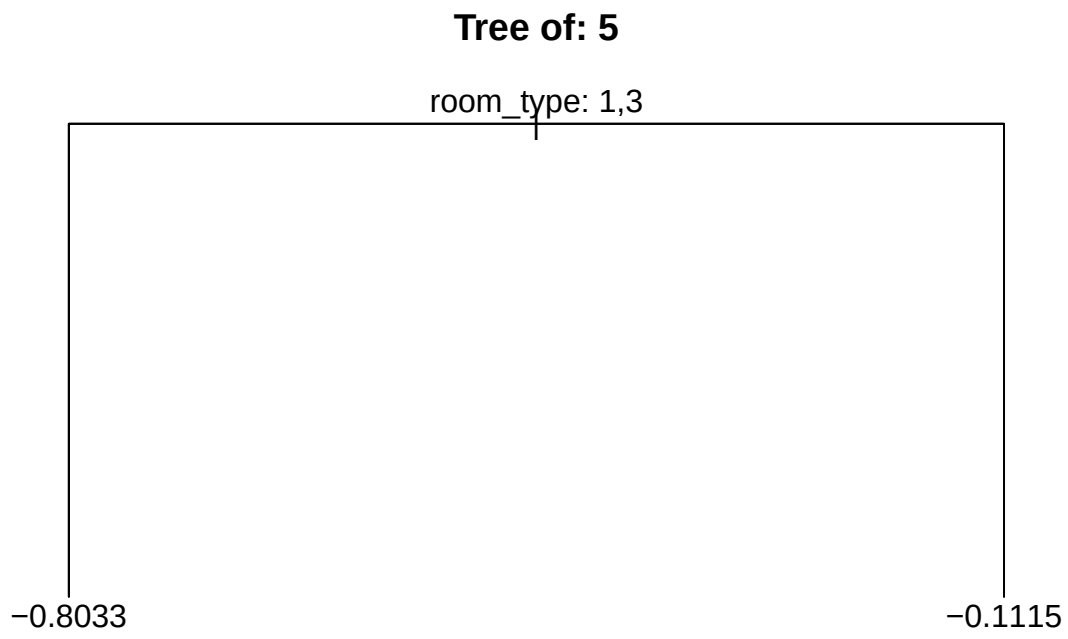

```
## [1] 0.7634093
##
##
## [1] "========== 3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type"
## Number of terminal nodes:  2
## Residual mean deviance:  0.3647 = 1540 / 4222
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.3900 -0.3014 -0.1245  0.0000  0.1319  4.9310
```



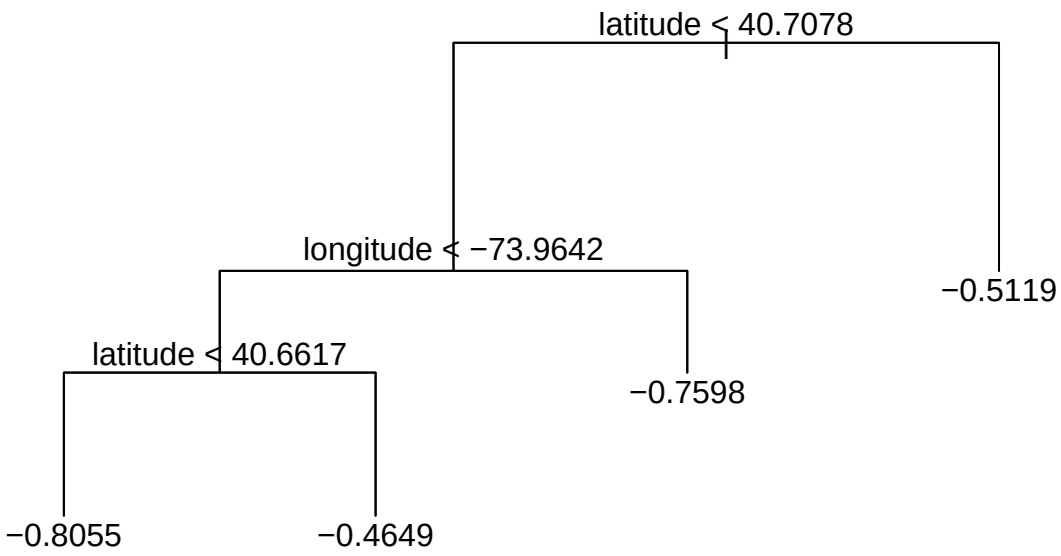

```
## [1] 0.3864543
##
##
## [1] "========== 4 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  10
## Residual mean deviance:  0.3202 = 84.54 / 264
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.3500 -0.3039 -0.1221  0.0000  0.2187  2.3540
```



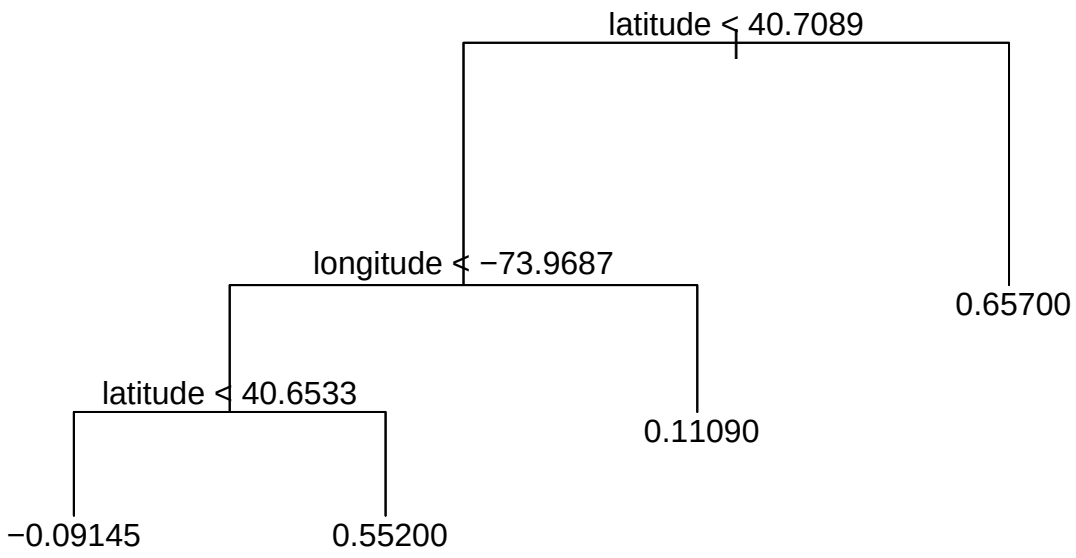

```
## [1] 0.3471879
##
##
## [1] "========== 5 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type"
## Number of terminal nodes:  2
## Residual mean deviance:  0.3927 = 317.7 / 809
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.9862 -0.2943 -0.1239  0.0000  0.1033  4.9880
```



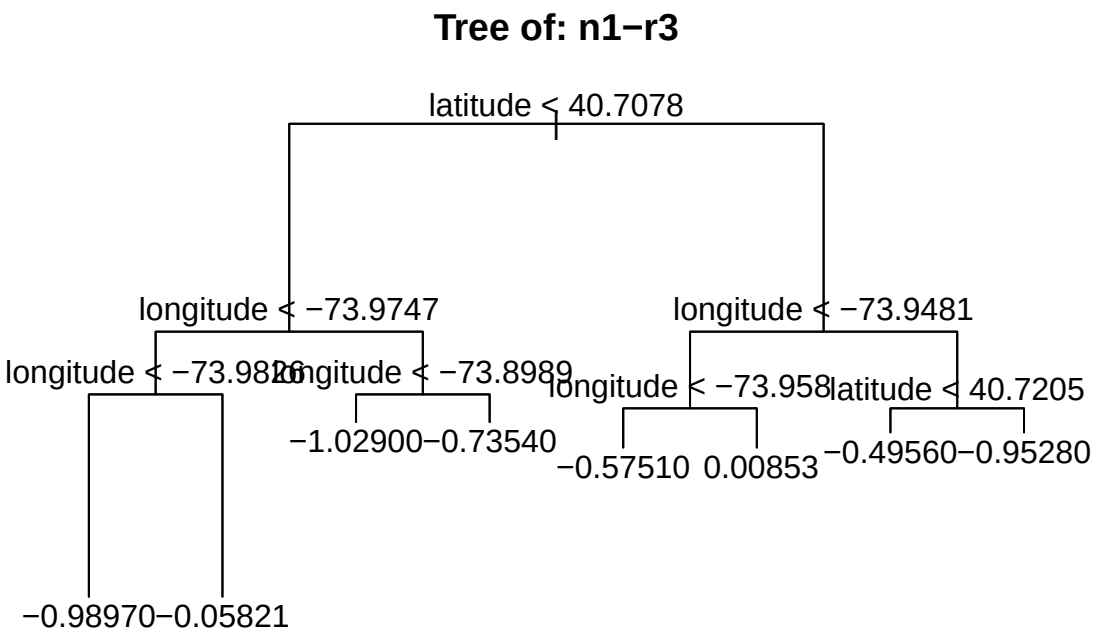

```
## [1] 0.2744778
##
##
## [1] "========== n1-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.1738 = 1168 / 6720
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.7561 -0.2242 -0.0745  0.0000  0.1166  4.9440
```



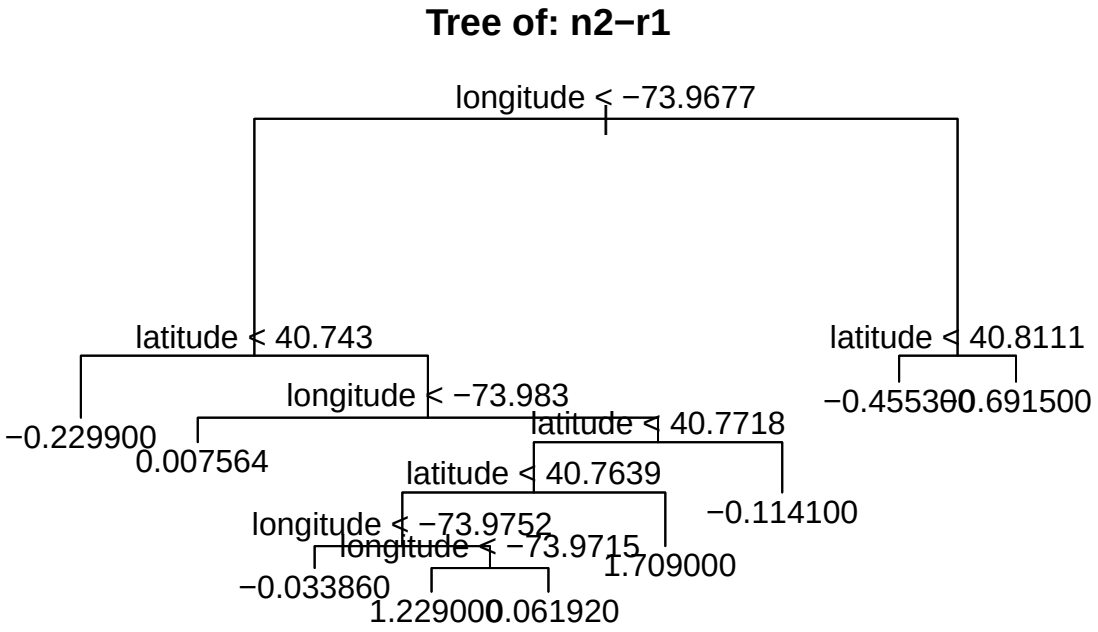

```
## [1] 0.1771088
##
##
## [1] "========== n1-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.731 = 4559 / 6237
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9250 -0.5518 -0.2070  0.0000  0.3002  4.0740
```



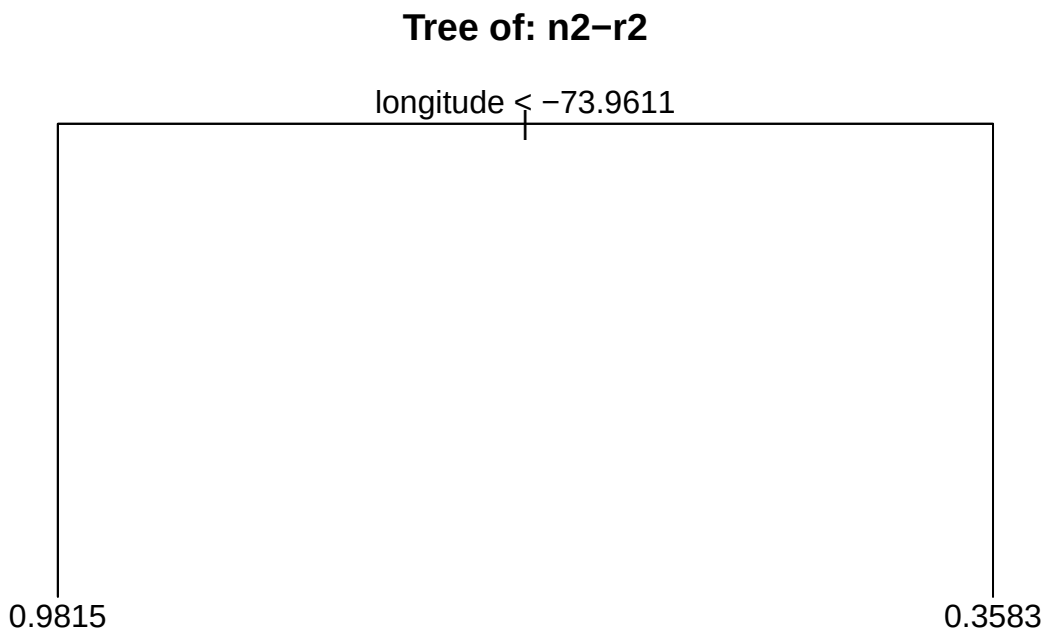

```
## [1] 0.7503308
##
##
## [1] "========== n1-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  8
## Residual mean deviance:  0.1442 = 38.21 / 265
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.02700 -0.15190 -0.06911  0.00000  0.04448  2.05000
```



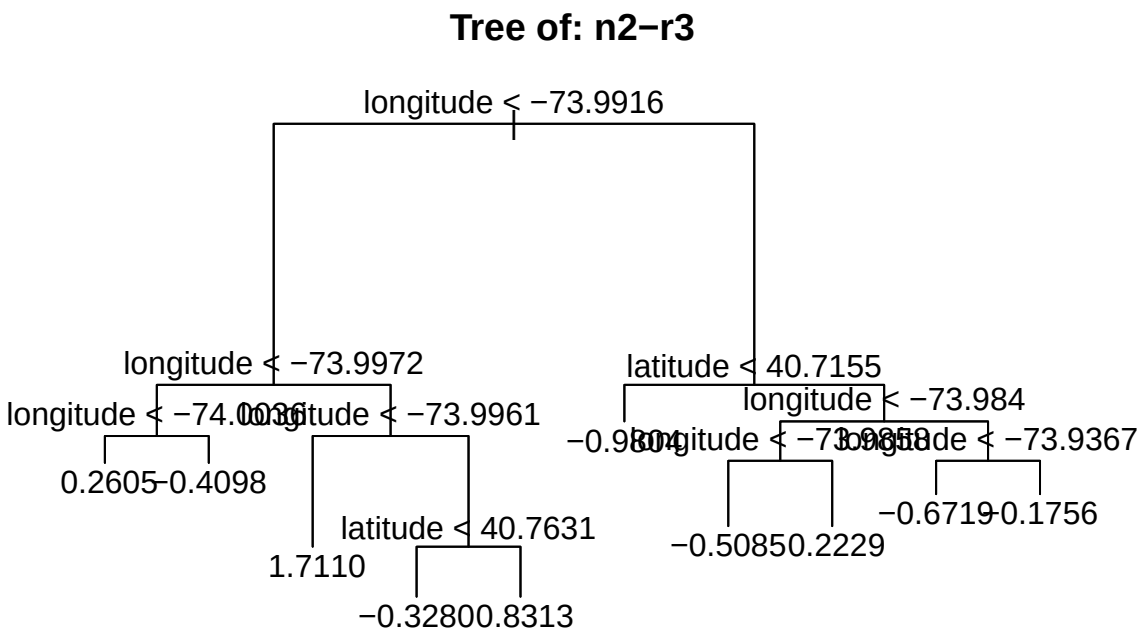

```
## [1] 0.2505339
##
##
## [1] "========== n2-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  9
## Residual mean deviance:  0.3895 = 2043 / 5246
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.5230 -0.3565 -0.1293  0.0000  0.1433  4.8760
```



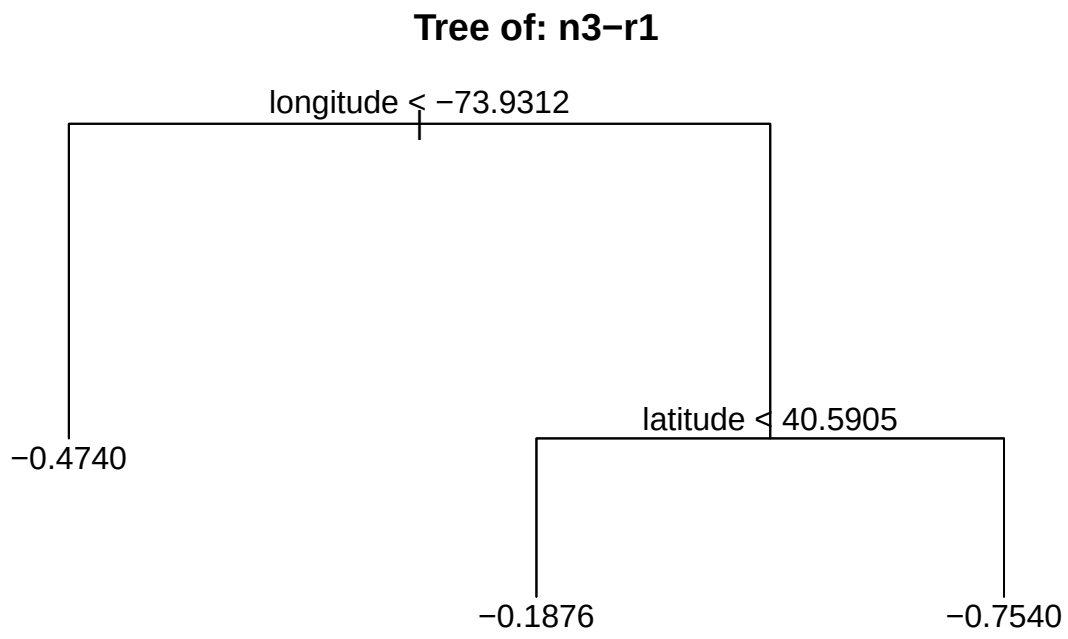

```
## [1] 0.3753113
##
##
## [1] "========== n2-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "longitude"
## Number of terminal nodes:  2
## Residual mean deviance:  1.01 = 8431 / 8344
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.3060 -0.7175 -0.2063  0.0000  0.4071  3.8260
```



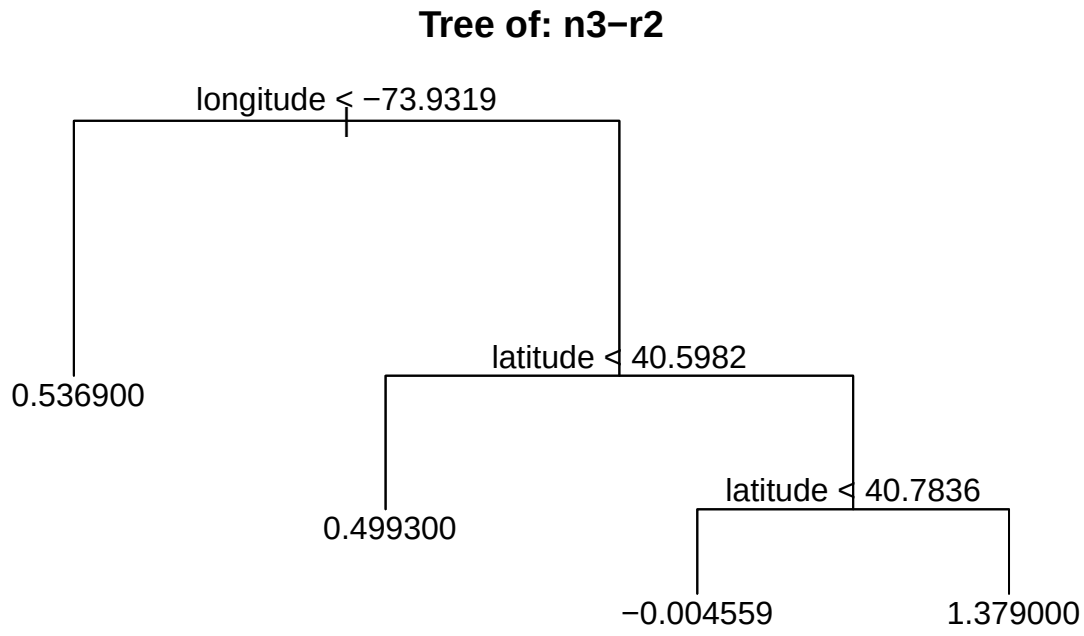

```
## [1] 0.9585893
##
##
## [1] "========== n2-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  10
## Residual mean deviance:  0.4952 = 151.5 / 306
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.6830 -0.2992 -0.1347  0.0000  0.0506  3.5570
```



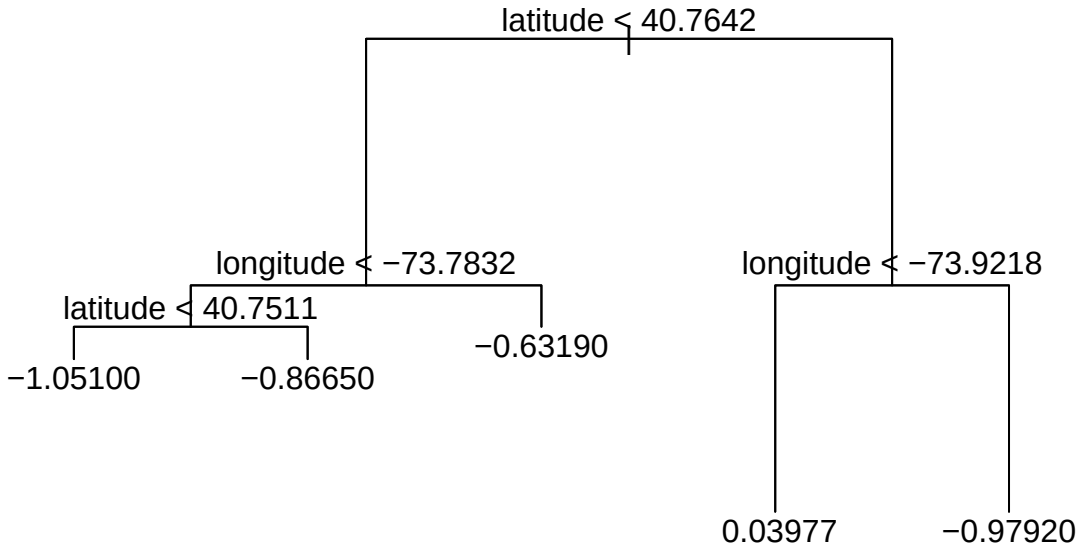

```
## [1] 0.4992158
##
##
## [1] "========== n3-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  3
## Residual mean deviance:  0.1771 = 396.5 / 2239
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.88730 -0.23000 -0.08237  0.00000  0.11070  4.65500
```



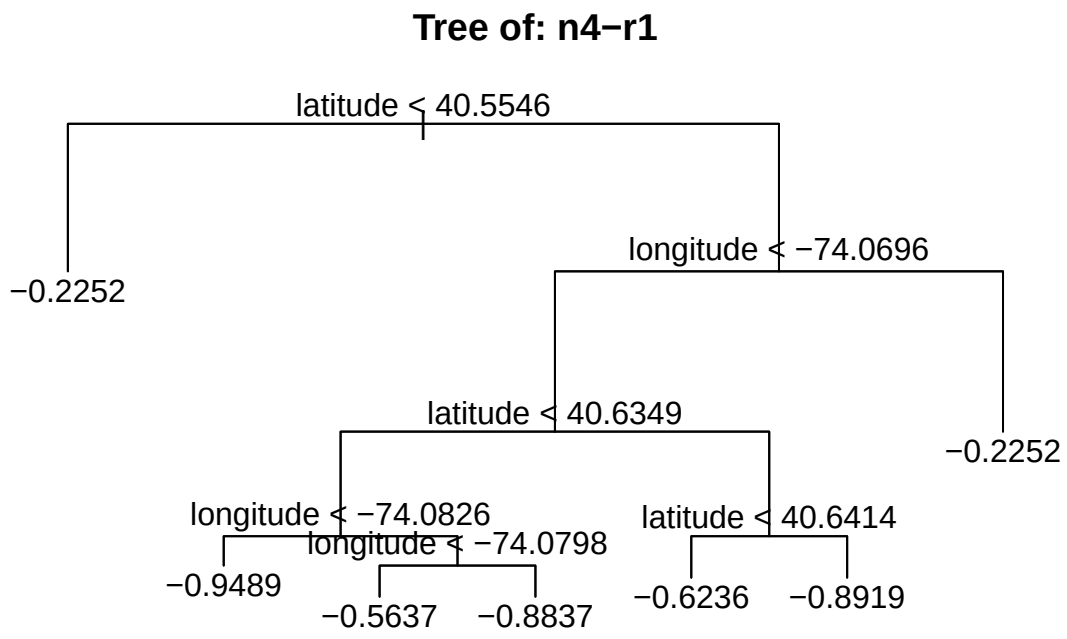

```
## [1] 0.1551588
##
##
## [1] "========== n3-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.6455 = 890.9 / 1380
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.1360 -0.4910 -0.2411  0.0000  0.2133  4.1890
```



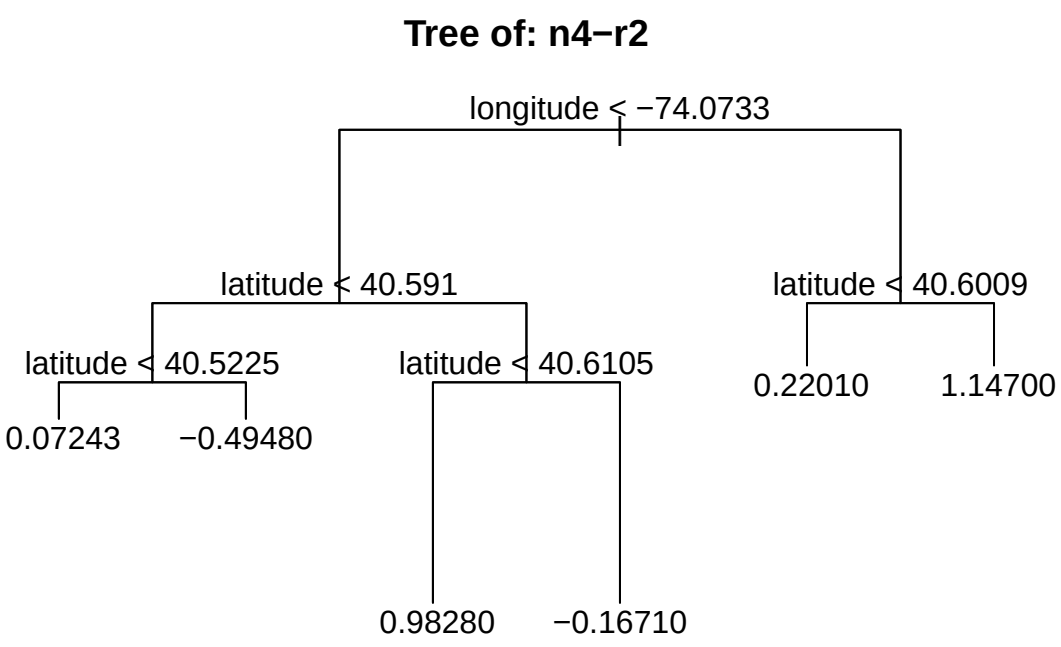

```
## [1] 0.6995247
##
##
## [1] "========== n3-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  5
## Residual mean deviance:  0.2795 = 34.65 / 124
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.17100 -0.13770 -0.07213  0.00000  0.06675  4.14500
```



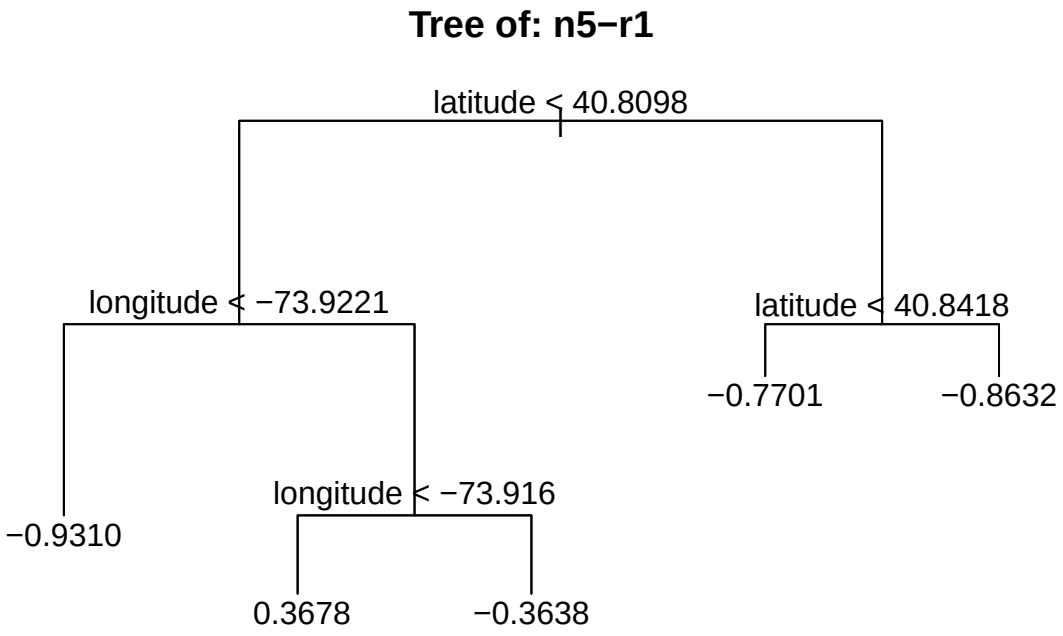

```
## [1] 0.1667691
##
##
## [1] "========== n4-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  7
## Residual mean deviance:  0.1462 = 17.4 / 119
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.81560 -0.18000 -0.03524  0.00000  0.09394  2.13800
```



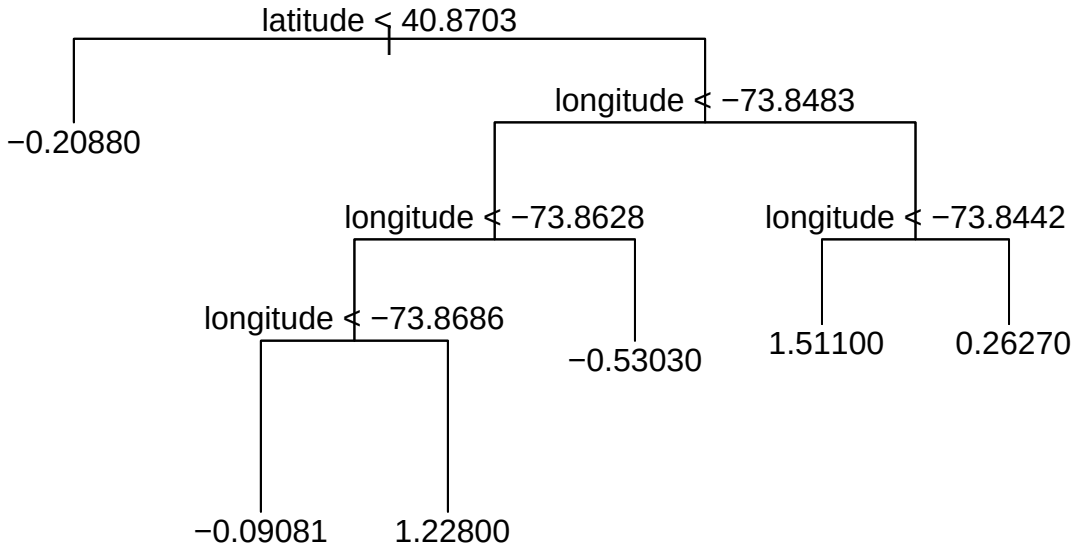

```
## [1] 0.1605553
##
##
## [1] "========== n4-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  6
## Residual mean deviance:  0.5117 = 54.76 / 107
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.7900 -0.3757 -0.1485  0.0000  0.3059  2.4700
```



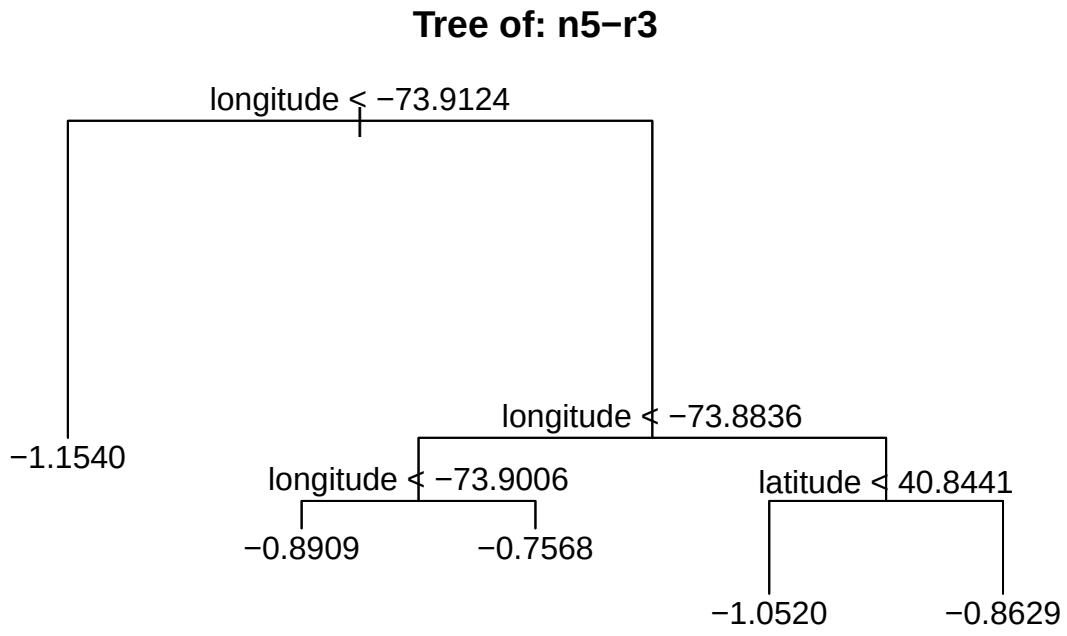

```
## [1] 0.7737648
##
##
## [1] "========== n4-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## character(0)
## Number of terminal nodes:  1
## Residual mean deviance:  0.09365 = 0.3746 / 4
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2931 -0.1908 -0.1795  0.0000  0.3317  0.3317
## Not possible to plot tree:  n4-r3[1] 0.5818964
##
##
## [1] "========== n5-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  5
## Residual mean deviance:  0.163 = 69.6 / 427
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.35200 -0.21400 -0.06405  0.00000  0.12680  3.81700
```



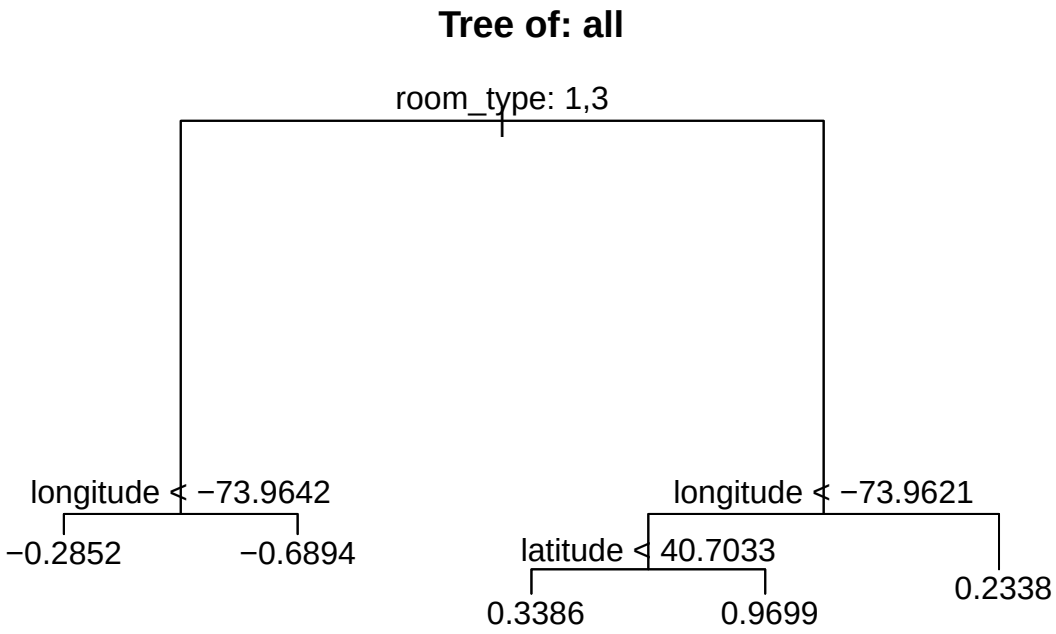

```
## [1] 0.1611146
##
##
## [1] "========== n5-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  6
## Residual mean deviance:  0.5703 = 139.7 / 245
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0970 -0.4345 -0.1505  0.0000  0.1693  3.8250
```




```
## [1] 0.886004
##
##
## [1] "========== n5-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  5
## Residual mean deviance:  0.07718 = 2.624 / 34
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.40520 -0.17040 -0.02272  0.00000  0.04430  1.07200
```




```
## [1] 0.1939584
##
##
## [1] "========== all =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type" "longitude" "latitude"
## Number of terminal nodes:  5
## Residual mean deviance:  0.5997 = 17200 / 28680
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.2830 -0.4196 -0.1356  0.0000  0.2166  4.8740
```




```
## [1] 0.6149423
```

```
dec_tree$name = "Decision Tree"
model_lis$decision_tree = dec_tree
```

RANDOM FOREST

```
rf = vector("list")

for (sub in names(trains))
{
  rf[[sub]]$fit = res =  randomForest(  price ~ . , data=trains[[sub]])

  rf[[sub]]$pred = predt = predict(res,tests[[sub]])

  print(paste0("========== ",sub, " =========="))

  print(res)

  rf[[sub]]$MSE = mse = sum((predt - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  print(paste0("MSE: ",mse))
  cat("\n\n")
}
```

```
## [1] "========== 1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.4328756
##                     % Var explained: 41.16
## [1] "MSE: 0.437649170189599"
##
##
## [1] "========== 2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.7406769
##                     % Var explained: 36.84
## [1] "MSE: 0.715054570578425"
##
##
## [1] "========== 3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.3479243
##                     % Var explained: 33.31
## [1] "MSE: 0.36011180740198"
##
##
## [1] "========== 4 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.4166214
##                     % Var explained: 21.91
## [1] "MSE: 0.3433678508242"
##
##
## [1] "========== 5 =========="
##
```

```
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.3912467
##                     % Var explained: 21.77
## [1] "MSE: 0.271493410039648"
##
##
## [1] "========== n1-r1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1855678
##                     % Var explained: 1.89
## [1] "MSE: 0.18377381564587"
##
##
## [1] "========== n1-r2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.7815159
##                     % Var explained: 2.05
## [1] "MSE: 0.801923378061946"
##
##
## [1] "========== n1-r3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1929481
##                     % Var explained: 4.76
## [1] "MSE: 0.240669947869671"
##
##
## [1] "========== n2-r1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
```

```
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.400548
##                     % Var explained: 23.35
## [1] "MSE: 0.37844968129523"
##
##
## [1] "========== n2-r2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 1.034235
##                     % Var explained: 4.44
## [1] "MSE: 0.977142205038175"
##
##
## [1] "========== n2-r3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.7127527
##                     % Var explained: -8.63
## [1] "MSE: 0.563504335056196"
##
##
## [1] "========== n3-r1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1810658
##                     % Var explained: 1.86
## [1] "MSE: 0.151722317885811"
##
##
## [1] "========== n3-r2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.6678137
##                     % Var explained: 2.11
## [1] "MSE: 0.713578829161869"
##
##
## [1] "========== n3-r3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.4066588
##                     % Var explained: -19.19
## [1] "MSE: 0.222952585446267"
##
##
## [1] "========== n4-r1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1895889
##                     % Var explained: -2.23
## [1] "MSE: 0.14235780616274"
##
##
## [1] "========== n4-r2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.7799421
##                     % Var explained: -15.53
## [1] "MSE: 0.71895373565863"
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
## [1] "========== n4-r3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
```

```
##
##           Mean of squared residuals: 0.0182975
##                     % Var explained: 75.58
## [1] "MSE: 0.303227871154577"
##
##
## [1] "========== n5-r1 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.2109273
##                     % Var explained: -15.99
## [1] "MSE: 0.133084811823872"
##
##
## [1] "========== n5-r2 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.726527
##                     % Var explained: -6.48
## [1] "MSE: 0.870094681564989"
##
##
## [1] "========== n5-r3 =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1083252
##                     % Var explained: -22.7
## [1] "MSE: 0.176332867102828"
##
##
## [1] "========== all =========="
##
## Call:
##  randomForest(formula = price ~ ., data = trains[[sub]])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.5704246
```

```
##                     % Var explained: 42.45
## [1] "MSE: 0.58523248835968"
```

```
rf$name = "Random Forest"
model_lis$random_forest = rf
```

RANGER RANDOM FOREST

```
ranger_rf = vector("list")

for (sub in names(trains))
{

   print(paste0("========== ",sub, " =========="))

  ranger_rf[[sub]]$fit = res = ranger( price~ ., data = trains[[sub]], write.forest = TRUE, classificati

  ranger_rf[[sub]]$pred = predt = predict(res,tests[[sub]])
  ranger_rf[[sub]]$MSE = mse =  sum((predt$predictions - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  print(res)
  print(paste0("MSE: ",mse))
  cat("\n\n")
}
```

```
## [1] "========== 1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      14893
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.4300719
## R squared (OOB):                  0.4154151
## [1] "MSE: 0.435661195179693"
##
##
## [1] "========== 2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      15658
```

```
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7402947
## R squared (OOB):                  0.368762
## [1] "MSE: 0.713232260548964"
##
##
## [1] "========== 3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      4224
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.3469485
## R squared (OOB):                  0.3351217
## [1] "MSE: 0.359439894738383"
##
##
## [1] "========== 4 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      274
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.4235784
## R squared (OOB):                  0.208931
## [1] "MSE: 0.349825116770277"
##
##
## [1] "========== 5 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
```

```
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      811
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.3985611
## R squared (OOB):                  0.2040182
## [1] "MSE: 0.268062661260826"
##
##
## [1] "========== n1-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      6724
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1858047
## R squared (OOB):                  0.01779825
## [1] "MSE: 0.183050763664203"
##
##
## [1] "========== n1-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      6241
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7820273
## R squared (OOB):                  0.02001014
## [1] "MSE: 0.805646710986738"
##
##
## [1] "========== n1-r3 =========="
## Ranger result
##
```

```
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      273
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1935494
## R squared (OOB):                  0.04816971
## [1] "MSE: 0.238626751943499"
##
##
## [1] "========== n2-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      5255
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.3999019
## R squared (OOB):                  0.2348782
## [1] "MSE: 0.378283997644987"
##
##
## [1] "========== n2-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      8346
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       1.036799
## R squared (OOB):                  0.04214508
## [1] "MSE: 0.974357995618573"
##
##
```

```
## [1] "========== n2-r3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      316
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7160939
## R squared (OOB):                  -0.08798569
## [1] "MSE: 0.57155262780546"
##
##
## [1] "========== n3-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      2242
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.181763
## R squared (OOB):                  0.01525706
## [1] "MSE: 0.152704198115273"
##
##
## [1] "========== n3-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      1384
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.6704211
## R squared (OOB):                  0.01803546
```

```
## [1] "MSE: 0.71238608718685"
##
##
## [1] "========== n3-r3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      129
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.4162768
## R squared (OOB):                  -0.2106128
## [1] "MSE: 0.22139444129044"
##
##
## [1] "========== n4-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      126
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1911728
## R squared (OOB):                  -0.02263984
## [1] "MSE: 0.143390186091069"
##
##
## [1] "========== n4-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      113
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
```

```
## Splitrule:                          variance
## OOB prediction error (MSE):       0.7852717
## R squared (OOB):                  -0.1528912
## [1] "MSE: 0.705639717093316"
##
##
## [1] "========== n4-r3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      5
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1158974
## R squared (OOB):                  -0.2376012
## [1] "MSE: 0.584749620286946"
##
##
## [1] "========== n5-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      432
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.2150884
## R squared (OOB):                  -0.1800232
## [1] "MSE: 0.132753624265668"
##
##
## [1] "========== n5-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      251
## Number of independent variables:  2
```

```
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7310383
## R squared (OOB):                  -0.06716958
## [1] "MSE: 0.870872319133183"
##
##
## [1] "========== n5-r3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      39
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1122889
## R squared (OOB):                  -0.2393209
## [1] "MSE: 0.17684139550054"
##
##
## [1] "========== all =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      28689
## Number of independent variables:  4
## Mtry:                             2
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.5431618
## R squared (OOB):                  0.4520079
## [1] "MSE: 0.553992505569662"
```

```
ranger_rf$name = "Ranger Random Forest"
model_lis$ranger = ranger_rf
```

```
build_model <- function(dimension) {

  model <- keras::keras_model_sequential() %>%
    layer_dense(units = 32, activation = "relu",
                input_shape = dimension) %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 1,activation="linear")

  model %>% compile(
    loss = "mse",
    optimizer = optimizer_rmsprop(),
    metrics = list("mean_absolute_error")
  )

  return(model)
}


print_dot_callback <- callback_lambda(
  on_epoch_end = function(epoch, logs) {
    if (epoch %% 1 == 0)
      cat(".")
  }
)


nn = vector("list")

for (sub in names(trains))
{
  d = trains[[sub]]
  d2 = tests[[sub]]
  len = length(d)
  len2 = length(d2)

  if(!is.null(d$room_type))
  {
    d$room_type = keras::to_categorical(d$room_type)
    d2$room_type = keras::to_categorical(d2$room_type)

  }

  if(!is.null(d$neighbourhood_group))
  {
    d$neighbourhood_group = keras::to_categorical(d$neighbourhood_group)
    d2$neighbourhood_group = keras::to_categorical(d2$neighbourhood_group)
  }


  target = as.vector(d$price)
  features = as.matrix(as_tibble(d[-len]))
```

```
  target_test = as.vector(d2$price)
  features_test = as.matrix(as_tibble(d2[-len]))


  nn[[sub]]$epochs = epochs =  30

  nn[[sub]]$model = model =  build_model(dim(features)[2])

  nn[[sub]]$summary = model %>% summary()
  nn[[sub]]$history = hist =  model %>% fit(
    x = features,
    y = target,
    epochs = epochs,
    validation_split = 0.2,
    verbose = 0,
    callbacks = list(print_dot_callback)
  )
  eva = model %>% evaluate(features_test,target_test, verbose = 0)


  nn[[sub]]$mae = eva[1]
  nn[[sub]]$loss = eva[2]

  nn[[sub]]$pred = pred = model %>% predict(features_test)
  nn[[sub]]$MSE = sum((pred - target_test)^2)/length(target_test)

}
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 32)                224
## _________________________________________________________________________
## dense_1 (Dense)              (None, 16)                528
## _________________________________________________________________________
## dense_2 (Dense)              (None, 1)                 17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_1"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_3 (Dense)              (None, 32)                224
## _________________________________________________________________________
## dense_4 (Dense)              (None, 16)                528
## _________________________________________________________________________
## dense_5 (Dense)              (None, 1)                 17
## =========================================================================
## Total params: 769
## Trainable params: 769
```

```
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_2"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_6 (Dense)              (None, 32)                224
## _________________________________________________________________
## dense_7 (Dense)              (None, 16)                528
## _________________________________________________________________
## dense_8 (Dense)              (None, 1)                 17
## =================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_3"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_9 (Dense)              (None, 32)                224
## _________________________________________________________________
## dense_10 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_11 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_4"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_12 (Dense)             (None, 32)                224
## _________________________________________________________________
## dense_13 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_14 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_5"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_15 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_16 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_17 (Dense)             (None, 1)                 17
## =================================================================
```

```
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_6"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_18 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_19 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_20 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_7"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_21 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_22 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_23 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_8"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_24 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_25 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_26 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_9"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_27 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_28 (Dense)             (None, 16)                528
## _________________________________________________________________
```

```
## dense_29 (Dense)                    (None, 1)                         17
## =================================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________________
## ............................Model: "sequential_10"
## _________________________________________________________________________________
## Layer (type)                        Output Shape                      Param #
## =================================================================================
## dense_30 (Dense)                    (None, 32)                        96
## _________________________________________________________________________________
## dense_31 (Dense)                    (None, 16)                        528
## _________________________________________________________________________________
## dense_32 (Dense)                    (None, 1)                         17
## =================================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________________
## ............................Model: "sequential_11"
## _________________________________________________________________________________
## Layer (type)                        Output Shape                      Param #
## =================================================================================
## dense_33 (Dense)                    (None, 32)                        96
## _________________________________________________________________________________
## dense_34 (Dense)                    (None, 16)                        528
## _________________________________________________________________________________
## dense_35 (Dense)                    (None, 1)                         17
## =================================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________________
## ............................Model: "sequential_12"
## _________________________________________________________________________________
## Layer (type)                        Output Shape                      Param #
## =================================================================================
## dense_36 (Dense)                    (None, 32)                        96
## _________________________________________________________________________________
## dense_37 (Dense)                    (None, 16)                        528
## _________________________________________________________________________________
## dense_38 (Dense)                    (None, 1)                         17
## =================================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________________
## ............................Model: "sequential_13"
## _________________________________________________________________________________
## Layer (type)                        Output Shape                      Param #
## =================================================================================
## dense_39 (Dense)                    (None, 32)                        96
## _________________________________________________________________________________
```

```
## dense_40 (Dense)                 (None, 16)                528
## _________________________________________________________________
## dense_41 (Dense)                 (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_14"
## _________________________________________________________________
## Layer (type)                     Output Shape              Param #
## =================================================================
## dense_42 (Dense)                 (None, 32)                96
## _________________________________________________________________
## dense_43 (Dense)                 (None, 16)                528
## _________________________________________________________________
## dense_44 (Dense)                 (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_15"
## _________________________________________________________________
## Layer (type)                     Output Shape              Param #
## =================================================================
## dense_45 (Dense)                 (None, 32)                96
## _________________________________________________________________
## dense_46 (Dense)                 (None, 16)                528
## _________________________________________________________________
## dense_47 (Dense)                 (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_16"
## _________________________________________________________________
## Layer (type)                     Output Shape              Param #
## =================================================================
## dense_48 (Dense)                 (None, 32)                96
## _________________________________________________________________
## dense_49 (Dense)                 (None, 16)                528
## _________________________________________________________________
## dense_50 (Dense)                 (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_17"
## _________________________________________________________________
## Layer (type)                     Output Shape              Param #
## =================================================================
```

```
## dense_51 (Dense)                    (None, 32)                    96
## _________________________________________________________________
## dense_52 (Dense)                    (None, 16)                    528
## _________________________________________________________________
## dense_53 (Dense)                    (None, 1)                     17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_18"
## _________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =================================================================
## dense_54 (Dense)                    (None, 32)                    96
## _________________________________________________________________
## dense_55 (Dense)                    (None, 16)                    528
## _________________________________________________________________
## dense_56 (Dense)                    (None, 1)                     17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_19"
## _________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =================================================================
## dense_57 (Dense)                    (None, 32)                    96
## _________________________________________________________________
## dense_58 (Dense)                    (None, 16)                    528
## _________________________________________________________________
## dense_59 (Dense)                    (None, 1)                     17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_20"
## _________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =================================================================
## dense_60 (Dense)                    (None, 32)                    416
## _________________________________________________________________
## dense_61 (Dense)                    (None, 16)                    528
## _________________________________________________________________
## dense_62 (Dense)                    (None, 1)                     17
## =================================================================
## Total params: 961
## Trainable params: 961
## Non-trainable params: 0
## _________________________________________________________________
## ............................
```

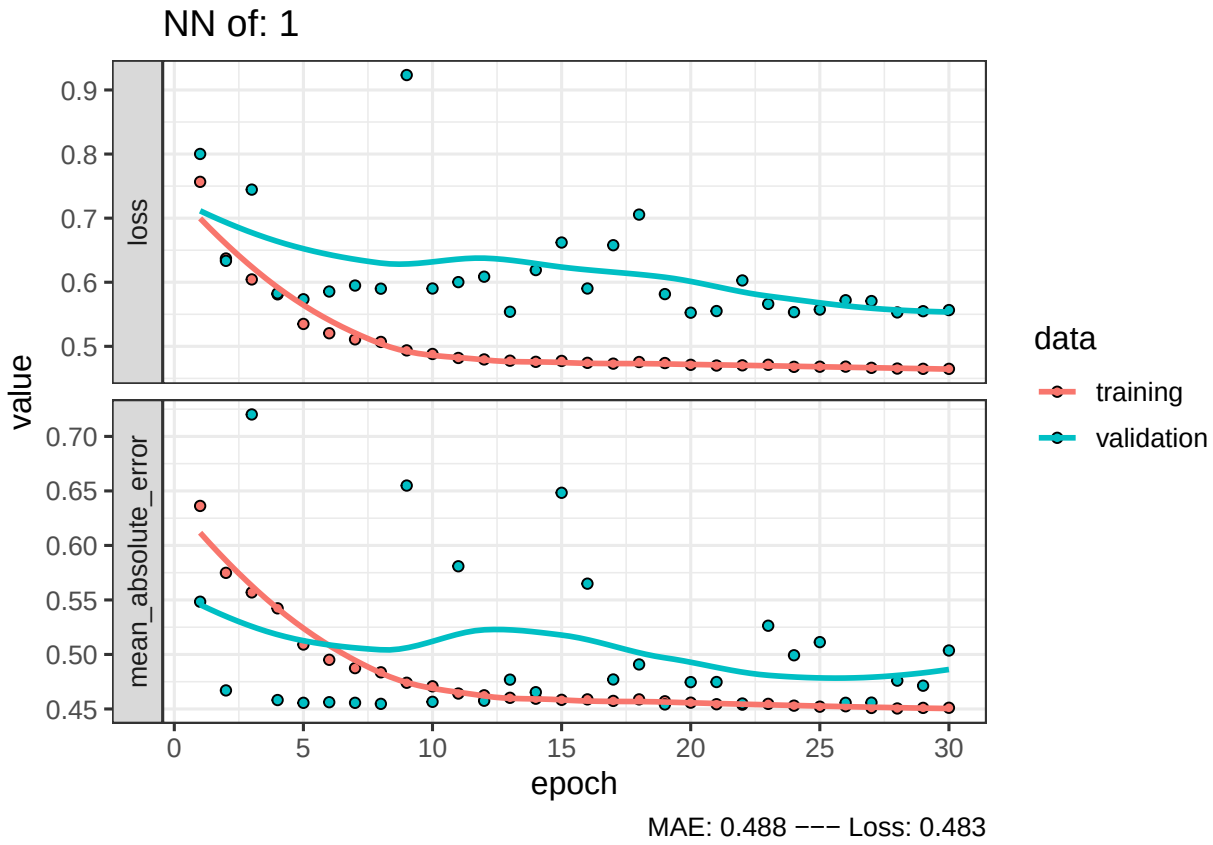
```
nn$name = "Neural Networks"
model_lis$neural_networks = nn
```

NN plots

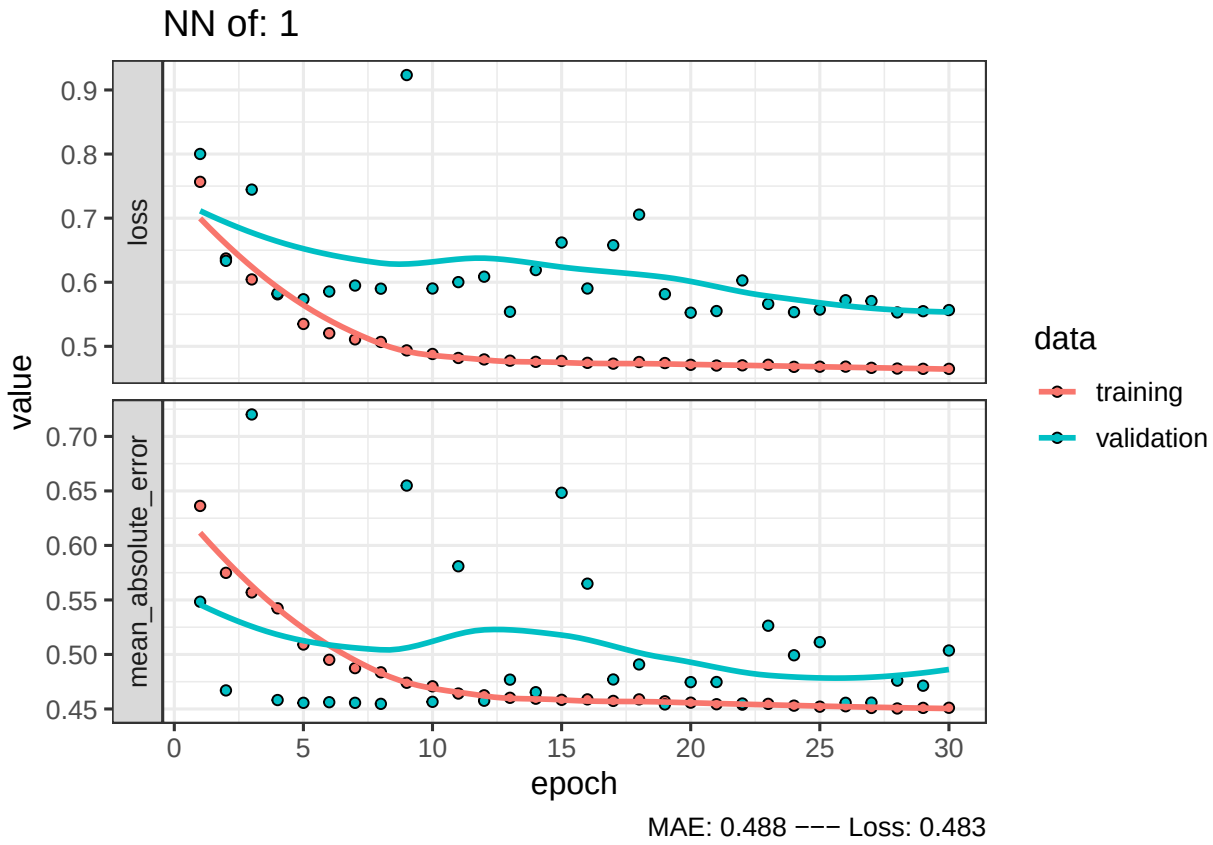
```
for (sub in names(nn))
{
  if(sub != "name")
  {
    m = nn[[sub]]
    hist = m$history
    print(paste0("========== ",sub, " =========="))
    str = paste0("MAE: ",round(m$mae,3)," --- Loss: ",round(m$loss,3))
    p = plot(hist, y ~ x) +  theme_bw(base_size = 12)  +ggtitle(paste0("NN of: ",sub))+ labs(caption= s

    print(p)
    plot(p)
    cat("MAE: ",m$mae)
    cat("\nLoss: ",m$loss,"\n\n")
  }

}
```

```
## [1] "========== 1 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

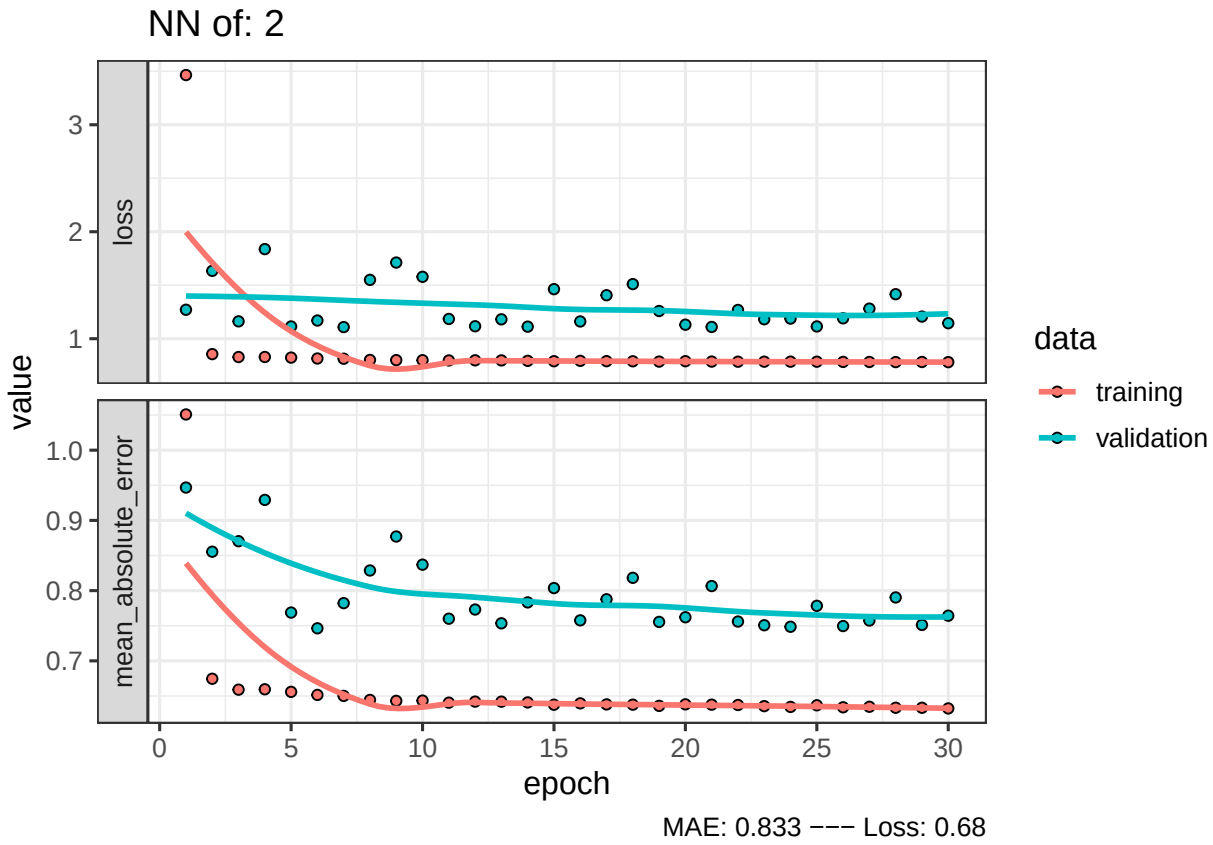



```
## `geom_smooth()` using formula 'y ~ x'
```

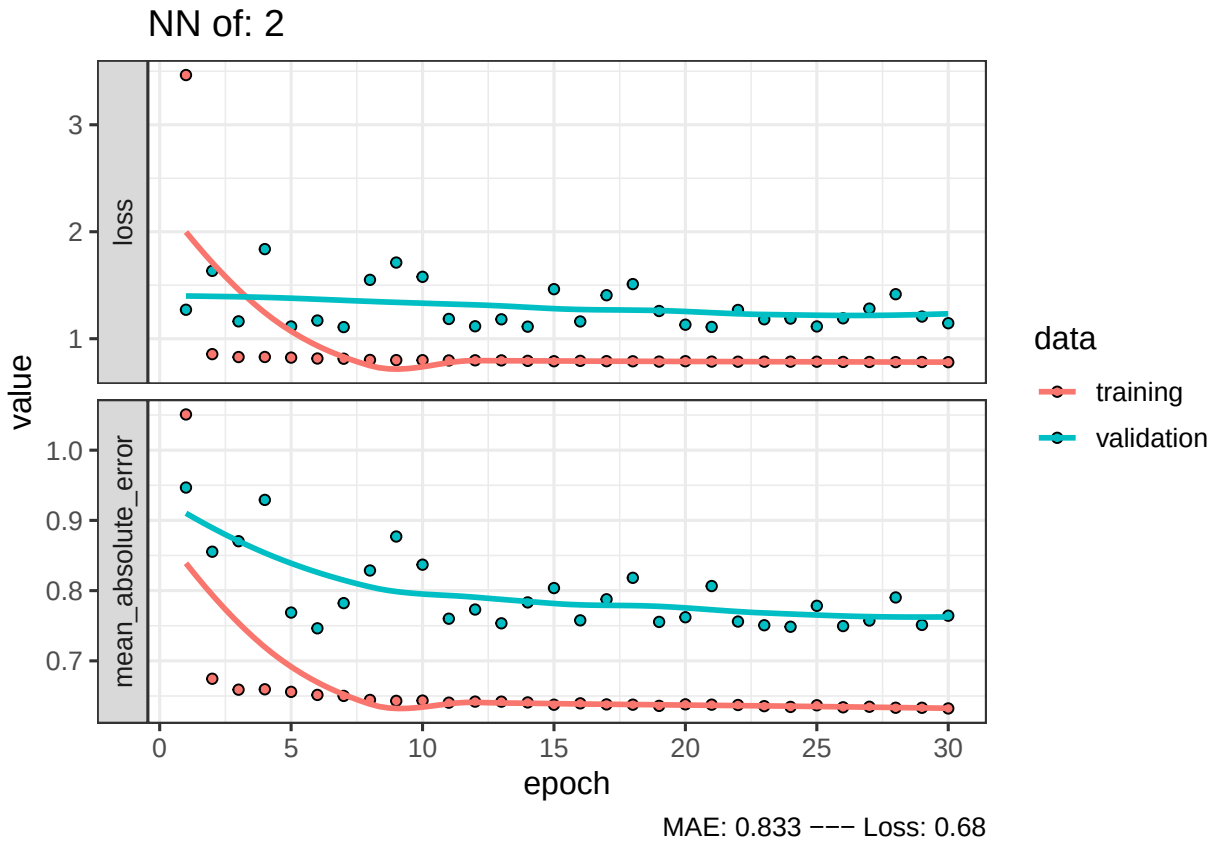



```
## MAE:  0.4877132
## Loss:  0.4832867
##
## [1] "========== 2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

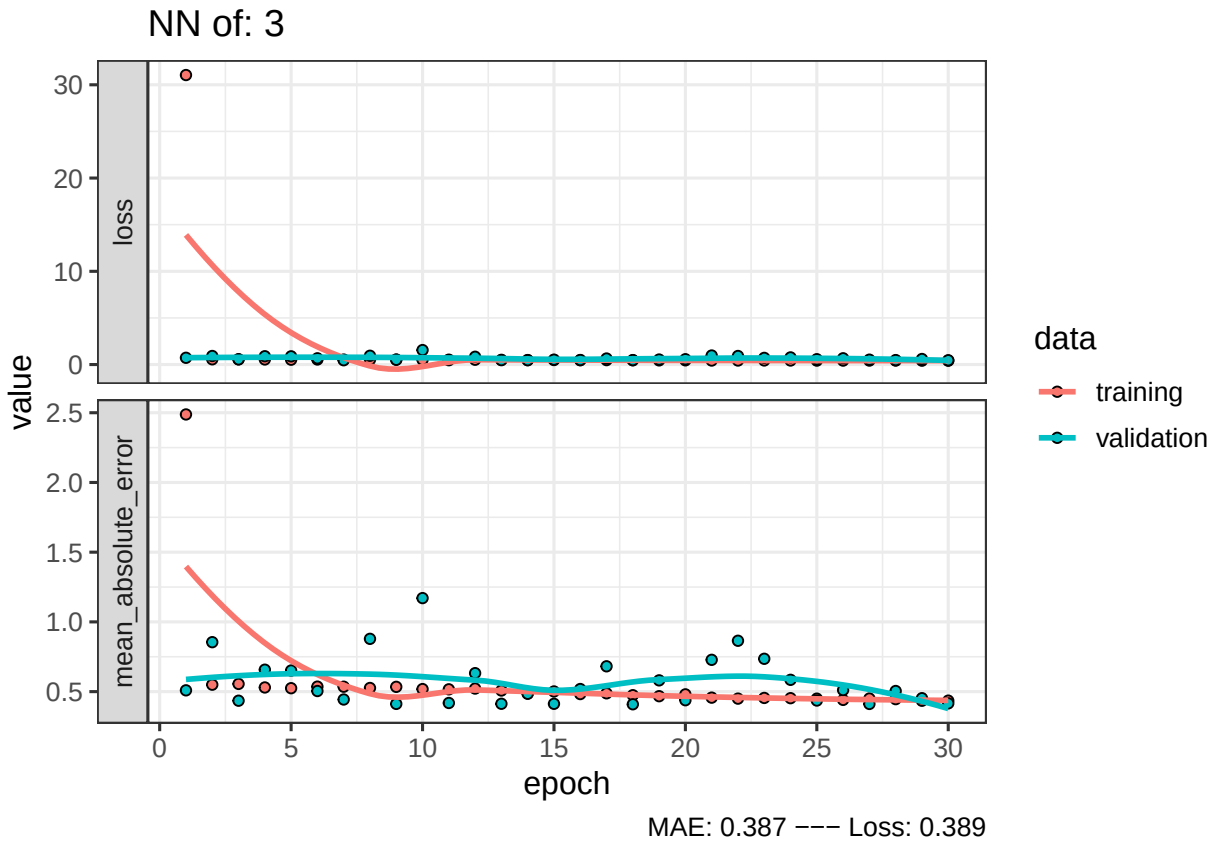



```
## `geom_smooth()` using formula 'y ~ x'
```

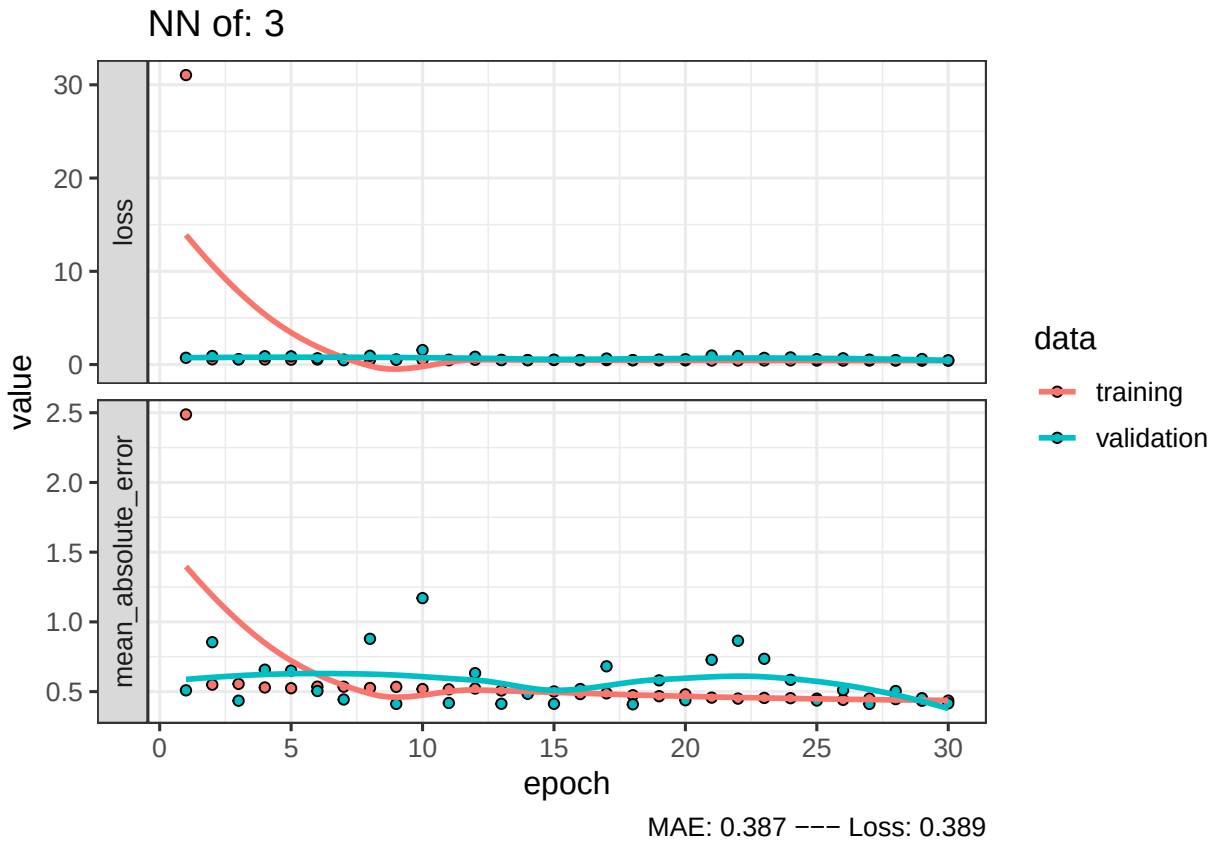



```
## MAE:  0.8333294
## Loss:  0.6800426
##
## [1] "========== 3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

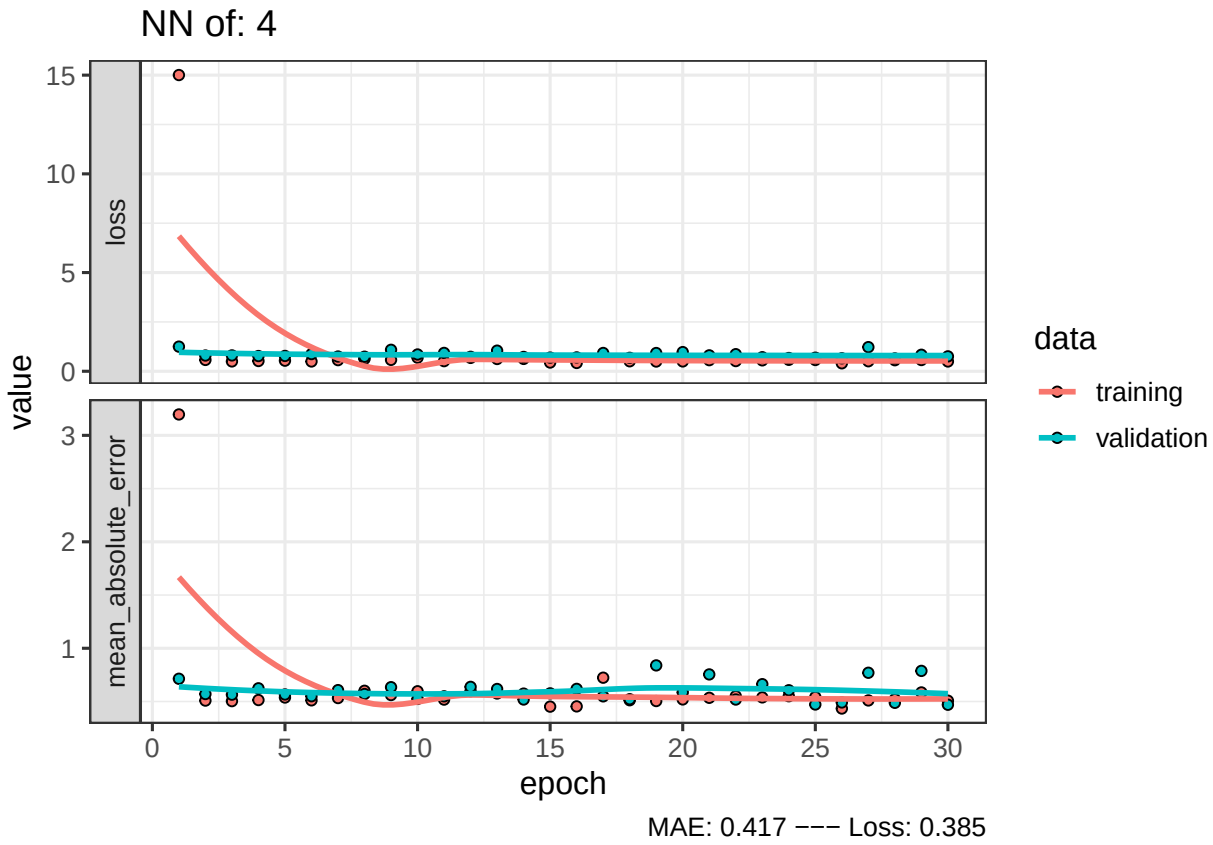



```
## `geom_smooth()` using formula 'y ~ x'
```

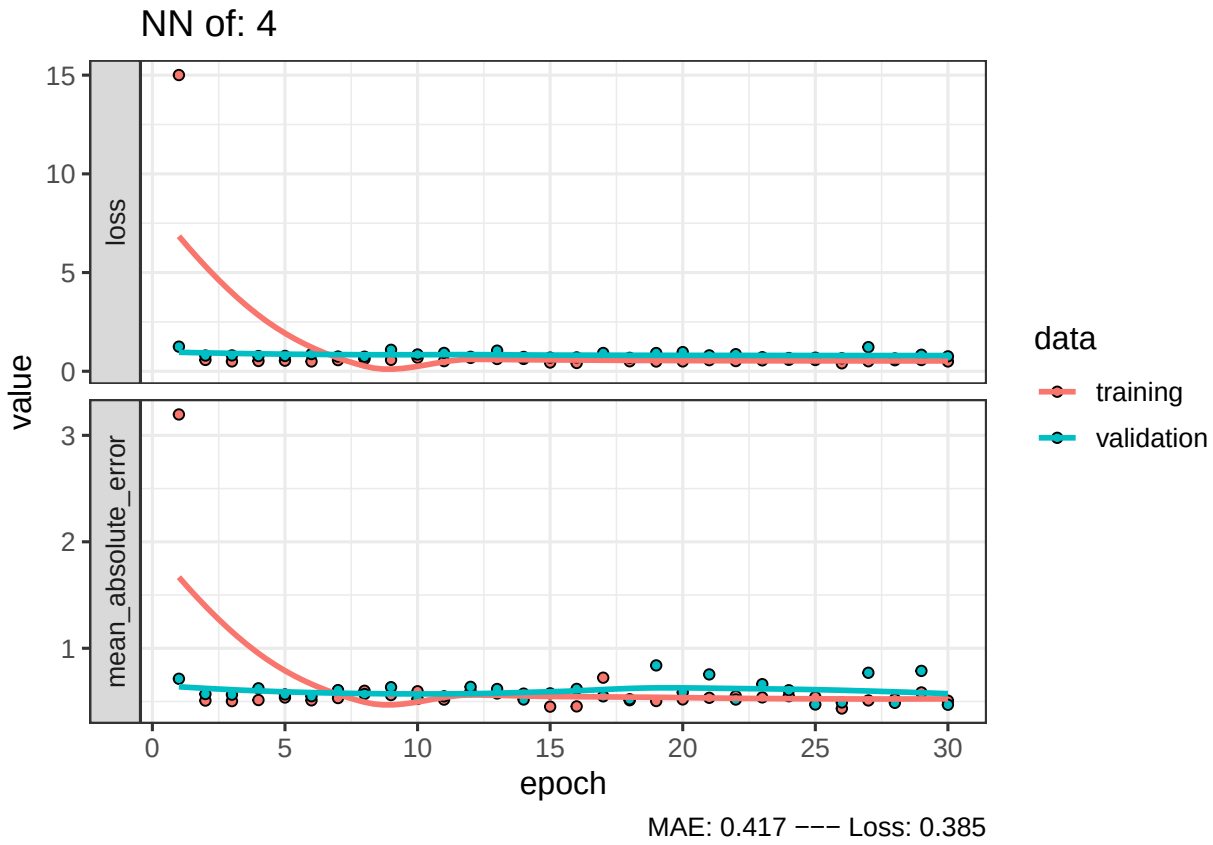



```
## MAE:  0.3867755
## Loss:  0.3894443
##
## [1] "========== 4 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

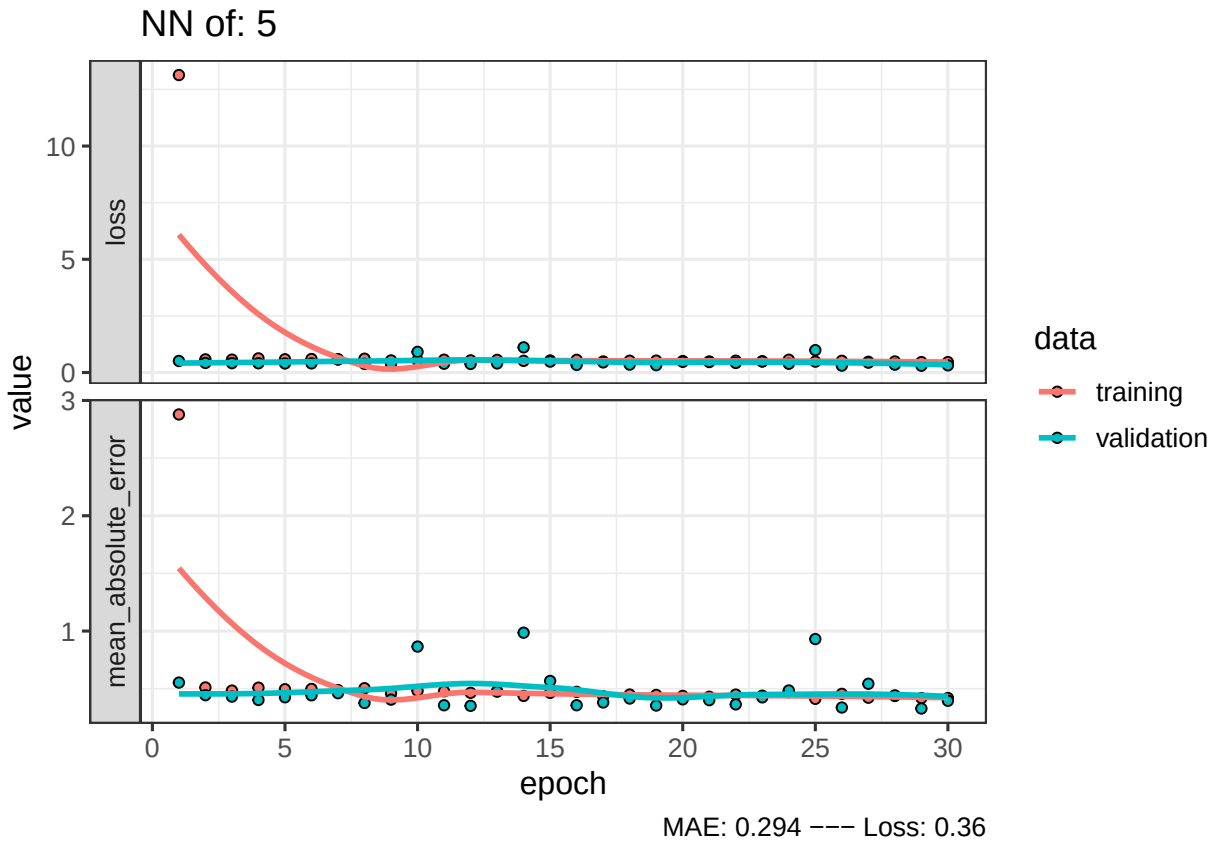



```
## `geom_smooth()` using formula 'y ~ x'
```

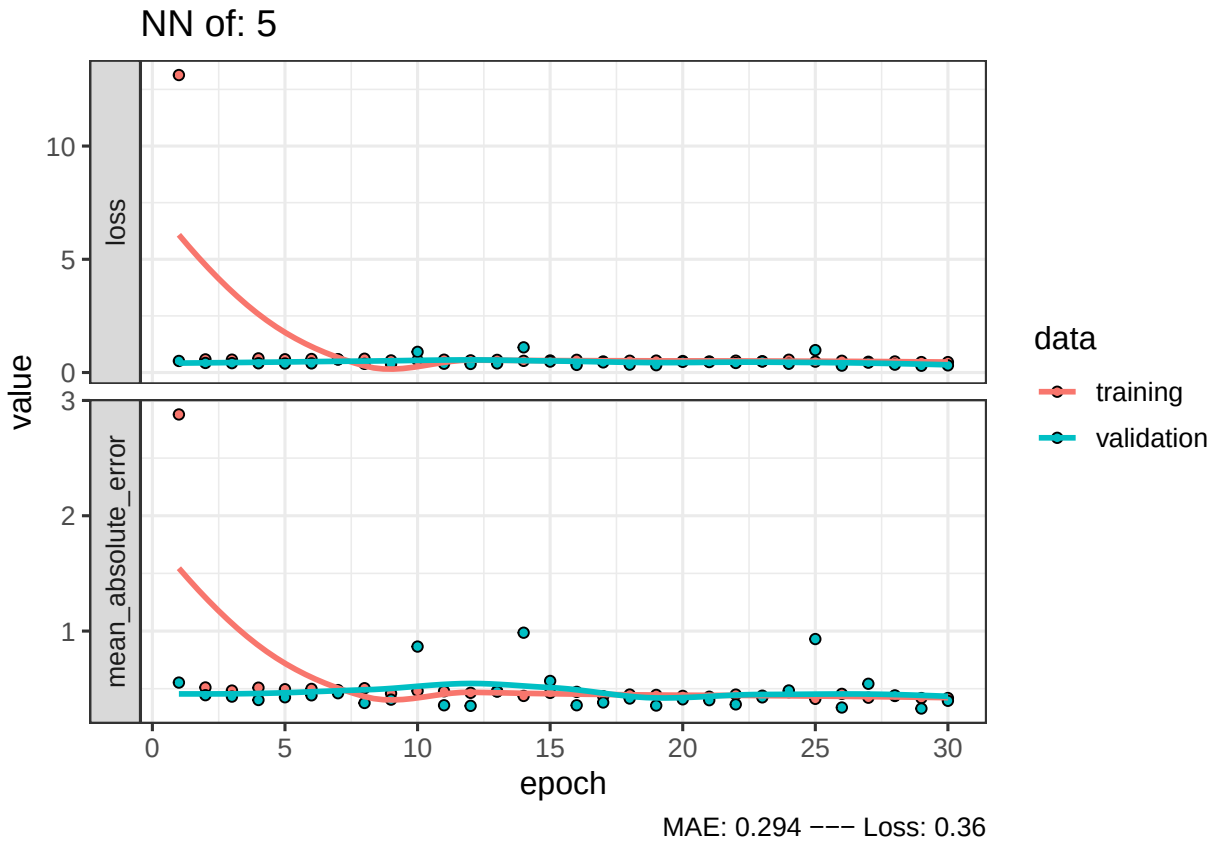



```
## MAE:  0.417328
## Loss:  0.3846916
##
## [1] "========== 5 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

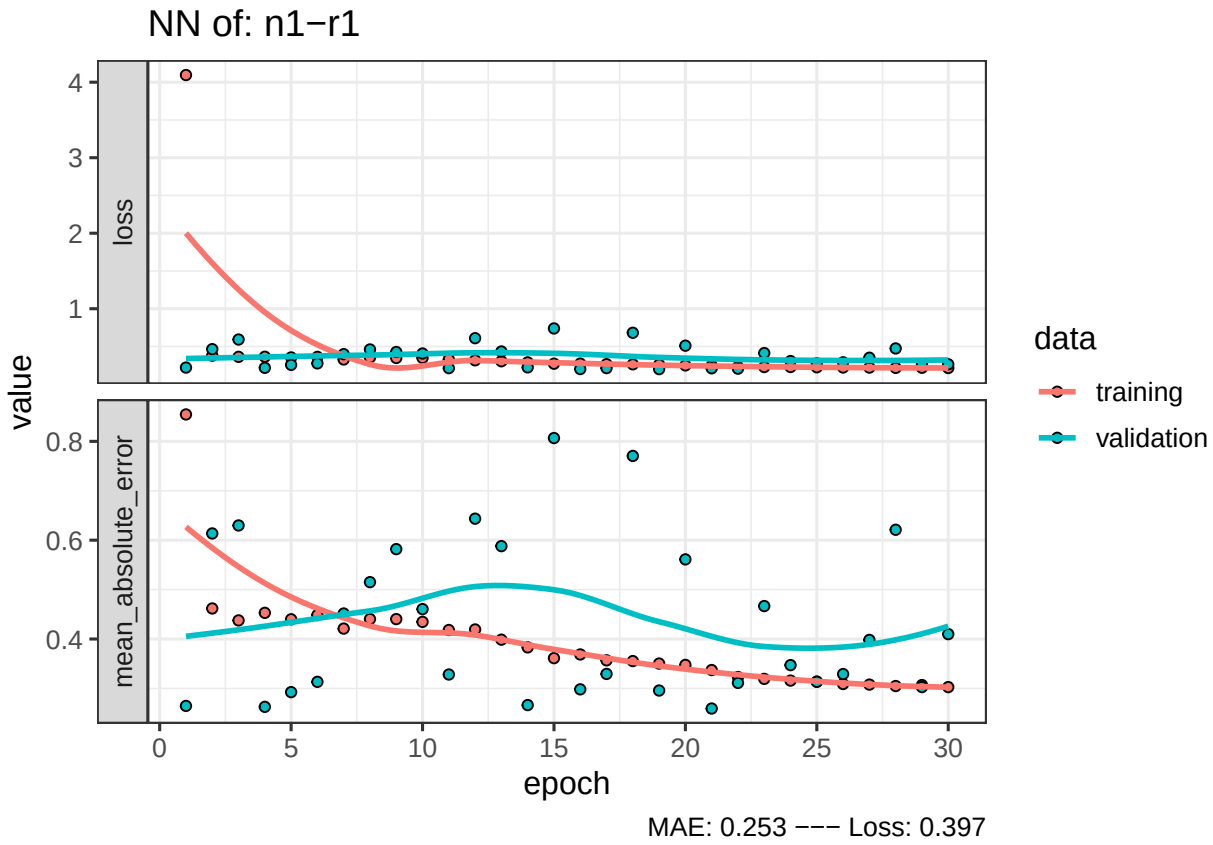



```
## `geom_smooth()` using formula 'y ~ x'
```

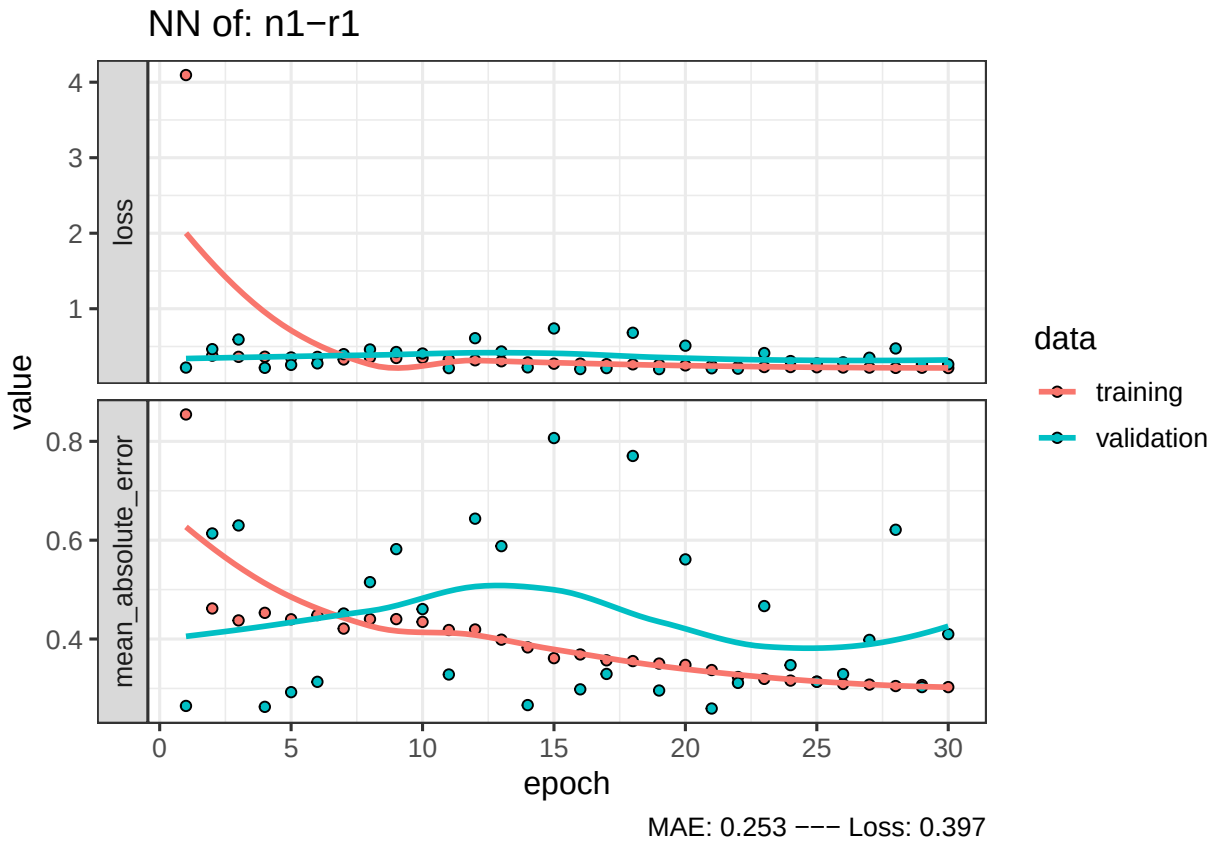



```
## MAE:  0.2938451
## Loss:  0.359958
##
## [1] "========== n1-r1 ==========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

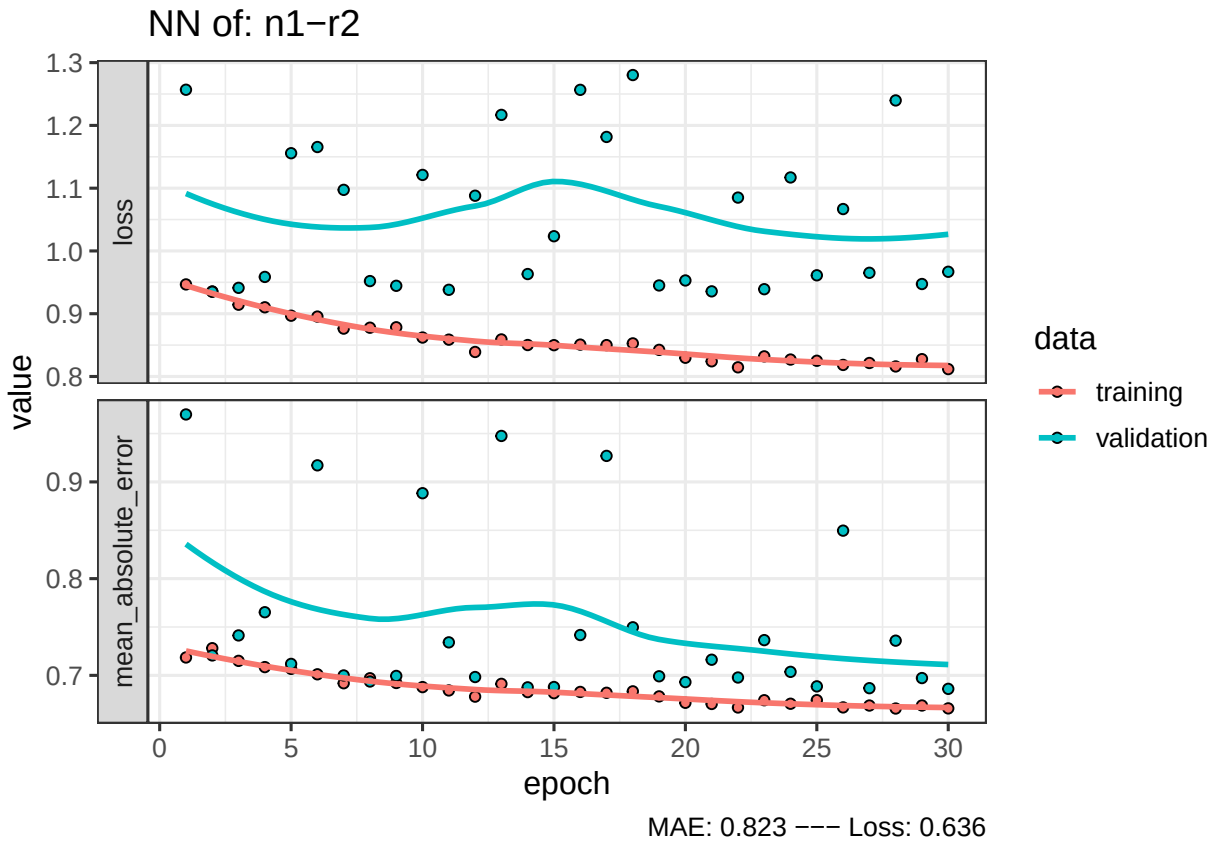



```
## `geom_smooth()` using formula 'y ~ x'
```

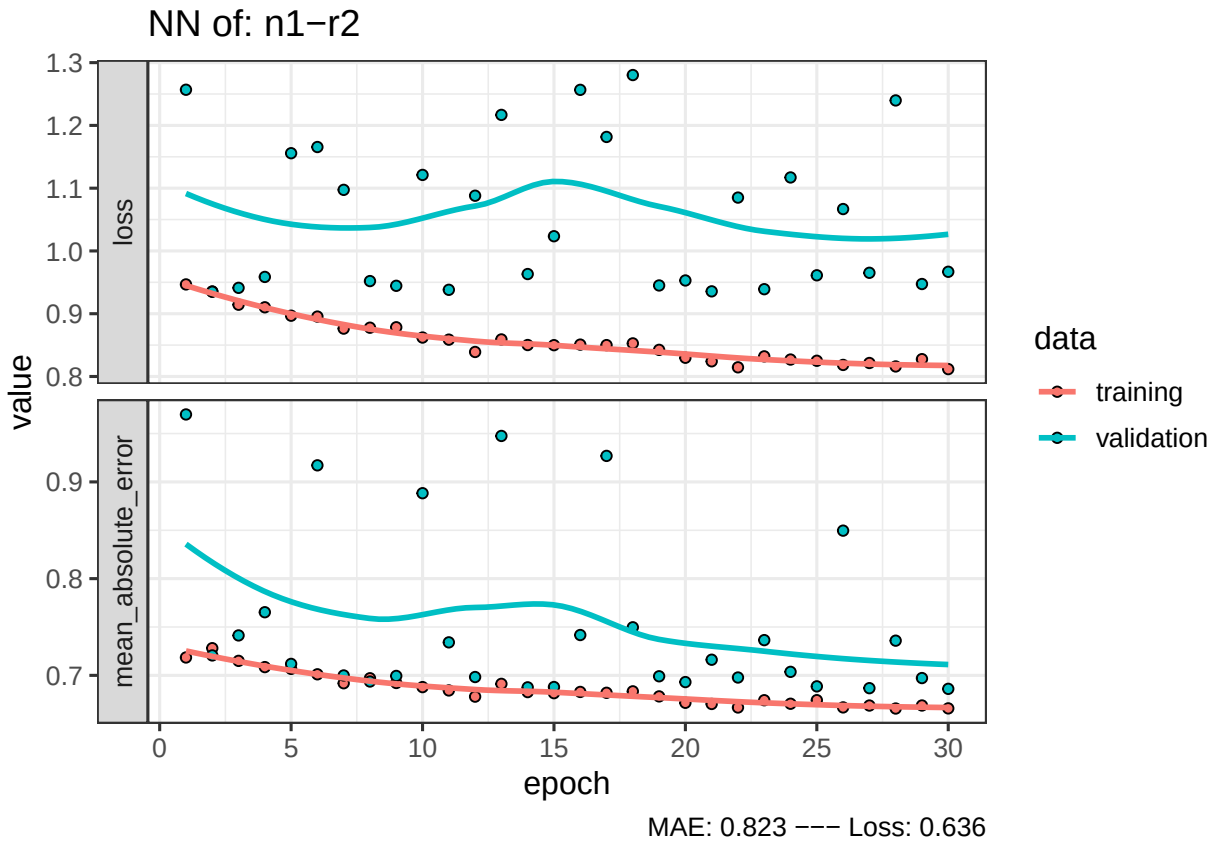



```
## MAE:  0.252656
## Loss:  0.3968593
##
## [1] "========== n1-r2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

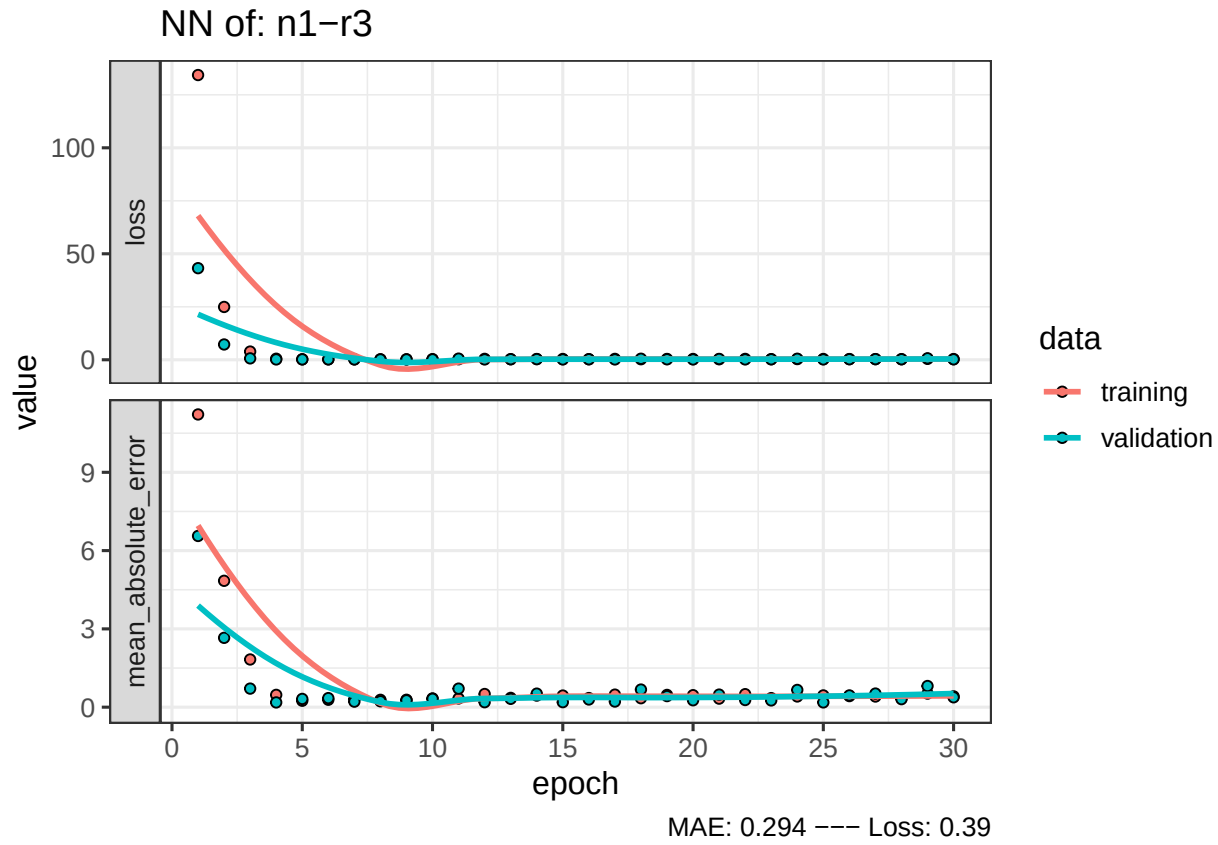



```
## `geom_smooth()` using formula 'y ~ x'
```

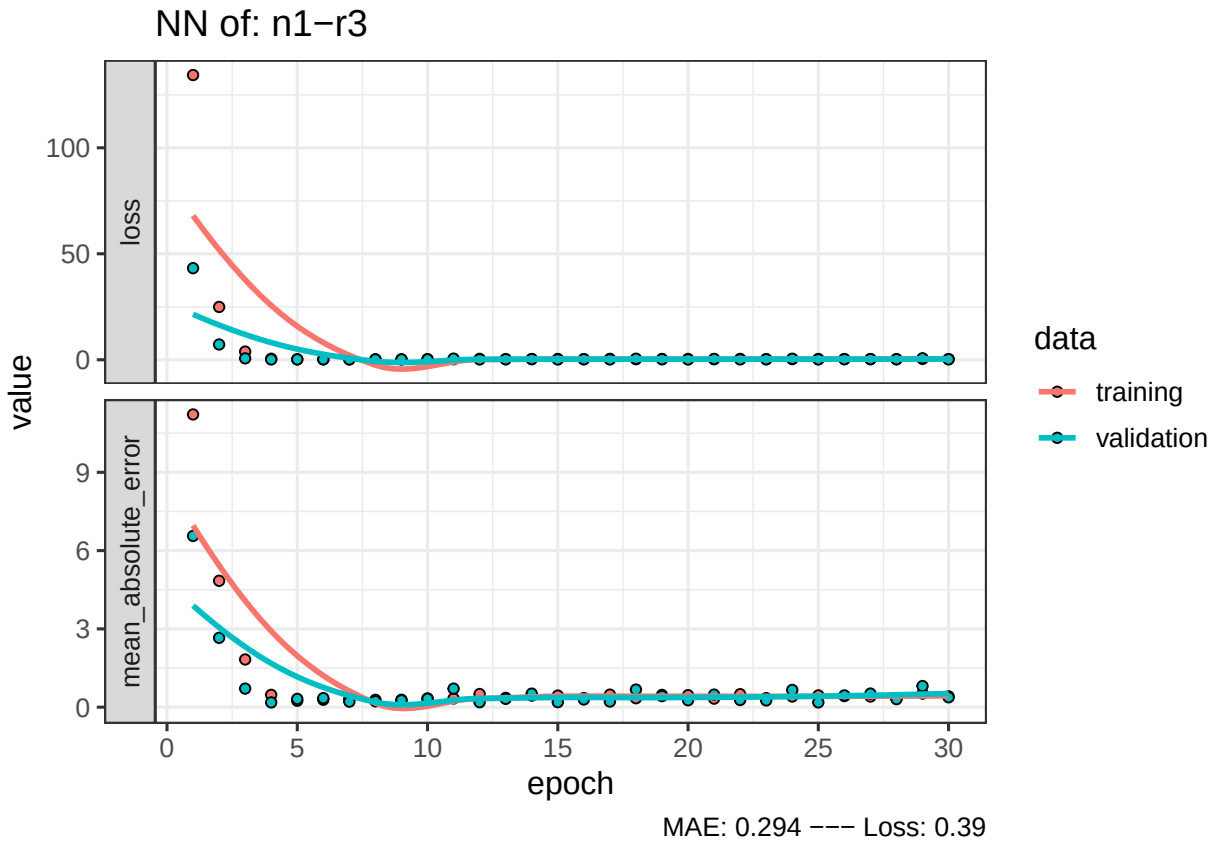



```
## MAE:  0.823441
## Loss:  0.6362852
##
## [1] "========== n1-r3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

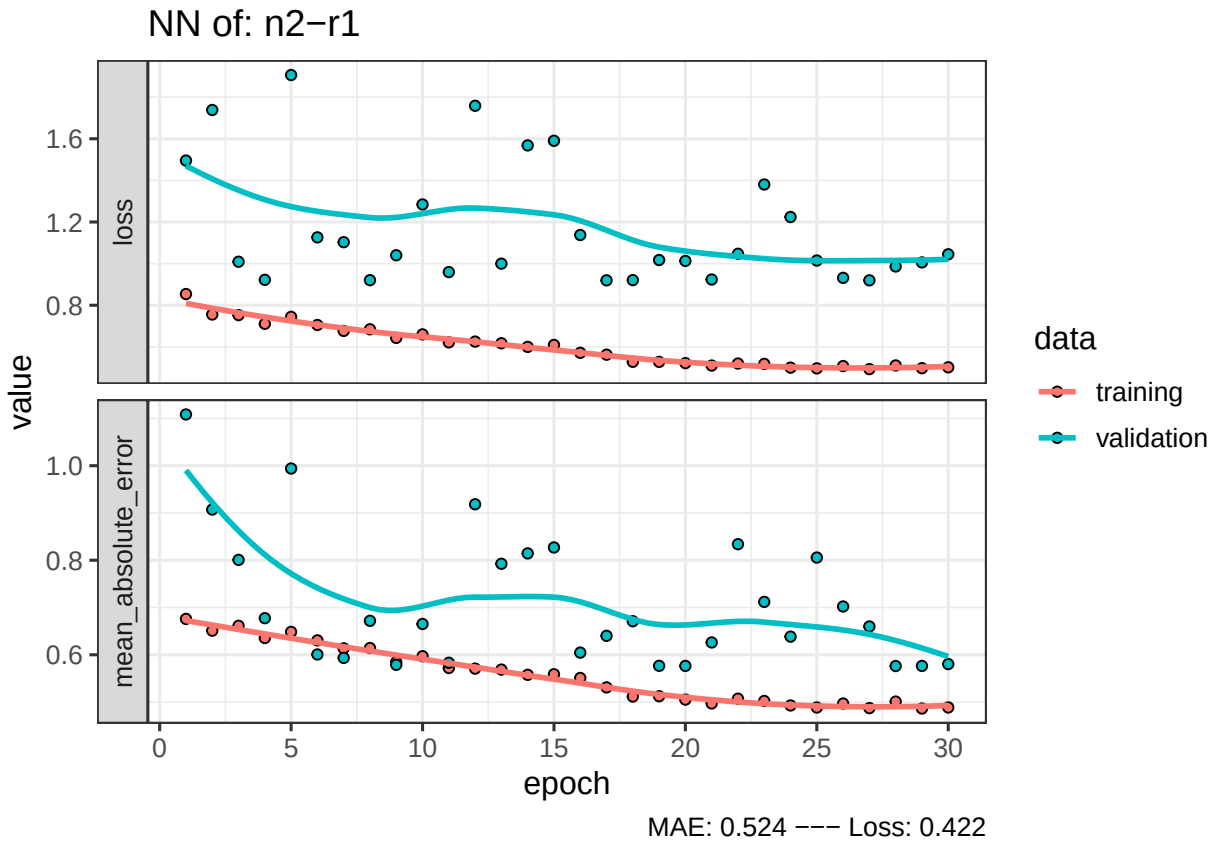



```
## `geom_smooth()` using formula 'y ~ x'
```

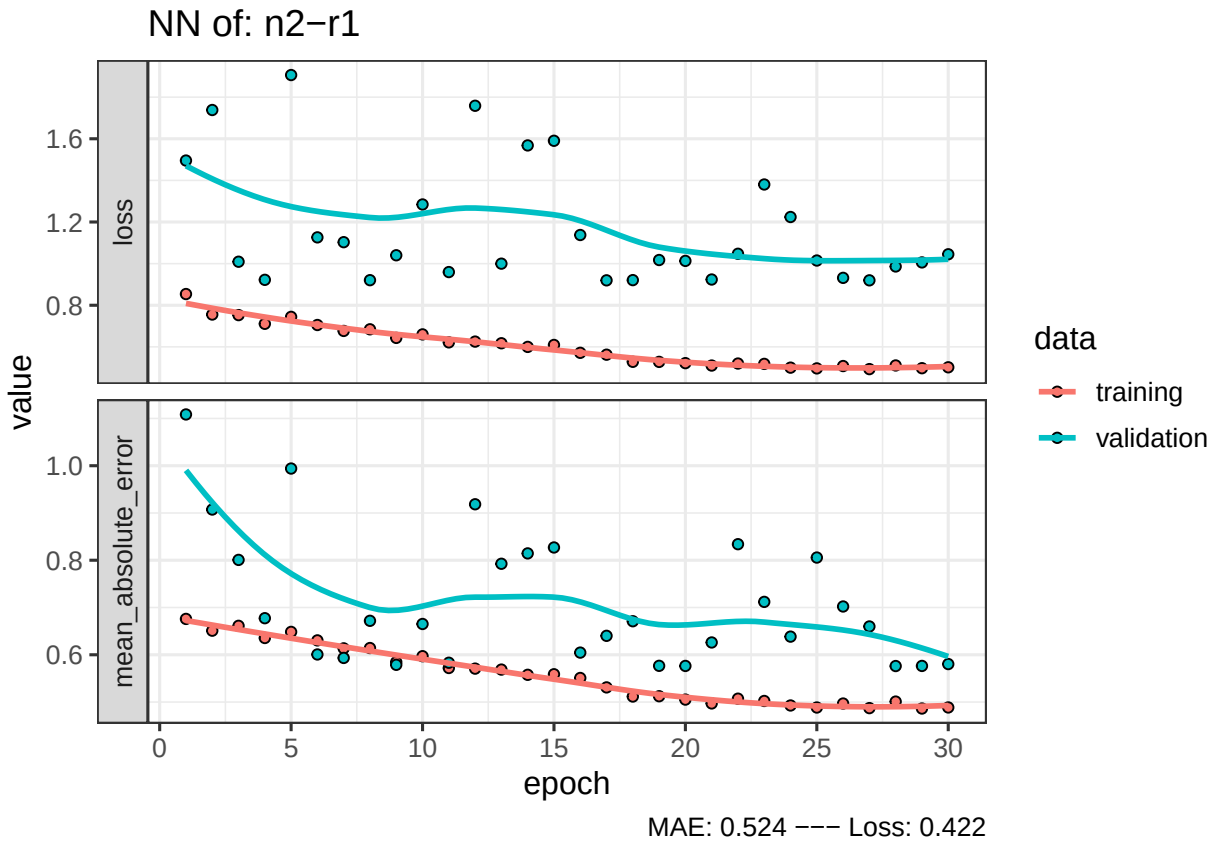



```
## MAE:  0.2944945
## Loss:  0.389903
##
## [1] "========== n2-r1 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

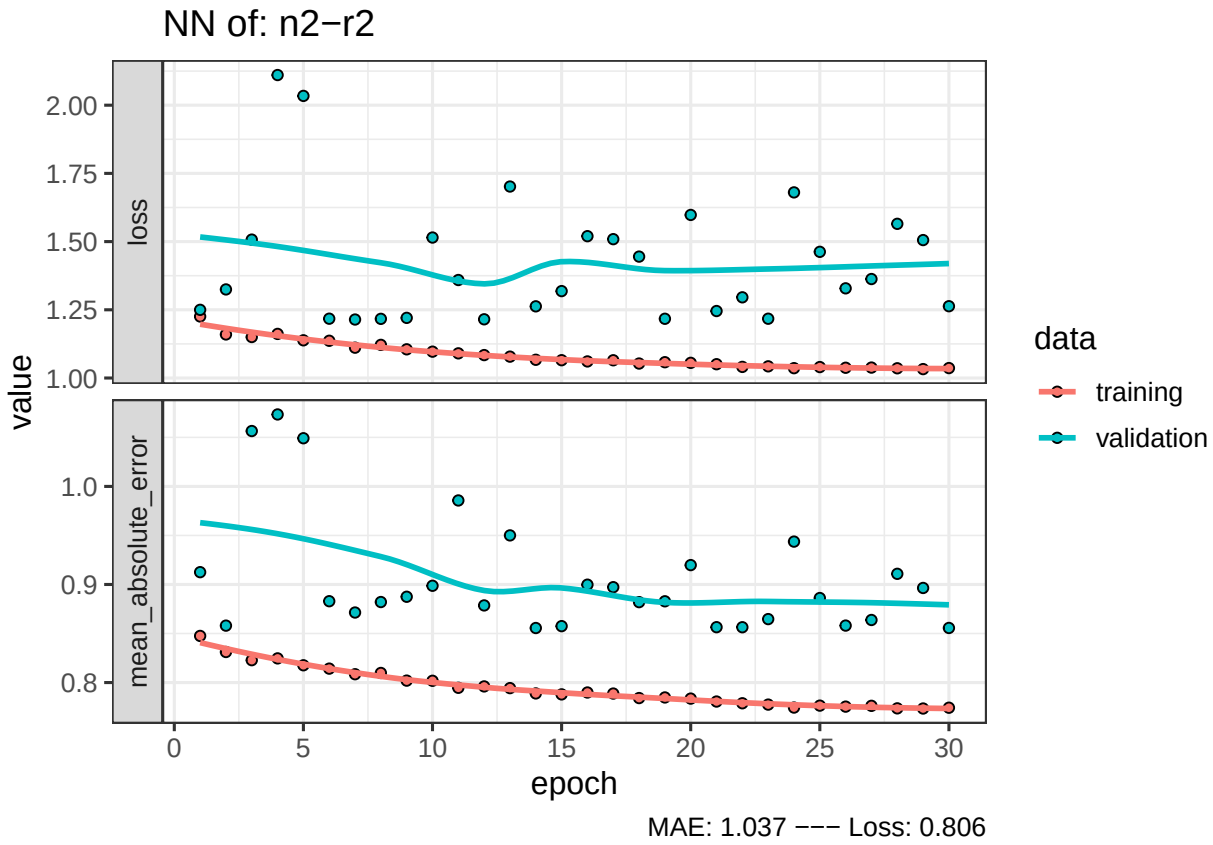



```
## `geom_smooth()` using formula 'y ~ x'
```

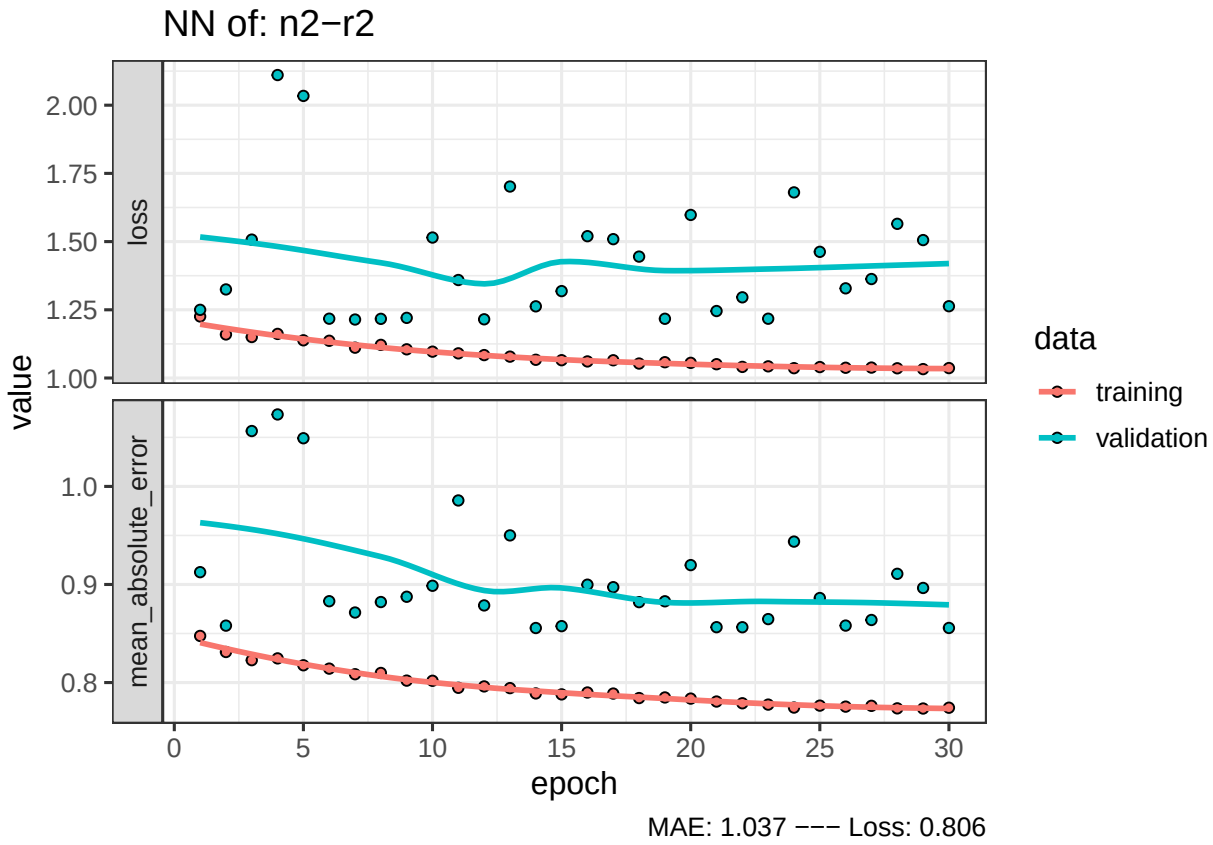



```
## MAE:  0.5244784
## Loss:  0.4219968
##
## [1] "========== n2-r2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

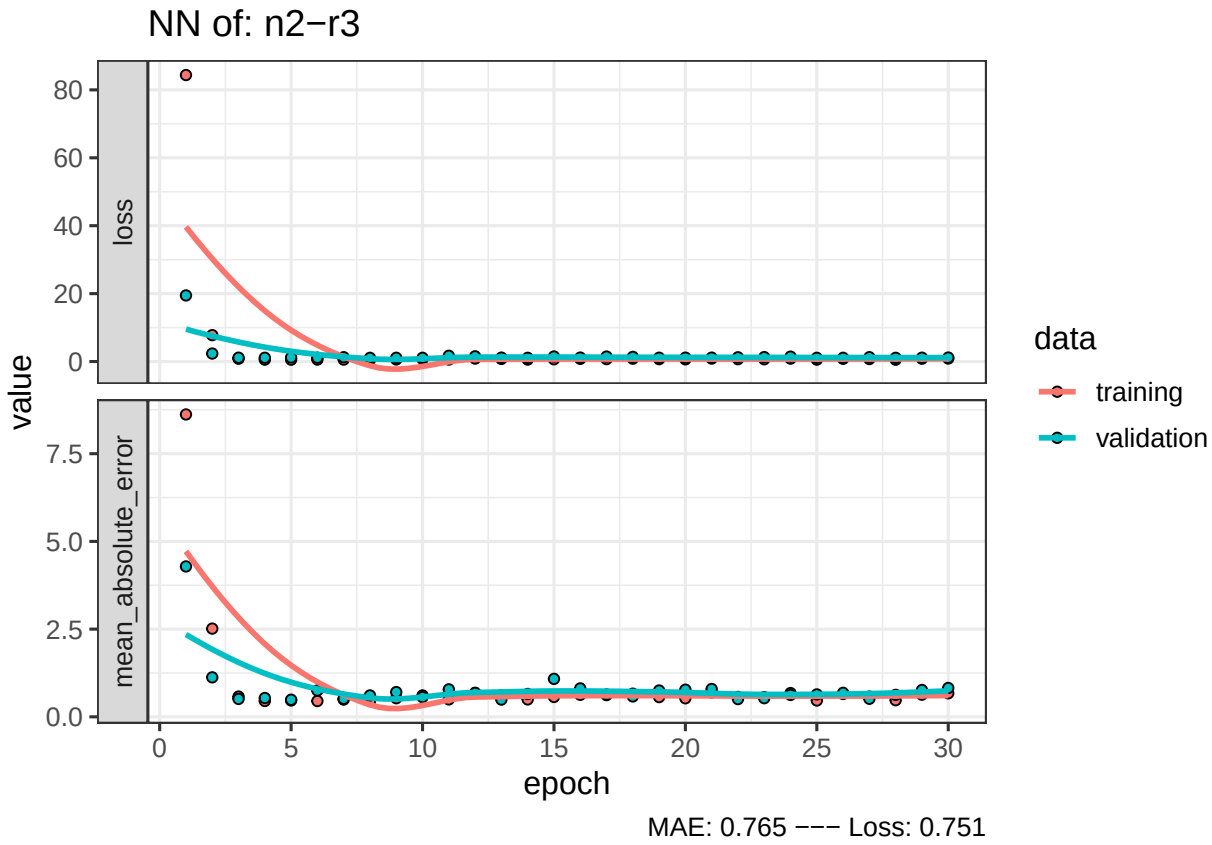



```
## `geom_smooth()` using formula 'y ~ x'
```

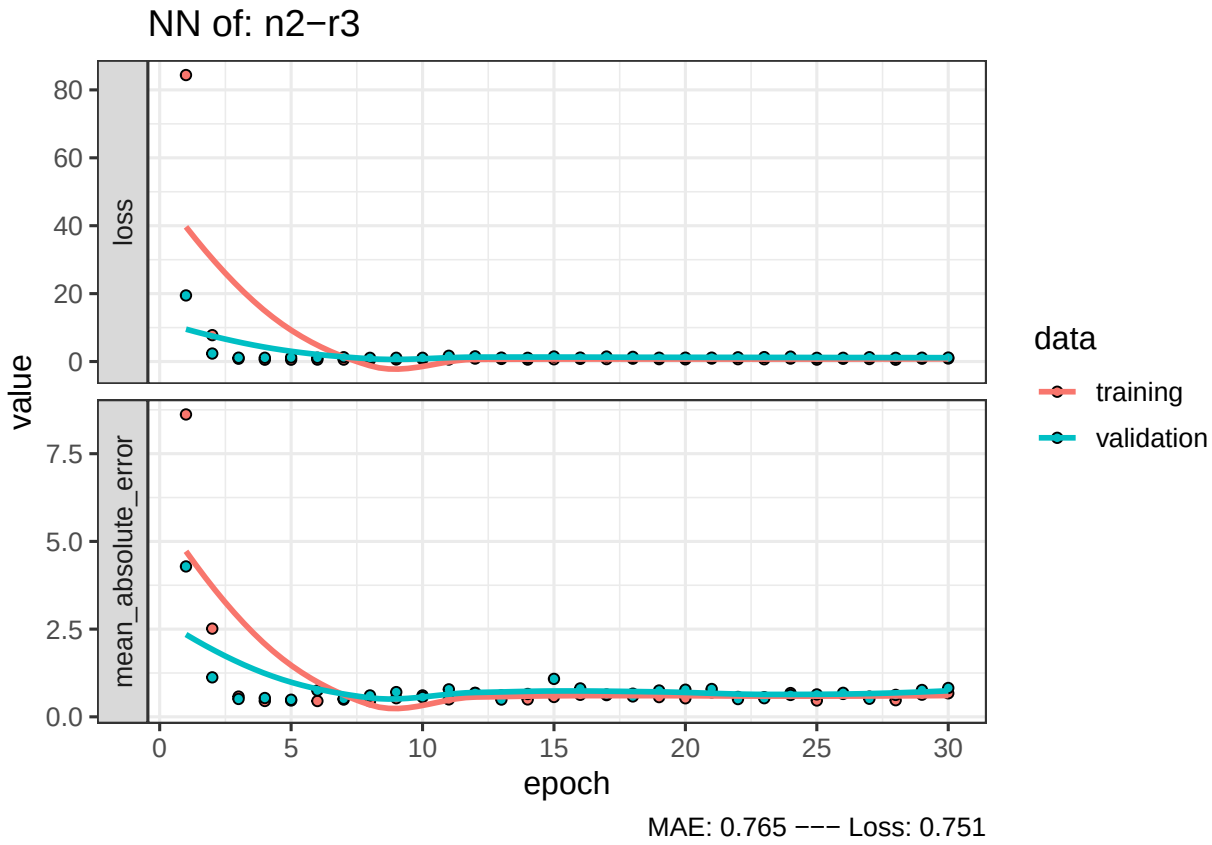



```
## MAE:  1.037113
## Loss:  0.8063703
##
## [1] "========== n2-r3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

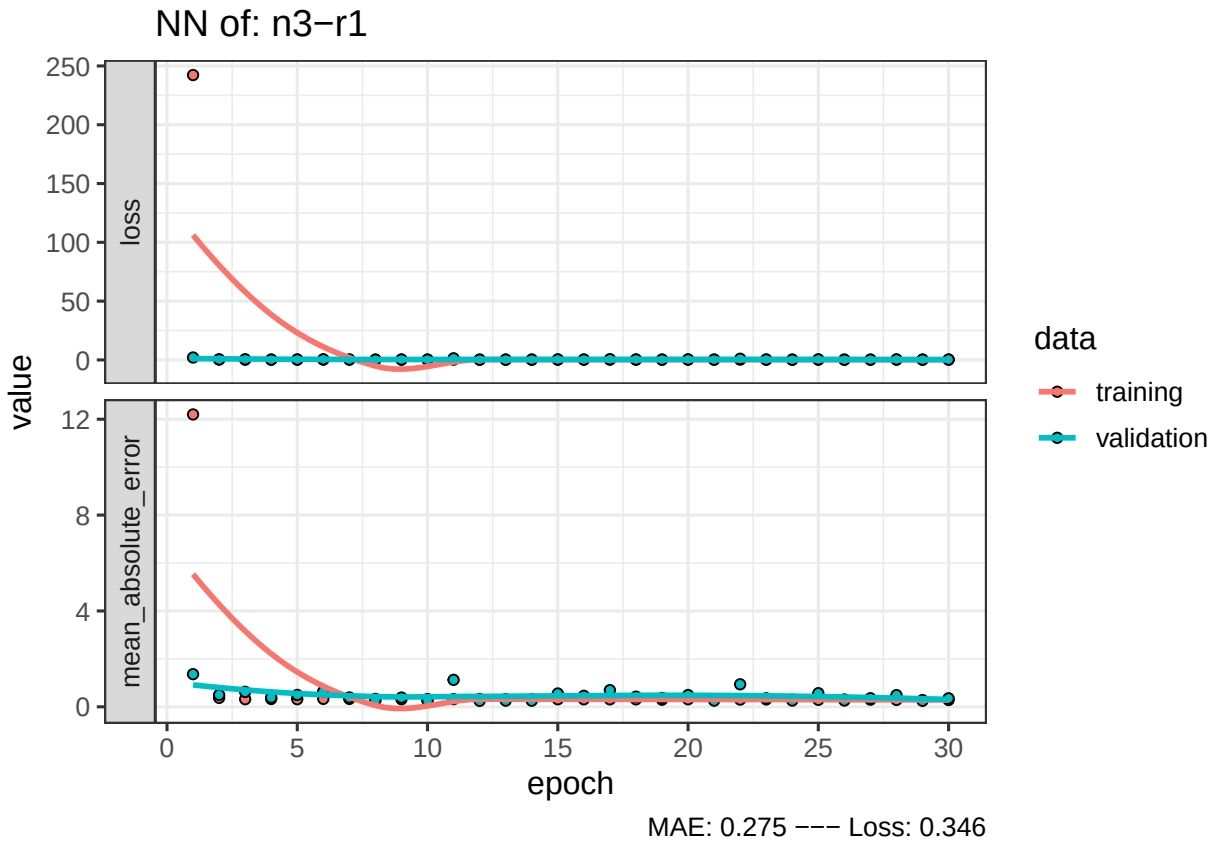



```
## `geom_smooth()` using formula 'y ~ x'
```

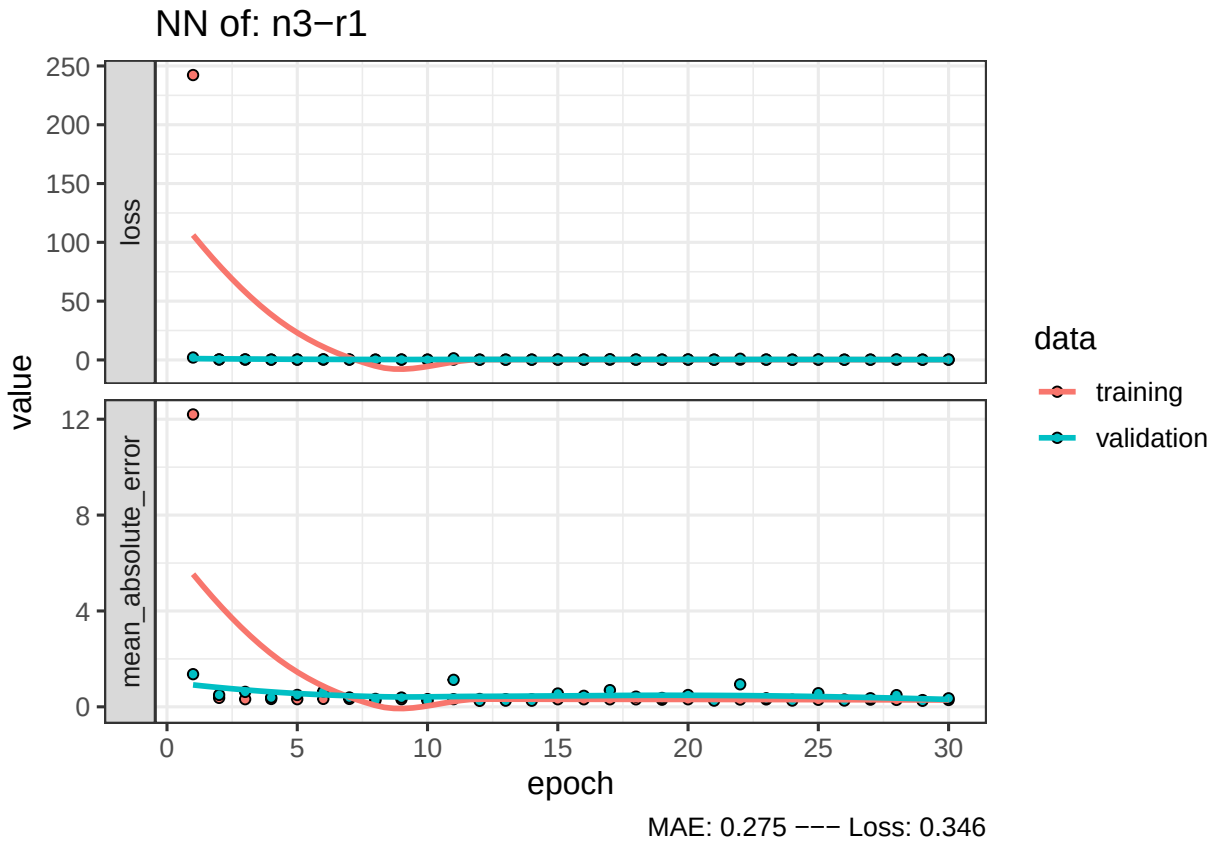



```
## MAE:  0.7650962
## Loss:  0.750608
##
## [1] "========== n3-r1 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

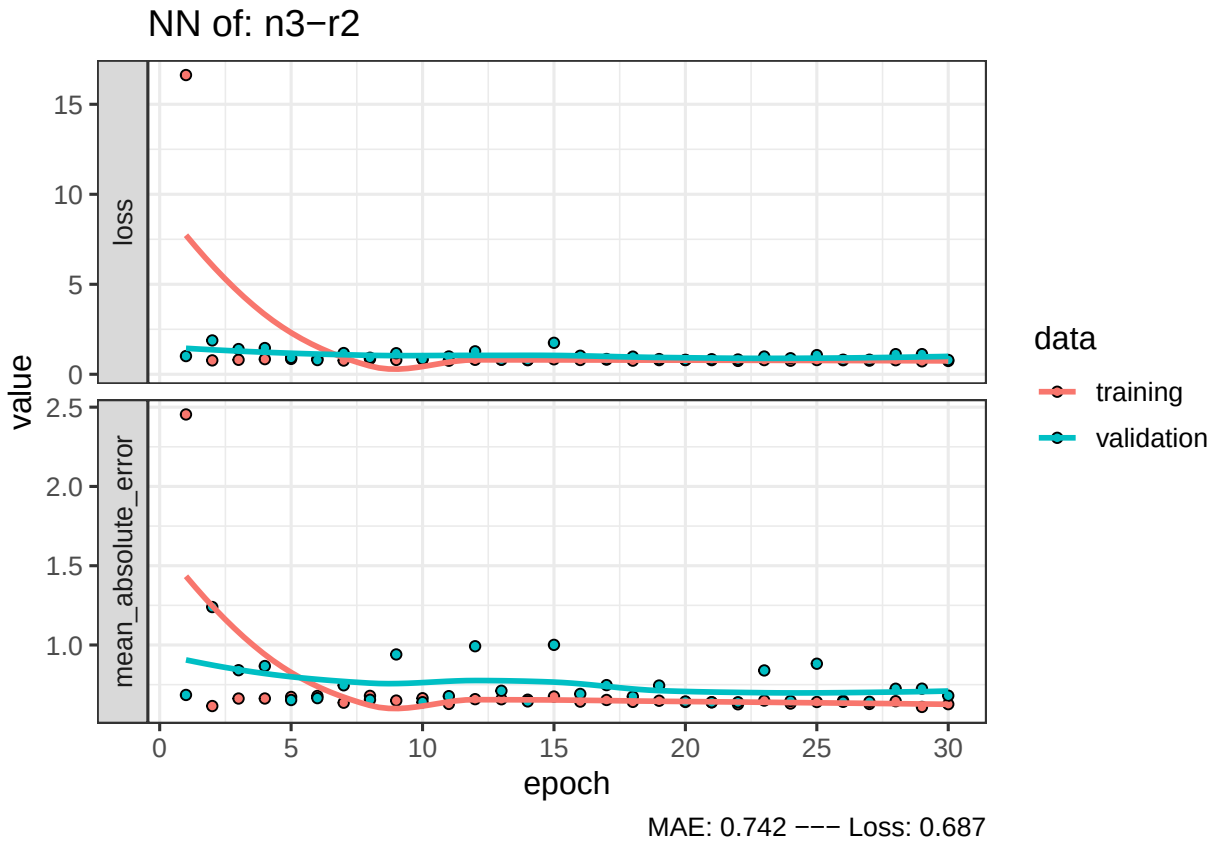



```
## `geom_smooth()` using formula 'y ~ x'
```

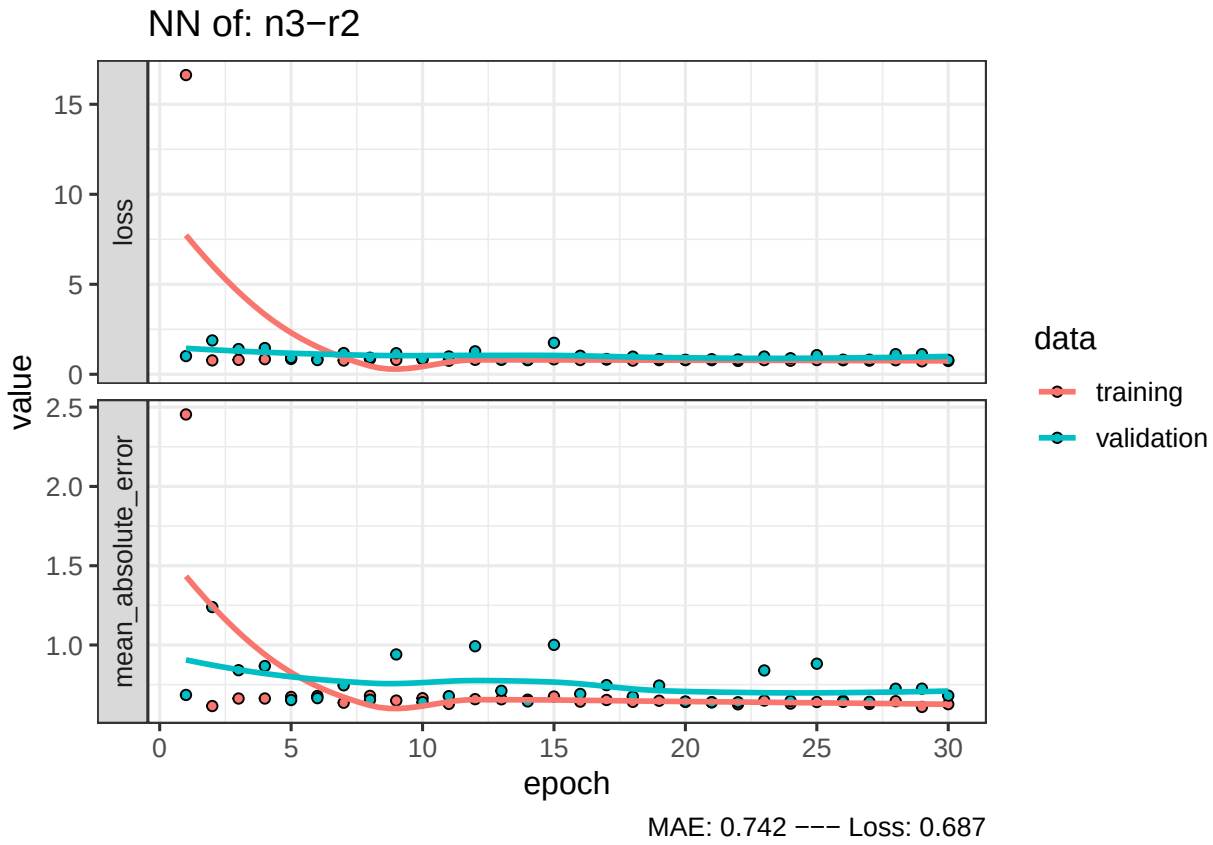



```
## MAE:  0.2753657
## Loss:  0.3455145
##
## [1] "========== n3-r2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

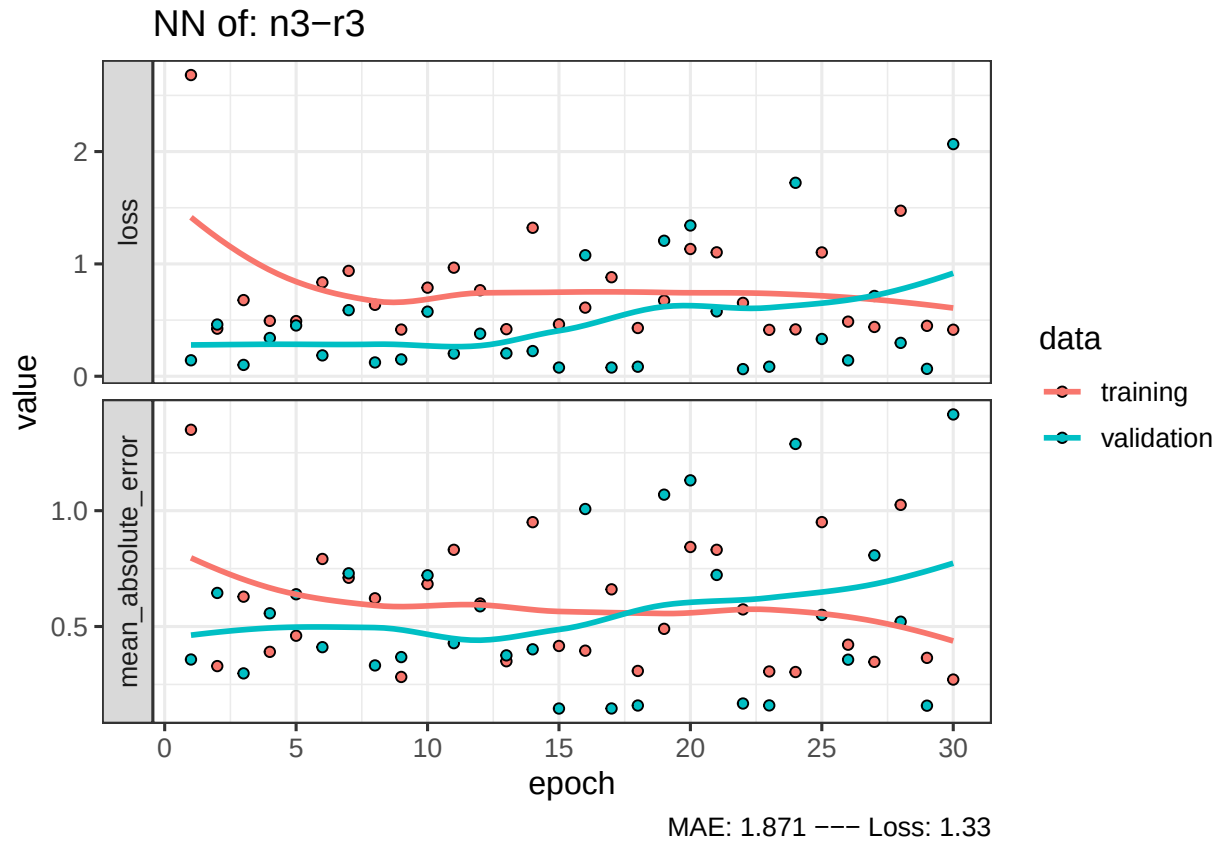



```
## `geom_smooth()` using formula 'y ~ x'
```

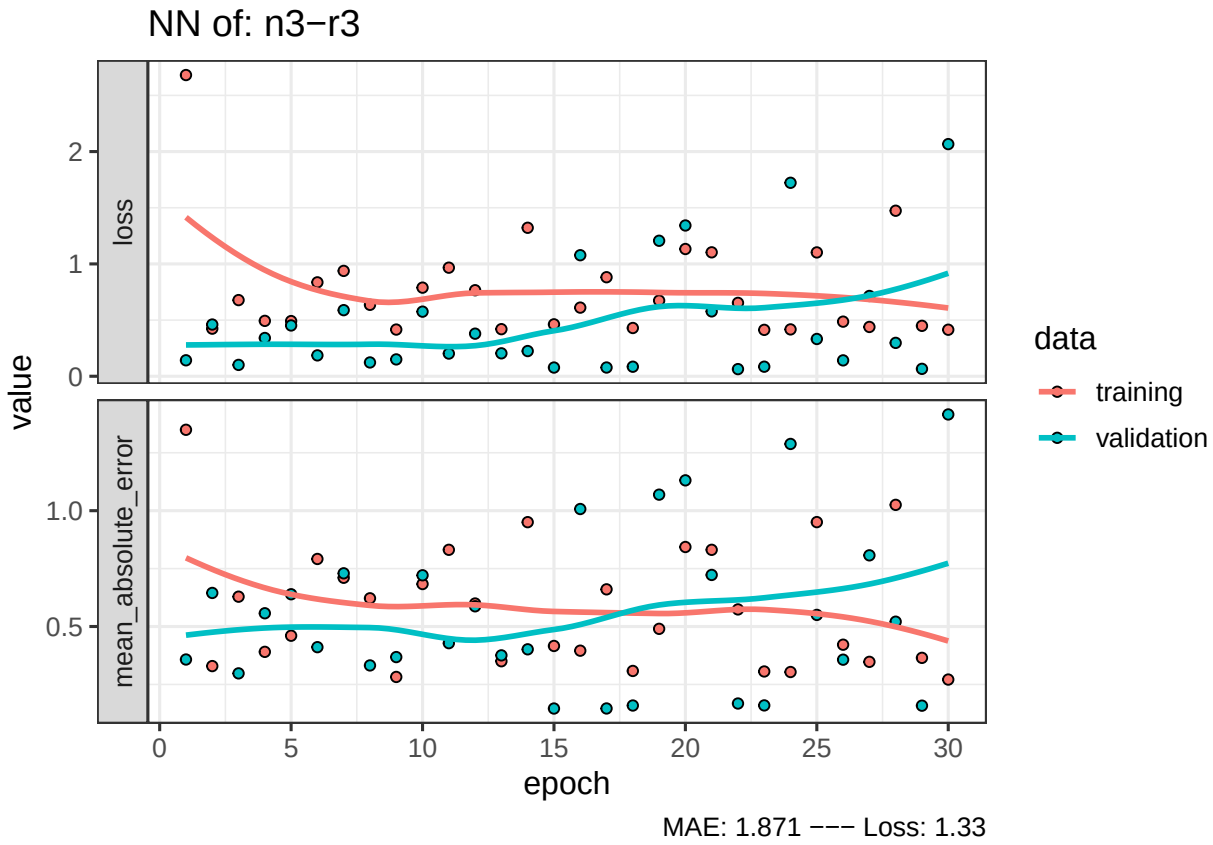



```
## MAE:  0.7421407
## Loss:  0.686858
##
## [1] "========== n3-r3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

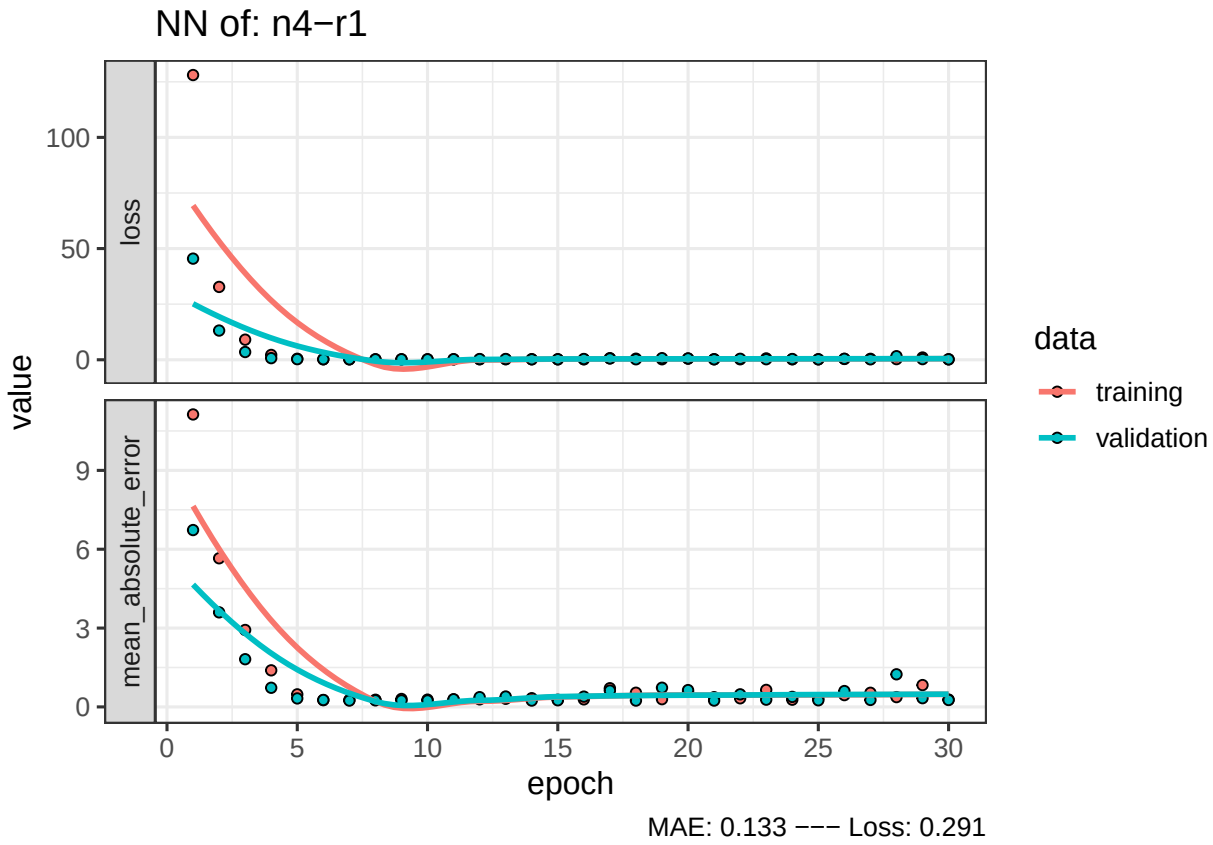



```
## `geom_smooth()` using formula 'y ~ x'
```

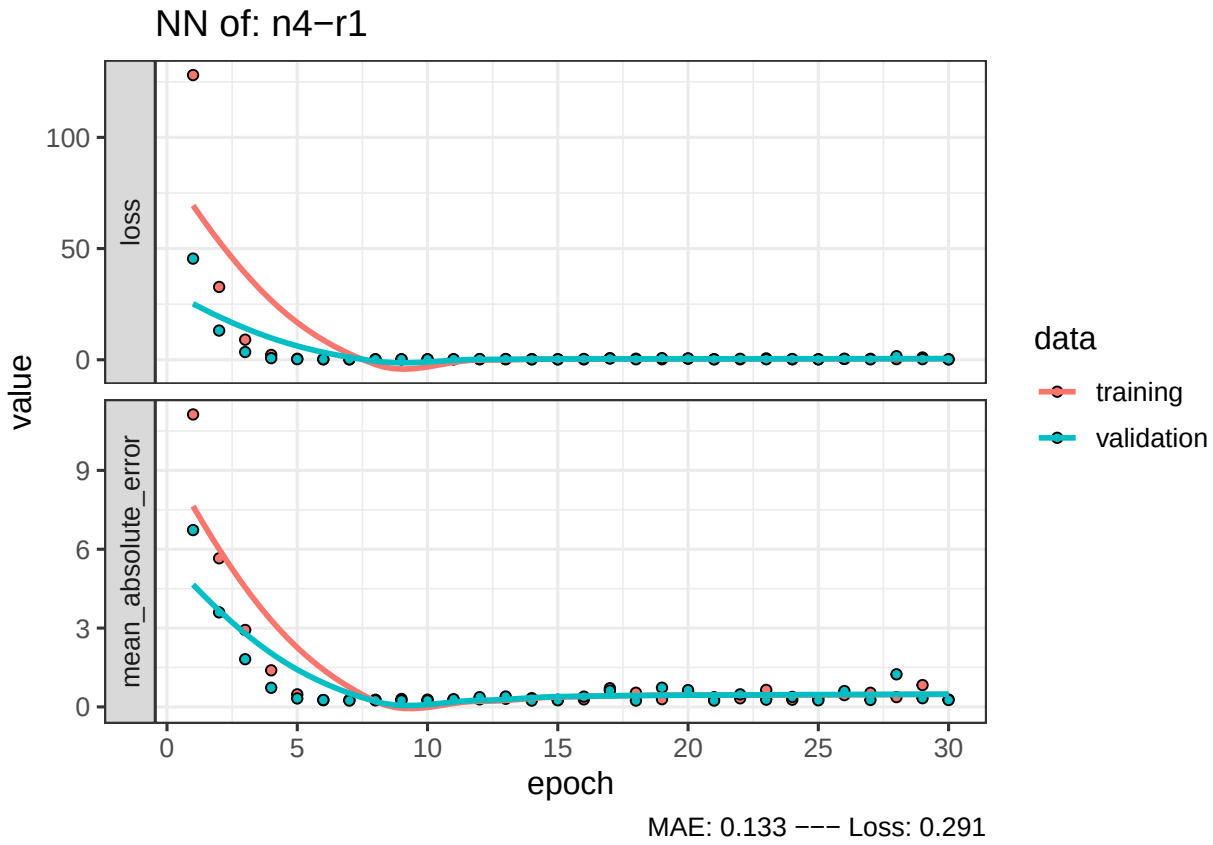



```
## MAE:  1.87118
## Loss:  1.32991
##
## [1] "========== n4-r1 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

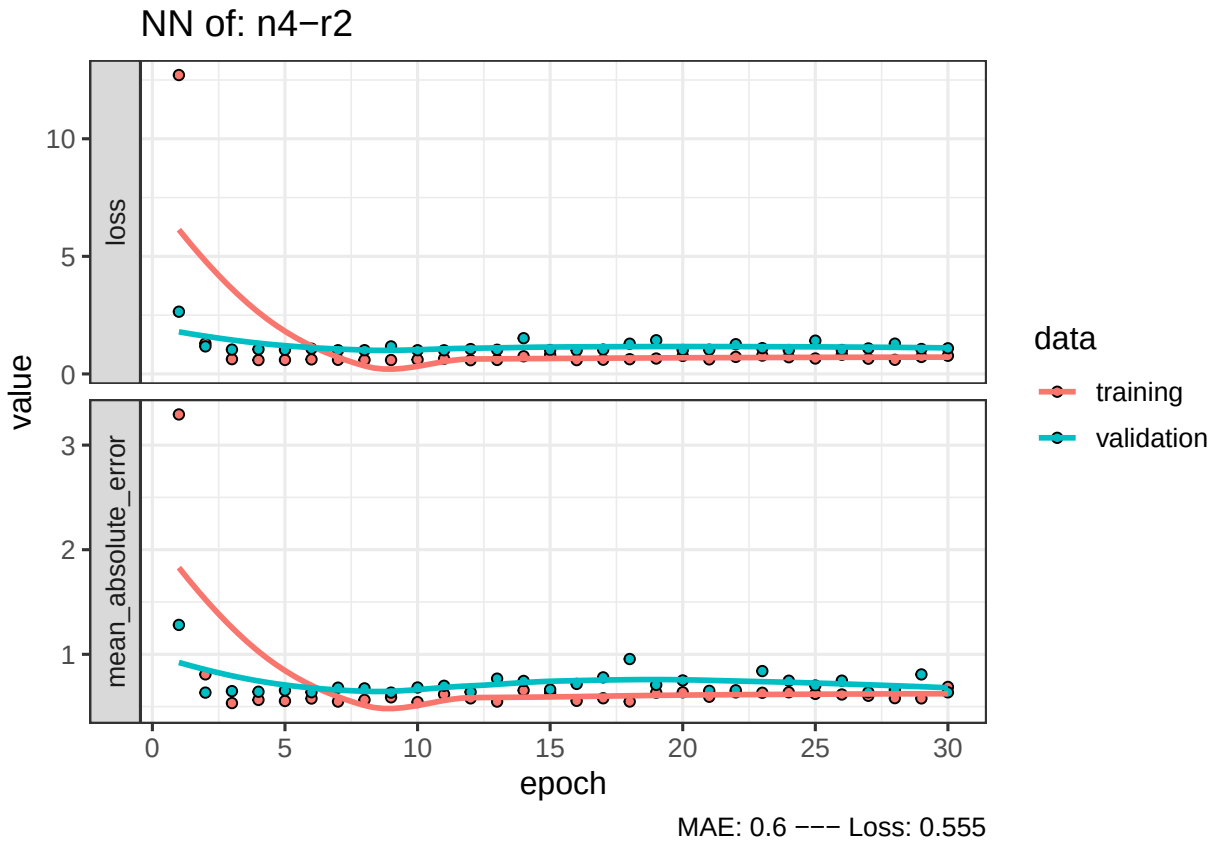



```
## `geom_smooth()` using formula 'y ~ x'
```

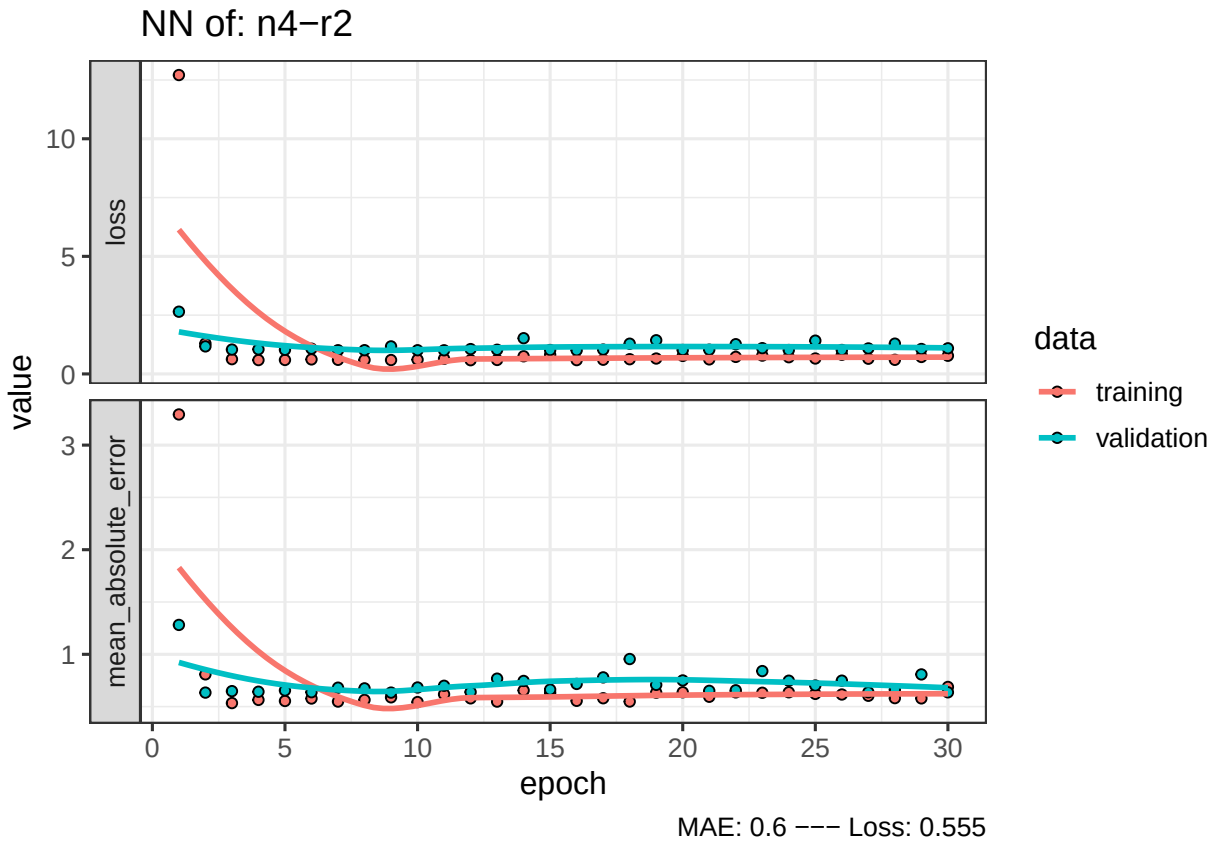



```
## MAE:  0.1332488
## Loss:  0.2908146
##
## [1] "========== n4-r2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

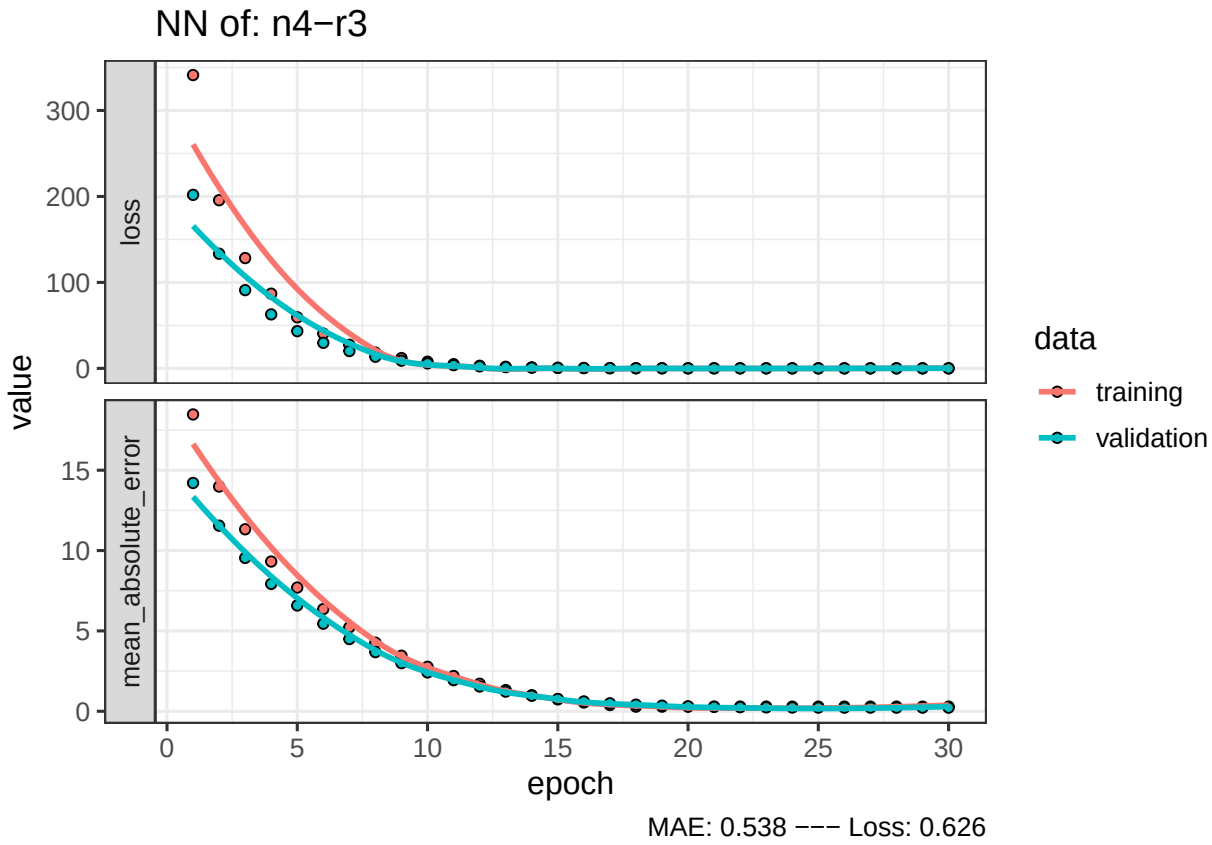



```
## `geom_smooth()` using formula 'y ~ x'
```

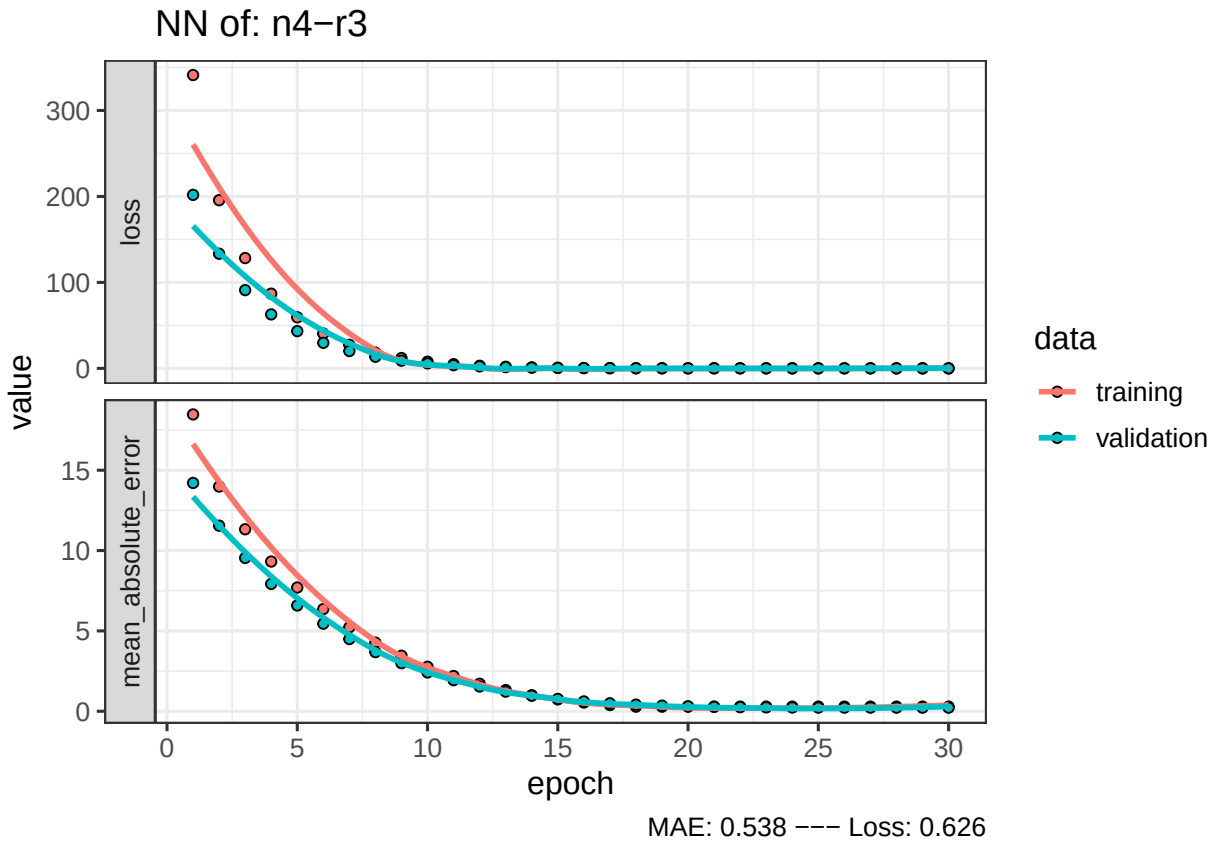



```
## MAE:  0.6004597
## Loss:  0.555418
##
## [1] "========== n4-r3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

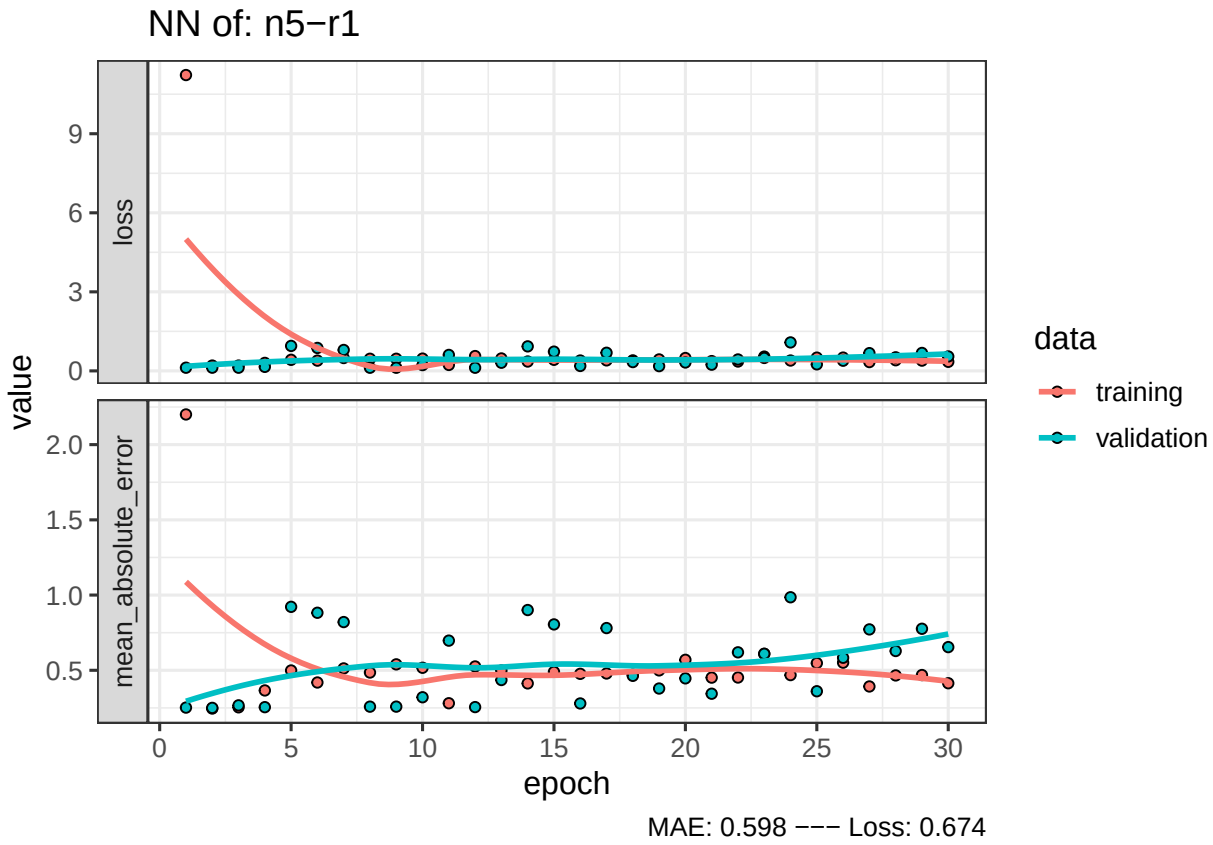



```
## `geom_smooth()` using formula 'y ~ x'
```

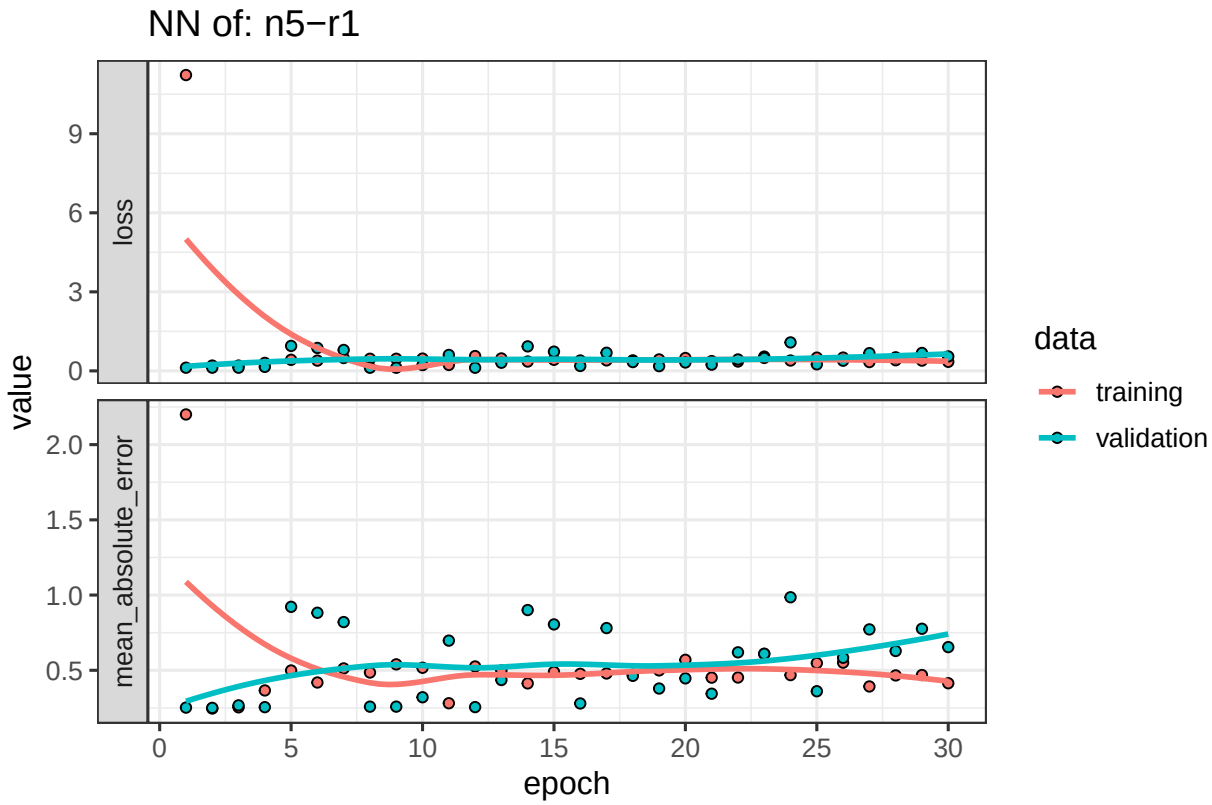



```
## MAE:  0.5377738
## Loss:  0.6261361
##
## [1] "========== n5-r1 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

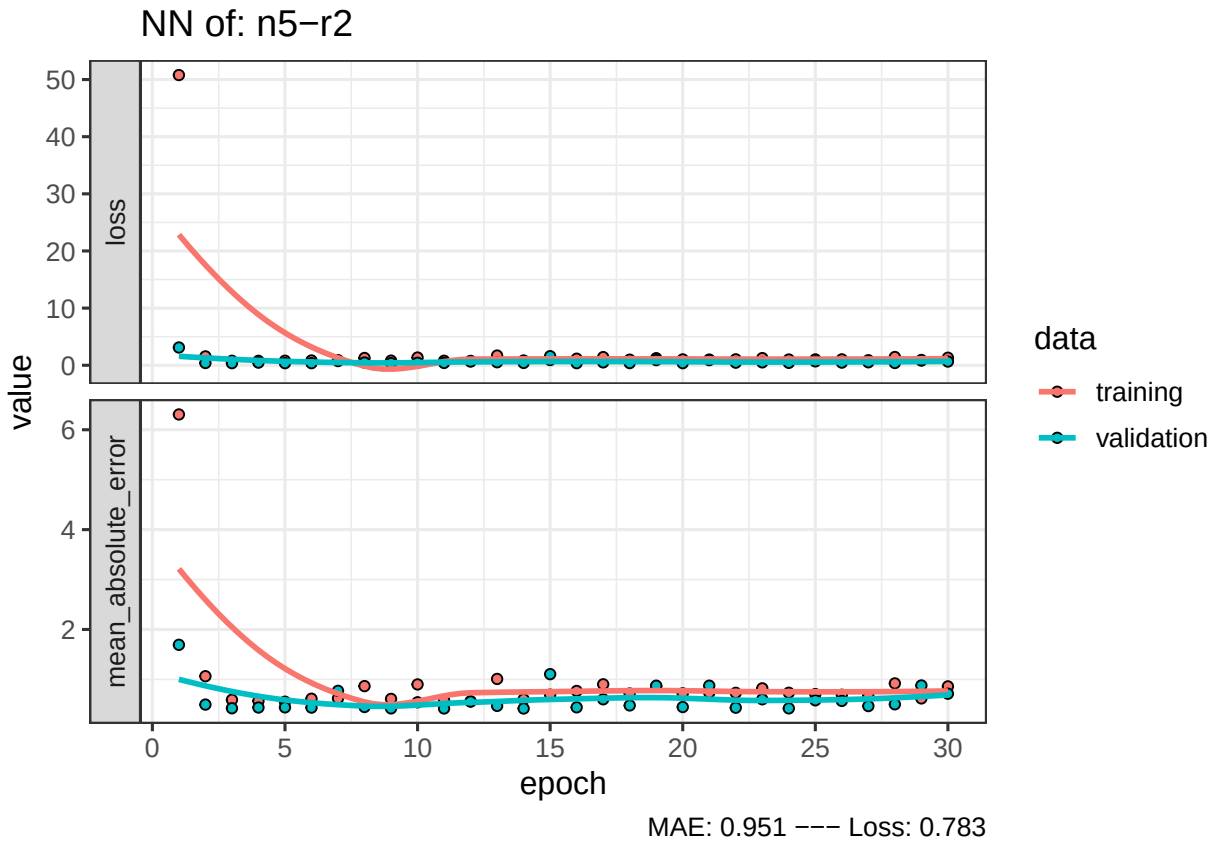



```
## `geom_smooth()` using formula 'y ~ x'
```

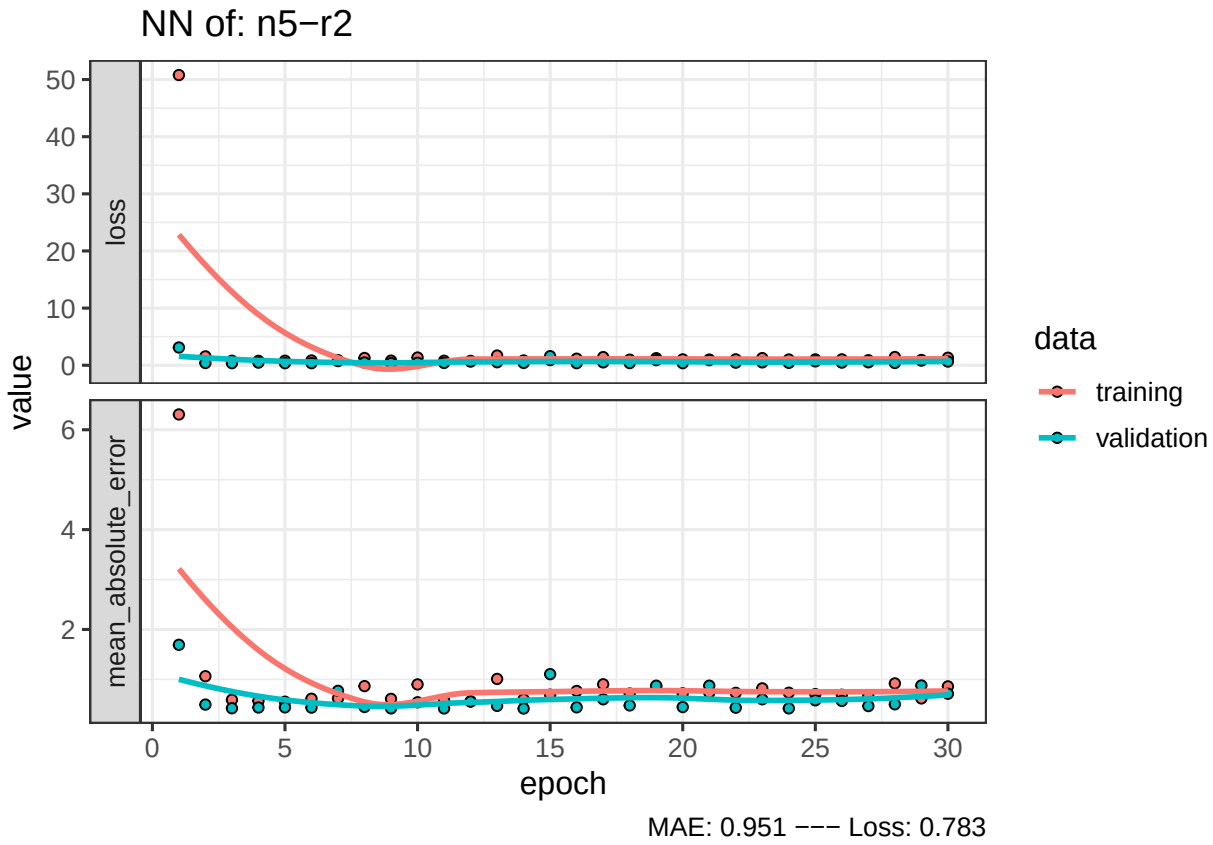



```
## MAE:  0.5982274
## Loss:  0.6743885
##
## [1] "========== n5-r2 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

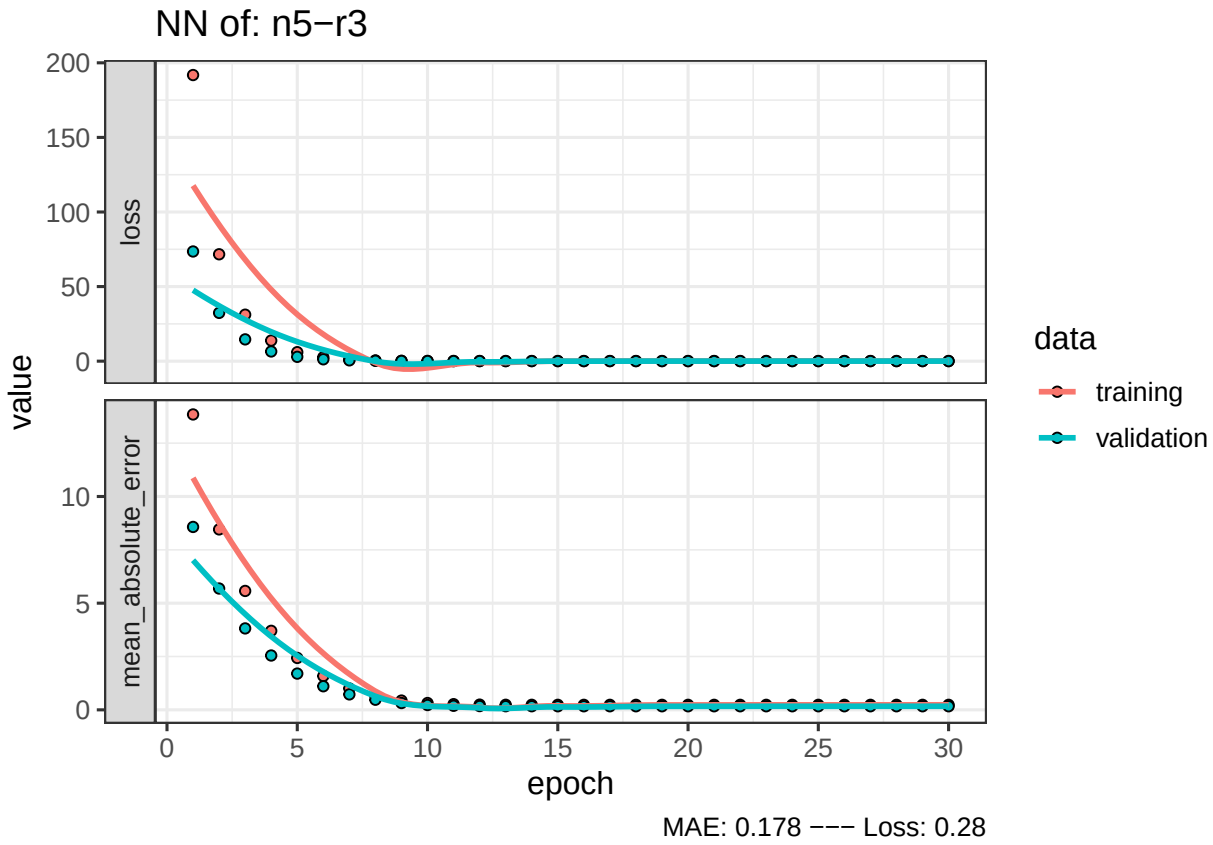



```
## `geom_smooth()` using formula 'y ~ x'
```

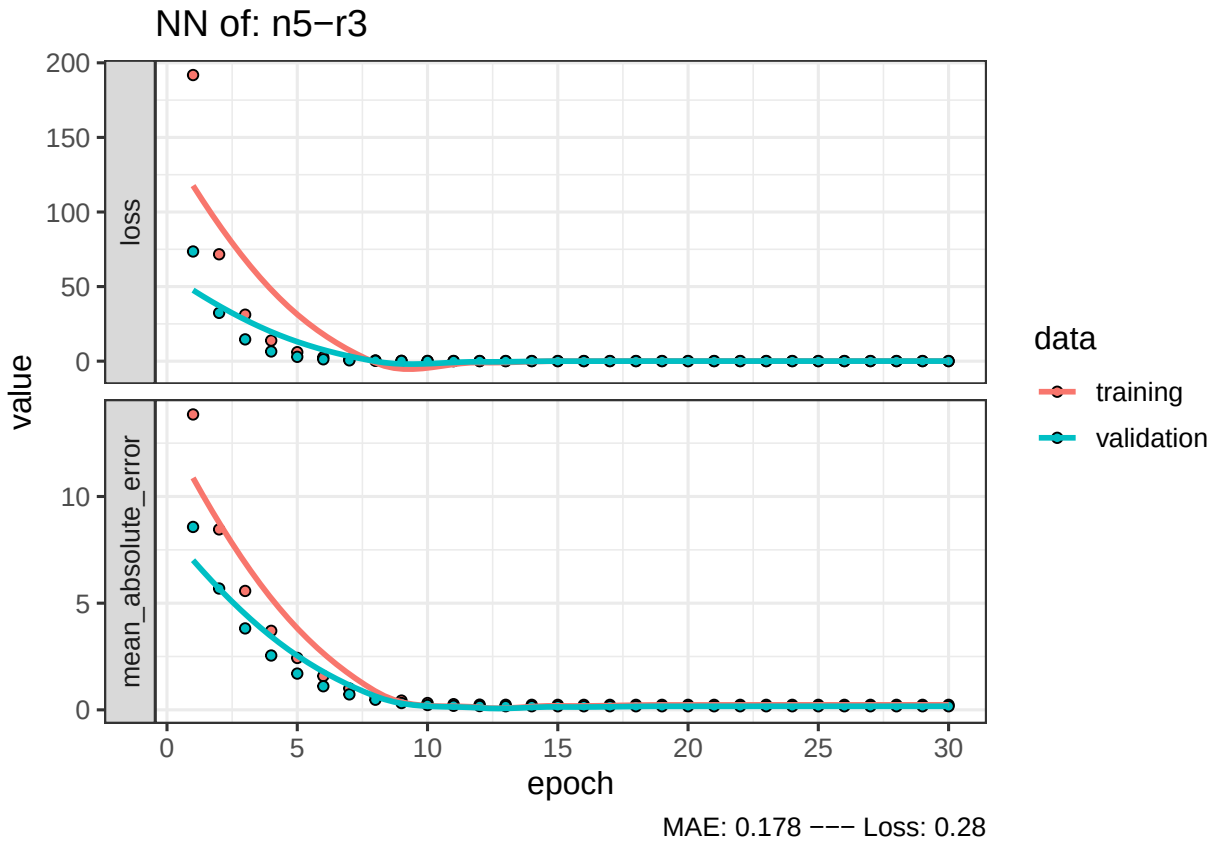



```
## MAE:  0.9505594
## Loss:  0.7829418
##
## [1] "========== n5-r3 =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```

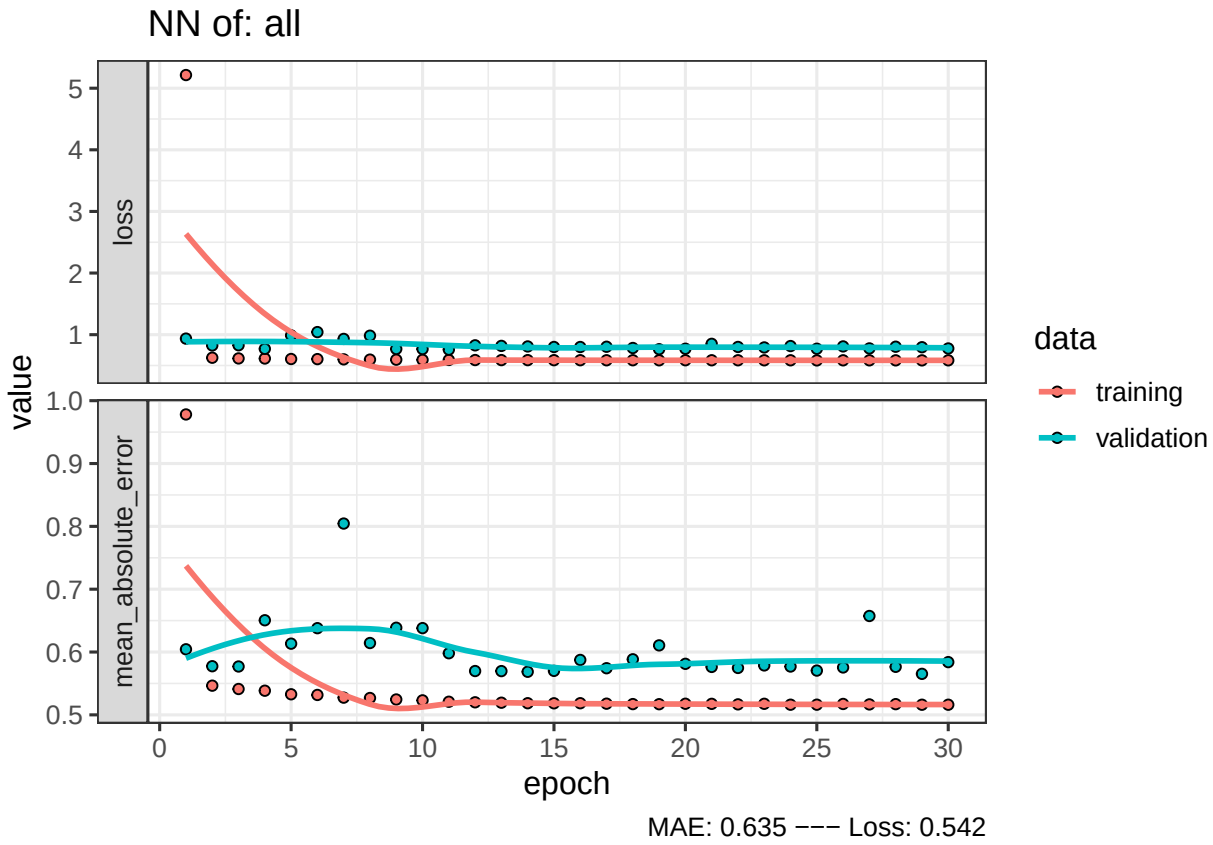



```
## `geom_smooth()` using formula 'y ~ x'
```

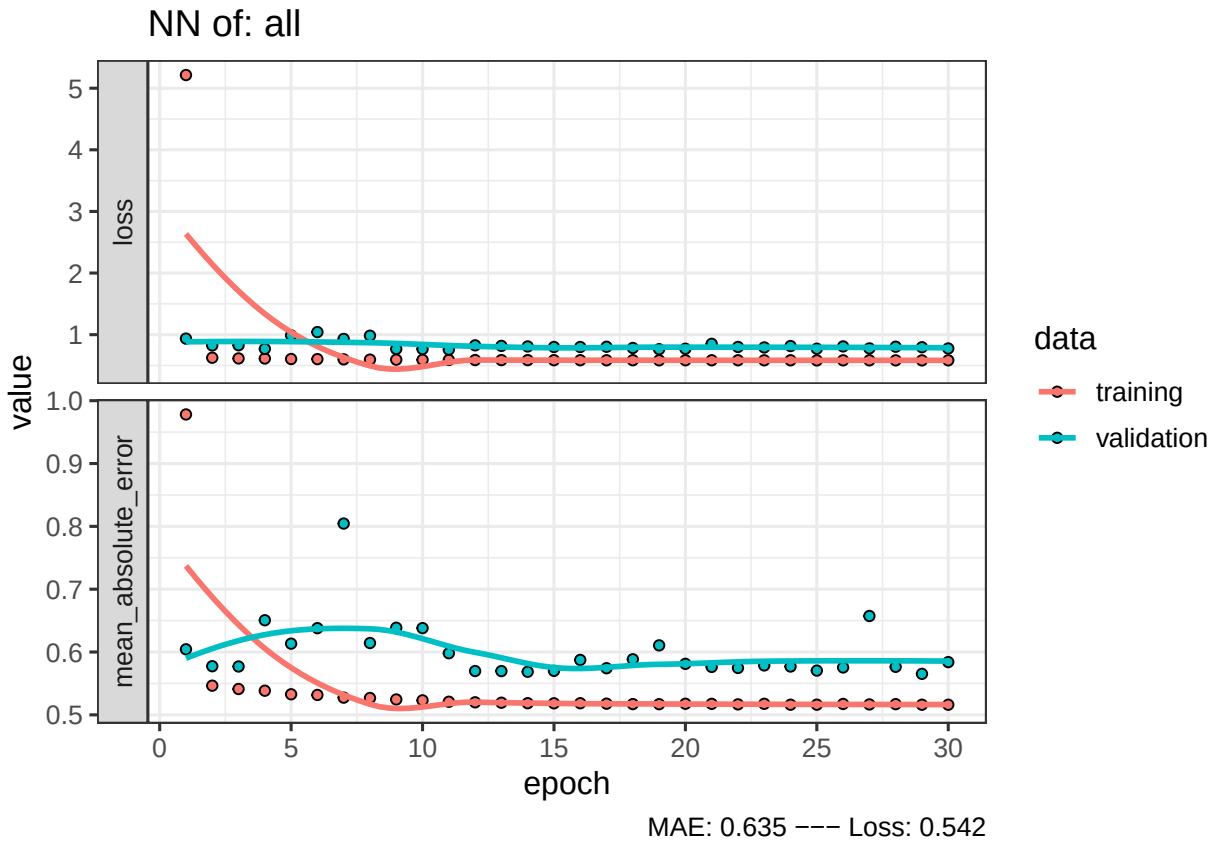



```
## MAE:  0.1779435
## Loss:  0.2803045
##
## [1] "========== all =========="
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
## `geom_smooth()` using formula 'y ~ x'
```




```
## MAE:  0.6347676
## Loss:  0.5417054
```

Comparison between the models

```
conc = c()

for (n in unique(ds$neighbourhood_group))
{
  conc = c(conc,n)
}
for (n in unique(ds$neighbourhood_group))
{
  for(r in unique(ds$room_type))
  {
    conc = c(conc, paste(n,"-",r))

  }
}
conc = c(conc,"All")
```

```
n = names(nn)



cols = vector("list")
cols[["Subset"]] = conc
for( m in model_lis)
{
  col = c()
  for( nam in n)
  {

    if( nam != "name")
    {
      col = c(col,m[[nam]]$MSE)

    }

  }
  cols[[m$name]] = col

}


mse_df = as.data.frame(cols)
best = c()
for (i in 1:length(cols$Subset))
{
  m = min(mse_df[i,2:6])
  col = which(mse_df[i,1:6 ] == m)
  nam = names(mse_df)[col]
  best = c(best, nam)
}
mse_df$Best = best
#kableExtra::kable(mse_df)%>%
#  kable_styling(bootstrap_options = "striped", full_width = F,font_size = 20) %>%
##  row_spec(0, bold = T, color = "white", background = "#D7261E")%>%
##  column_spec(1, bold = T, border_right = T,color = "white", background = "#191970")%>%
#  column_spec(2:6,extra_css="text-align:Center")

mse_df
```

```
##                          Subset Linear.Regression Decision.Tree
## 1                      Brooklyn         0.4581633     0.4607817
## 2                     Manhattan         0.7581404     0.7634093
## 3                        Queens         0.3846665     0.3864543
## 4                Staten Island         0.3467308     0.3471879
## 5                         Bronx         0.2741482     0.2744778
## 6          Brooklyn - Private room         0.1818182     0.1771088
## 7       Brooklyn - Entire home/apt         0.7598732     0.7503308
## 8           Brooklyn - Shared room         0.2639579     0.2505339
## 9         Manhattan - Private room         0.4445611     0.3753113
## 10     Manhattan - Entire home/apt         0.9361941     0.9585893
## 11         Manhattan - Shared room         0.4904792     0.4992158
```

```
## 12           Queens - Private room         0.1616290     0.1551588
## 13        Queens - Entire home/apt         0.6992986     0.6995247
## 14            Queens - Shared room         0.1659605     0.1667691
## 15    Staten Island - Private room         0.1332495     0.1605553
## 16 Staten Island - Entire home/apt         0.5703345     0.7737648
## 17     Staten Island - Shared room         0.3209208     0.5818964
## 18            Bronx - Private room         0.1475827     0.1611146
## 19         Bronx - Entire home/apt         0.8252451     0.8860040
## 20             Bronx - Shared room         0.2071295     0.1939584
## 21                             All         0.6167807     0.6149423
##    Random.Forest Ranger.Random.Forest Neural.Networks                 Best
## 1      0.4376492            0.4356612       0.4877132 Ranger.Random.Forest
## 2      0.7150546            0.7132323       0.8333294 Ranger.Random.Forest
## 3      0.3601118            0.3594399       0.3867755 Ranger.Random.Forest
## 4      0.3433679            0.3498251       0.4173280        Random.Forest
## 5      0.2714934            0.2680627       0.2938451 Ranger.Random.Forest
## 6      0.1837738            0.1830508       0.2526560        Decision.Tree
## 7      0.8019234            0.8056467       0.8234408        Decision.Tree
## 8      0.2406699            0.2386268       0.2944945 Ranger.Random.Forest
## 9      0.3784497            0.3782840       0.5244784        Decision.Tree
## 10     0.9771422            0.9743580       1.0371131    Linear.Regression
## 11     0.5635043            0.5715526       0.7650962    Linear.Regression
## 12     0.1517223            0.1527042       0.2753657        Random.Forest
## 13     0.7135788            0.7123861       0.7421406    Linear.Regression
## 14     0.2229526            0.2213944       1.8711798    Linear.Regression
## 15     0.1423578            0.1433902       0.1332488      Neural.Networks
## 16     0.7189537            0.7056397       0.6004597    Linear.Regression
## 17     0.3032279            0.5847496       0.5377738        Random.Forest
## 18     0.1330848            0.1327536       0.5982274 Ranger.Random.Forest
## 19     0.8700947            0.8708723       0.9505594    Linear.Regression
## 20     0.1763329            0.1768414       0.1779435        Random.Forest
## 21     0.5852325            0.5539925       0.6347674 Ranger.Random.Forest
```

Clustering and Groups

Clustering for Mixed type of data

```
clust_num = 5

get_clusters = function(dts, num, dim_plot,name)
{
  l = list()
  if(is.null(dts$room_type))
  {
    l$cl = kmeans(dts,num)
  }
```

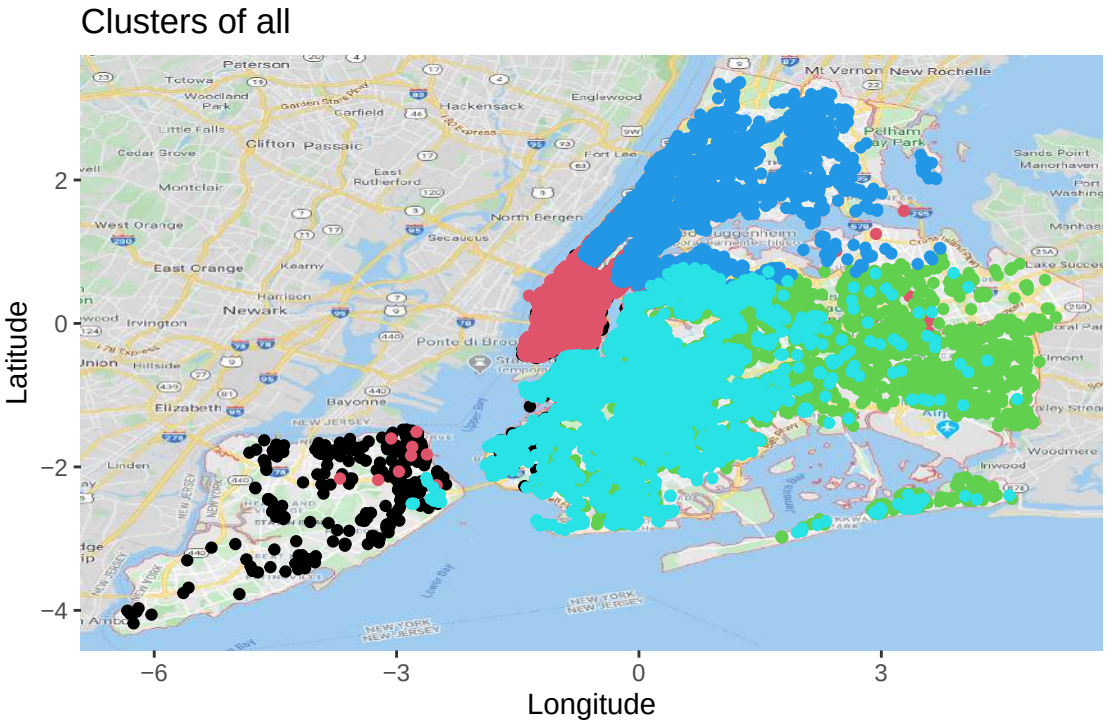
```
  else
  {
    l$cl = kproto(dts,num)

  }

  clust = list()

  for (i in 1:num)
  {
    indexes = l$cl$cluster == i
    clust[[i]] = dts[indexes,]
  }

  min_lat =  dim_plot[1]
  max_lat = dim_plot[2]
  min_long = dim_plot[3]
  max_long= dim_plot[4]


  myplot= ggplot() +  background_image(img)+  xlab('Longitude') +  ylab('Latitude')+ theme(plot.margin =


  count = 1
  l$clust_plot = vector("list")
  for(el in clust)
  {
    myplot = myplot+ geom_point(data = el, aes(y = latitude, x = longitude),color= count)

    p =ggplot()+ background_image(img)+geom_point(data = el, aes(y = latitude, x = longitude),color= cou

    l$clust_plot[[as.character(count)]] = p

    l$summary[[as.character(count)]] = summary(el)

    count= count+1
  }
  l$myplot = myplot
  return(l)
}

get_all_cluster = function(clust_data, clust_num, dim)
{
  lis = vector("list")
  plots = vector("list")

  cnt= 1
  for(sub in names(clust_data))
  {

    lis[[sub]] = get_clusters(clust_data[[sub]], clust_num, dim, sub)
    plots[[cnt]] = lis[[sub]]$myplot
    cnt = cnt+1
```

```
  }
  lis[["all_plots"]]= plots
  return(lis)
}

borders = c(min(clust_data$all$latitude) ,max(clust_data$all$latitude), min(clust_data$all$longitude), m
all_cluster = get_all_cluster(clust_data,5, borders)
```

```
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.8287906
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.272312
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.701065
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.935192
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.7781472
##
## # NAs in variables:
## neighbourhood_group            latitude           longitude           room_type
##                   0                   0                   0                   0
##               price
##                   0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.747748
```

```
multiplots <- function(plotlist, file=NULL,cols = 2, layout = NULL) {
  require(grid)
```

```
  plots <- c(plotlist)

  numPlots = length(plots)

  if (is.null(layout)) {
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols),byrow =T)
  }

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

  } else {
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    for (i in 1:numPlots) {
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

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

all_cluster$all_plots[[21]]
```




```
all_cluster$all$summary
```

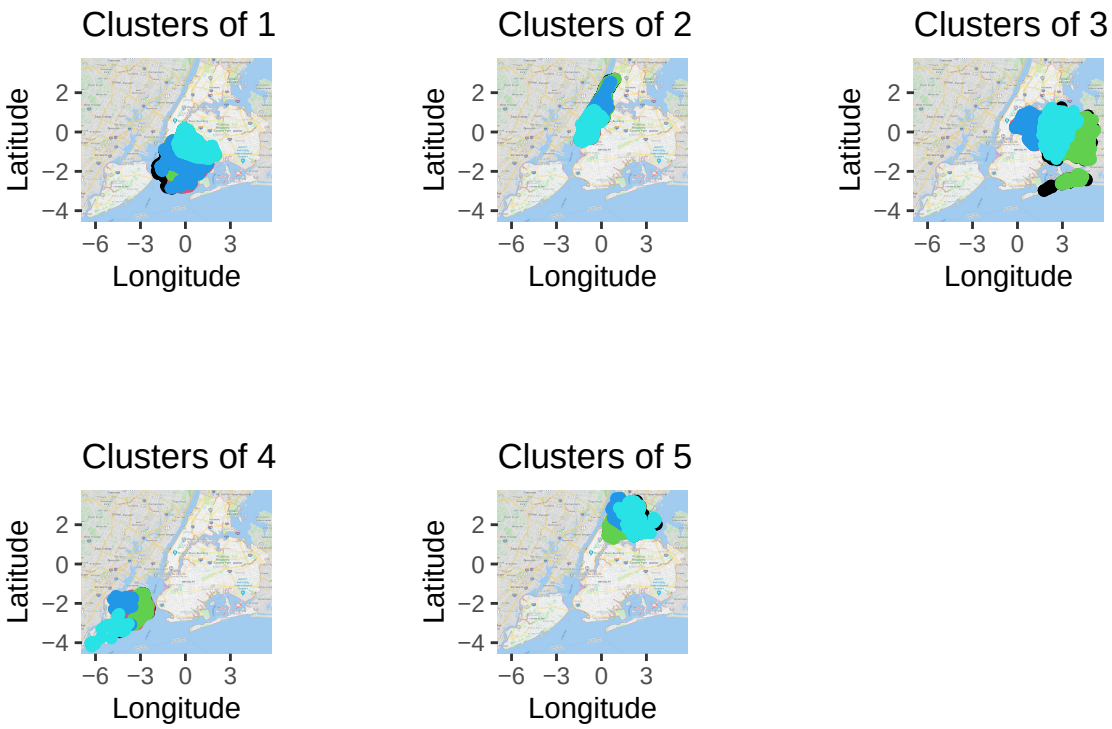
```
## $`1`
##     neighbourhood_group    latitude          longitude
##  Brooklyn      : 460     Min.   :-4.18085   Min.   :-6.3324
##  Manhattan     :4431     1st Qu.:-0.25722   1st Qu.:-1.0416
##  Queens        :  25     Median :-0.02283   Median :-0.8472
##  Staten Island: 347     Mean   :-0.13418   Mean   :-1.0242
##  Bronx         :   0     3rd Qu.: 0.37603   3rd Qu.:-0.7134
##                          Max.   : 0.98828   Max.   : 0.1238
##            room_type         price
##  Private room   :4126   Min.   :-1.3248
##  Entire home/apt: 882   1st Qu.:-0.6432
##  Shared room    : 255   Median :-0.3706
##                         Mean   :-0.3519
##                         3rd Qu.:-0.1320
##                         Max.   : 1.9127
##
## $`2`
##     neighbourhood_group    latitude          longitude
##  Brooklyn      :  581    Min.   :-2.25573   Min.   :-3.7011
##  Manhattan     :10566    1st Qu.:-0.01754   1st Qu.:-0.9270
##  Queens        :  289    Median : 0.34290   Median :-0.6884
##  Staten Island:    9     Mean   : 0.37677   Mean   :-0.6004
##  Bronx         :   23    3rd Qu.: 0.69443   3rd Qu.:-0.3314
##                          Max.   : 2.84824   Max.   : 3.7225
```

```
##            room_type         price
##  Private room   :  448   Min.   :-0.4160
##  Entire home/apt:10977   1st Qu.: 0.4359
##  Shared room    :   43   Median : 0.8903
##                          Mean   : 1.2135
##                          3rd Qu.: 1.8559
##                          Max.   : 4.1847
##
## $`3`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     :9691     Min.   :-2.9821   Min.   :-1.27977
##  Manhattan    :   0     1st Qu.:-0.9275   1st Qu.:-0.04326
##  Queens       :3276     Median :-0.6285   Median : 0.35290
##  Staten Island:   0     Mean   :-0.6363   Mean   : 0.72064
##  Bronx        :   0     3rd Qu.:-0.3029   3rd Qu.: 0.96298
##                         Max.   : 0.9290   Max.   : 5.16264
##            room_type         price
##  Private room   :11690   Min.   :-1.3248
##  Entire home/apt:  800   1st Qu.:-0.9499
##  Shared room    :  477   Median :-0.8136
##                          Mean   :-0.7311
##                          3rd Qu.:-0.5864
##                          Max.   : 3.0487
##
## $`4`
##     neighbourhood_group    latitude        longitude
##  Brooklyn     :   0     Min.   :0.3207   Min.   :-0.681685
##  Manhattan    :5880     1st Qu.:1.0549   1st Qu.:-0.001084
##  Queens       :1236     Median :1.4428   Median : 0.206028
##  Staten Island:   0     Mean   :1.4808   Mean   : 0.365556
##  Bronx        :1059     3rd Qu.:1.8253   3rd Qu.: 0.551574
##                         Max.   :3.3632   Max.   : 3.731570
##            room_type        price
##  Private room   :5609   Min.   :-1.3135
##  Entire home/apt:2281   1st Qu.:-0.8250
##  Shared room    : 285   Median :-0.5864
##                         Mean   :-0.5373
##                         3rd Qu.:-0.3592
##                         Max.   : 2.0149
##
## $`5`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     :9126     Min.   :-2.9118   Min.   :-2.81358
##  Manhattan    :   0     1st Qu.:-1.0253   1st Qu.:-0.33830
##  Queens       : 806     Median :-0.7691   Median :-0.04618
##  Staten Island:  10     Mean   :-0.7513   Mean   :-0.00574
##  Bronx        :   0     3rd Qu.:-0.3492   3rd Qu.: 0.30683
##                         Max.   : 0.7754   Max.   : 4.68028
##            room_type        price
##  Private room   : 294   Min.   :-1.32482
##  Entire home/apt:9563   1st Qu.:-0.35924
##  Shared room    :  85   Median : 0.08379
##                         Mean   : 0.18185
##                         3rd Qu.: 0.54954
```

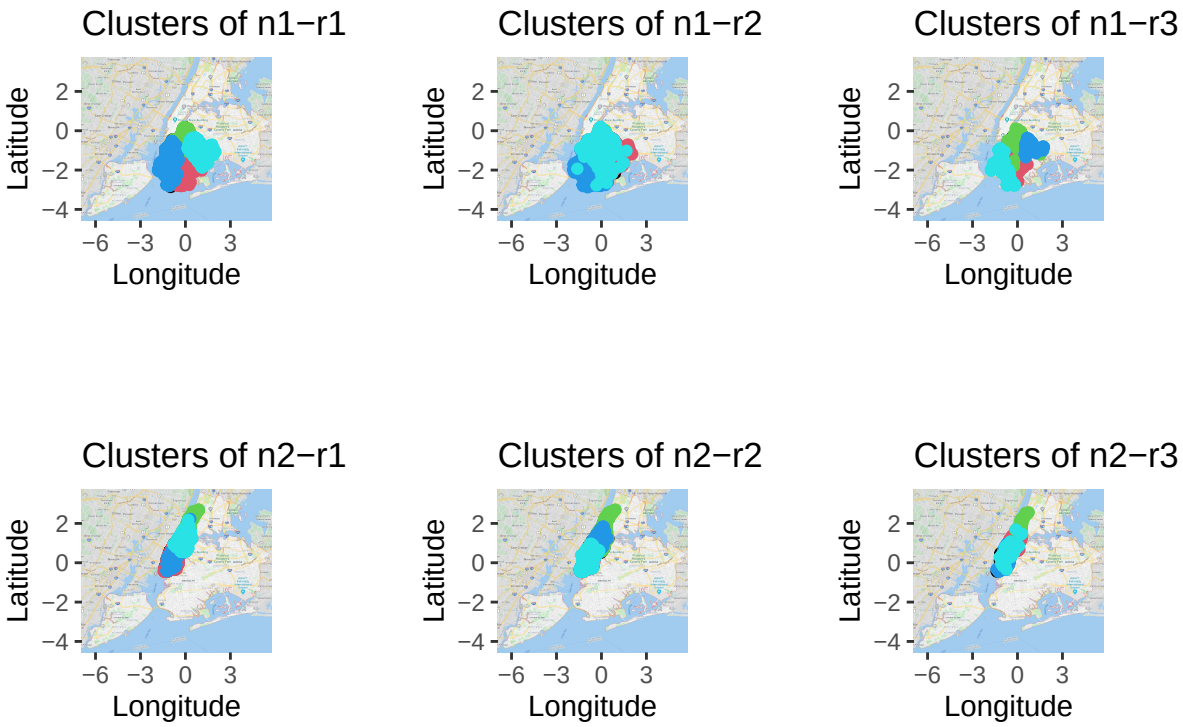
```
##                          Max.   : 4.18465
```

```
multiplots(all_cluster$all_plot[1:5], cols=3)
```

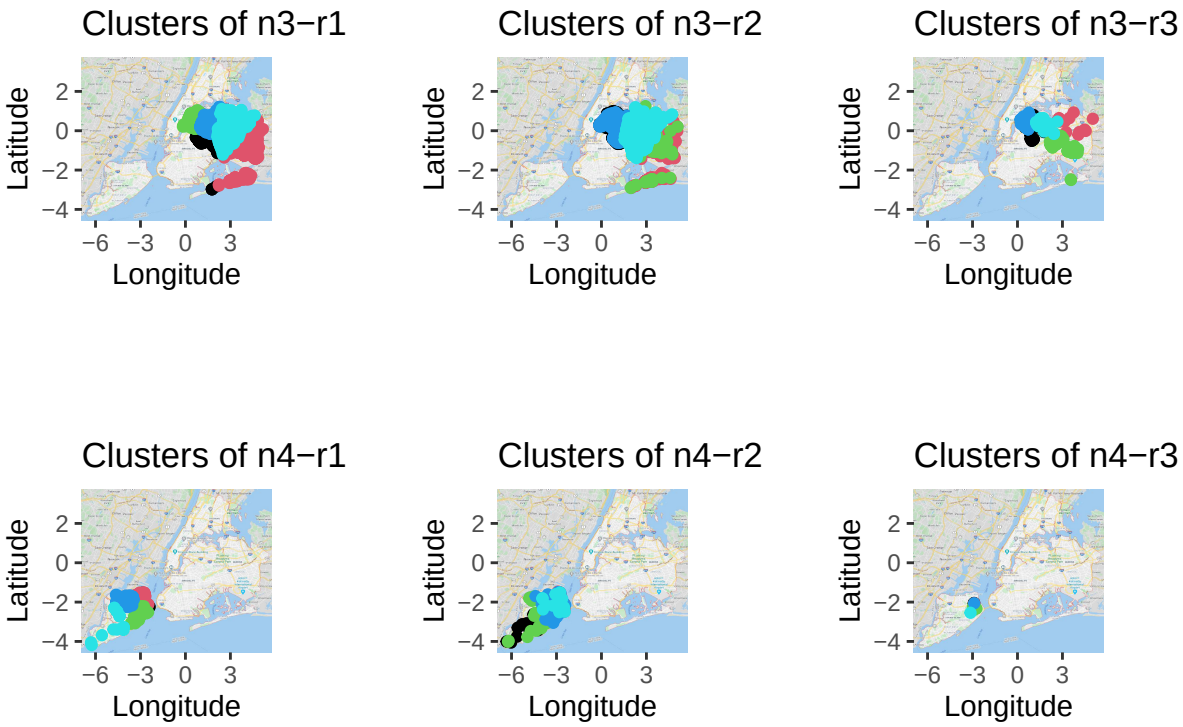
```
## Loading required package: grid
```



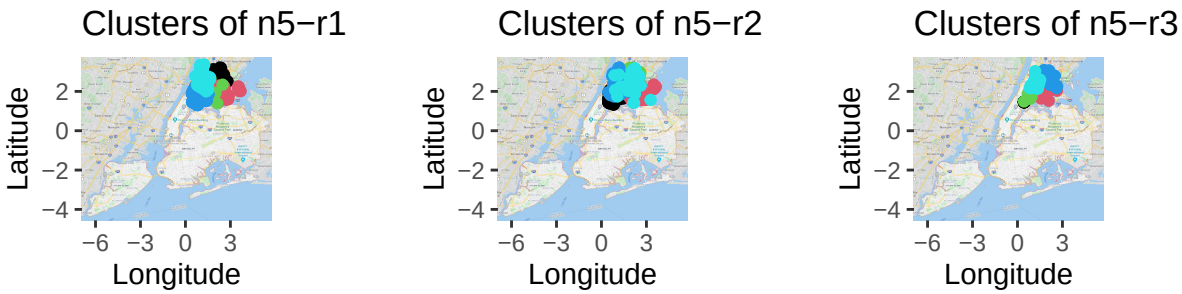

```
multiplots(all_cluster$all_plot[6:11], cols=3)
```




```
multiplots(all_cluster$all_plot[12:17], cols=3)
```




```
multiplots(all_cluster$all_plot[c(18:20,22,23)], cols=3)
```




```
## NULL
## NULL
```

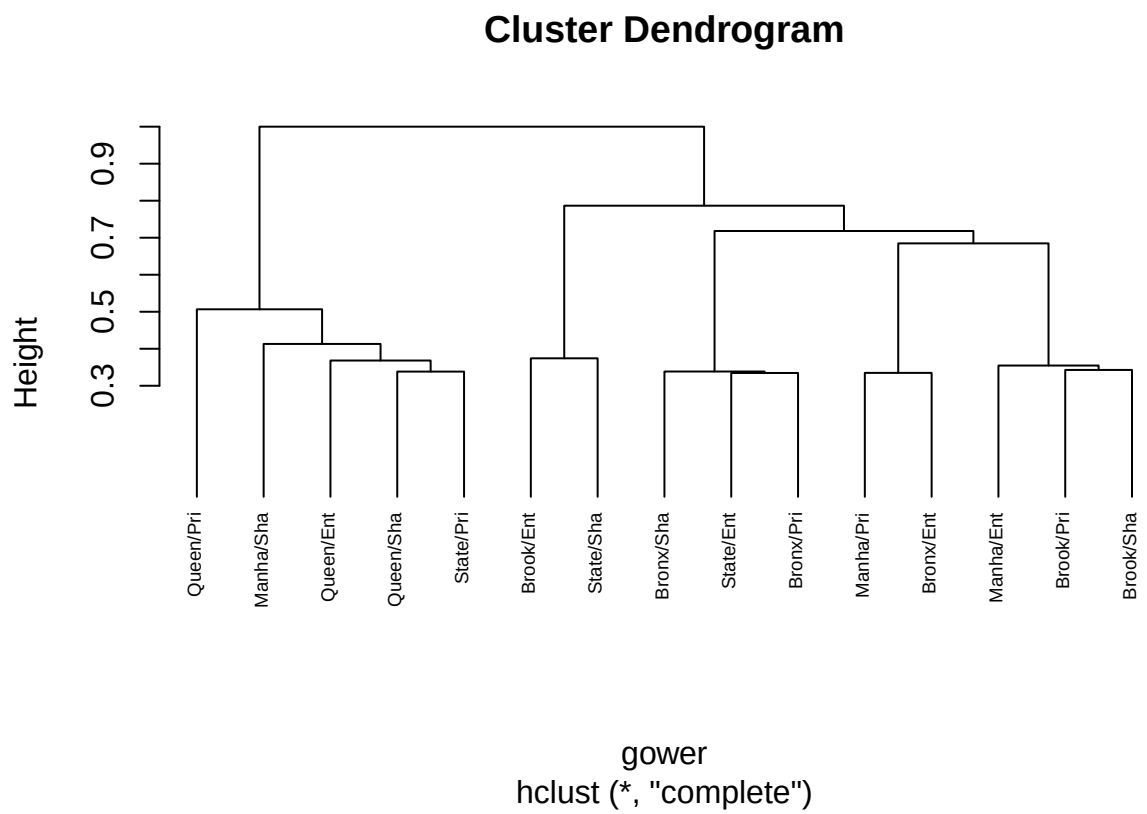
Hierarchical Cluster Analysis

```
agg = aggregate(price ~neighbourhood_group+room_type, clust_data$all , mean)


name_hc = c()
for (n1 in substr(unique(agg$neighbourhood_group),1,5))
{
  for(n2 in substr(unique(agg$room_type),1,3))
  {
    name_hc = c(name_hc, paste0(n1,"/",n2))
  }
}
rownames(agg) = name_hc

agg
```

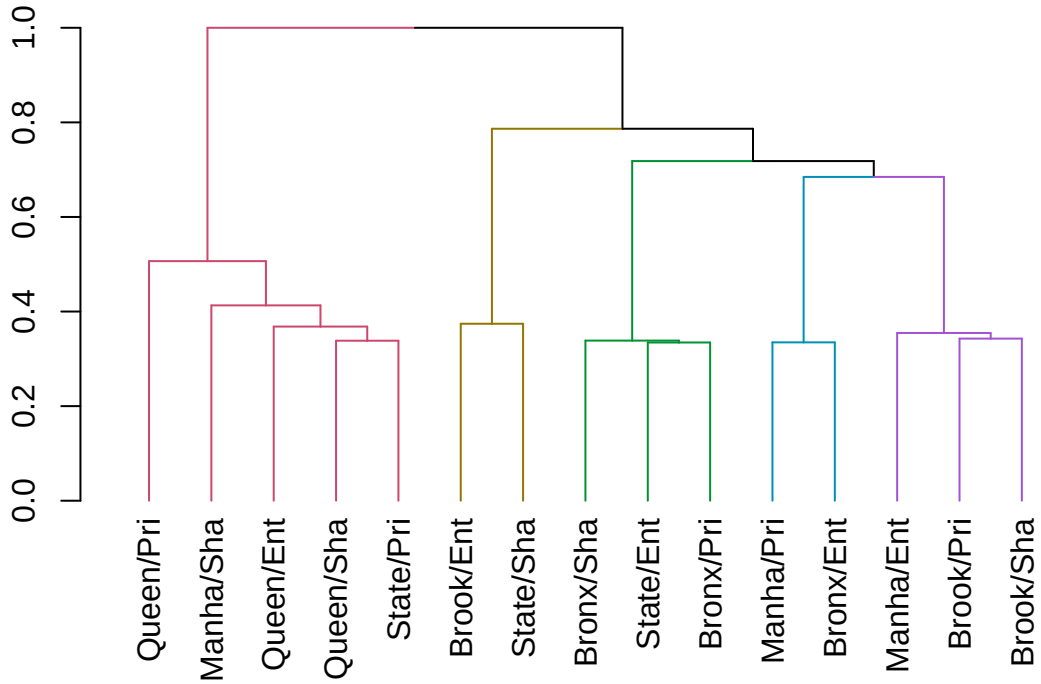
```
##           neighbourhood_group       room_type       price
## Brook/Pri            Brooklyn    Private room -0.68331638
## Brook/Ent           Manhattan    Private room -0.31844499
## Brook/Sha              Queens    Private room -0.73368292
```

```
## Manha/Pri       Staten Island    Private room -0.78758723
## Manha/Ent              Bronx    Private room -0.79719615
## Manha/Sha           Brooklyn Entire home/apt  0.32170945
## Queen/Pri          Manhattan Entire home/apt  0.82148490
## Queen/Ent             Queens Entire home/apt  0.08237678
## Queen/Sha       Staten Island Entire home/apt -0.07711867
## State/Pri              Bronx Entire home/apt -0.10379804
## State/Ent           Brooklyn     Shared room -0.93711857
## State/Sha          Manhattan     Shared room -0.53647655
## Bronx/Pri             Queens     Shared room -0.93058224
## Bronx/Ent       Staten Island     Shared room -0.77955083
## Bronx/Sha              Bronx     Shared room -0.95841842
```

```r
gower <- daisy(agg, metric = "gower")
hc1 <- hclust(gower, method = "complete" )

plot(hc1, cex = 0.6, hang = -1)
```



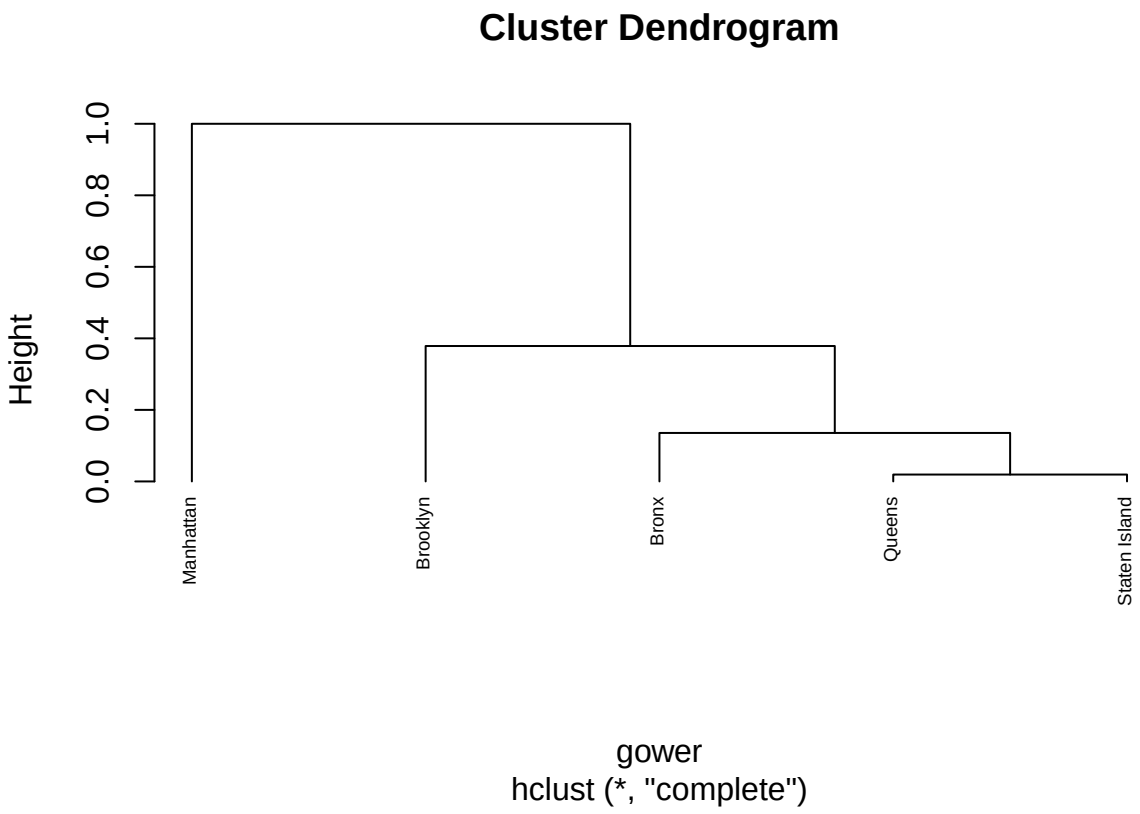

```r
avg_dend_obj <- as.dendrogram(hc1)
avg_col_dend <- color_branches(avg_dend_obj, h = 0.6)
plot(avg_col_dend)
```




```
agg = aggregate(price ~neighbourhood_group, clust_data$all , mean)
agg
```

```
##   neighbourhood_group      price
## 1            Brooklyn -0.2147398
## 2           Manhattan  0.3601591
## 3              Queens -0.4396243
## 4       Staten Island -0.4574125
## 5               Bronx -0.5650283
```

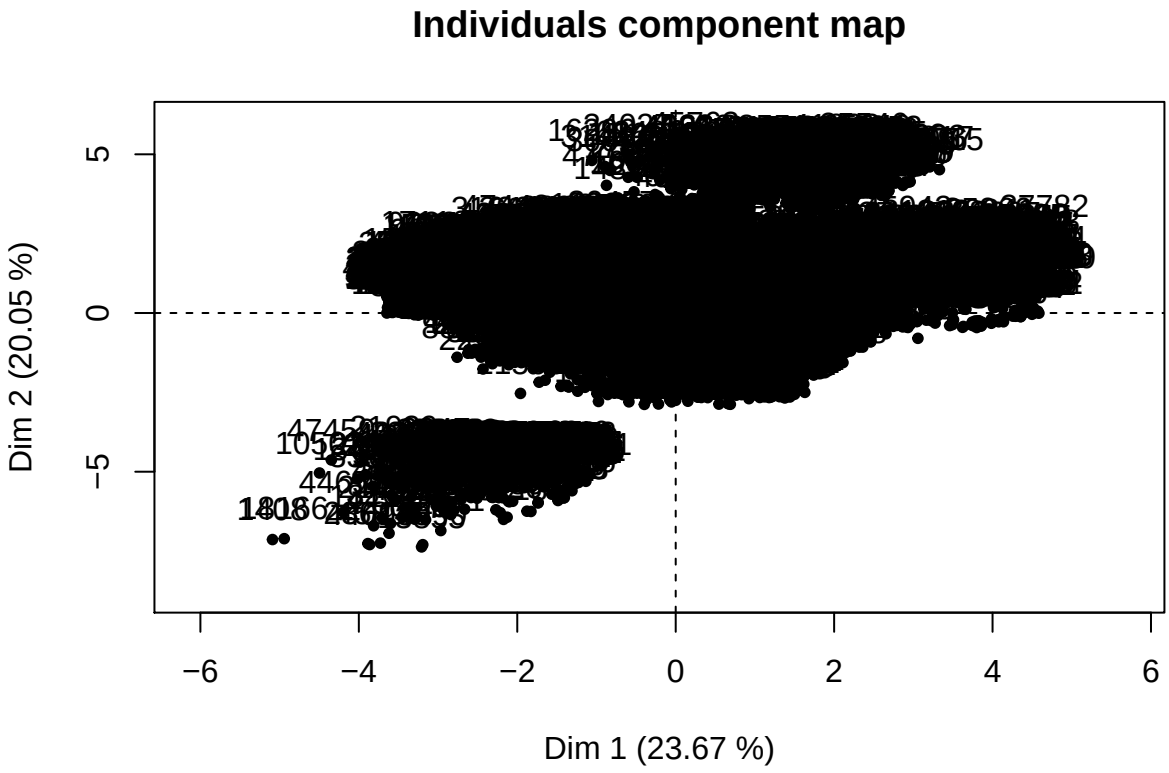
```
rownames(agg) = c("Brooklyn","Manhattan",
                  "Queens","Staten Island", "Bronx")
agg$neighbourhood_group = NULL
gower <- daisy(agg, metric = "gower")
hc1 <- hclust(gower, method = "complete" )

plot(hc1, cex = 0.6, hang = -1)
```

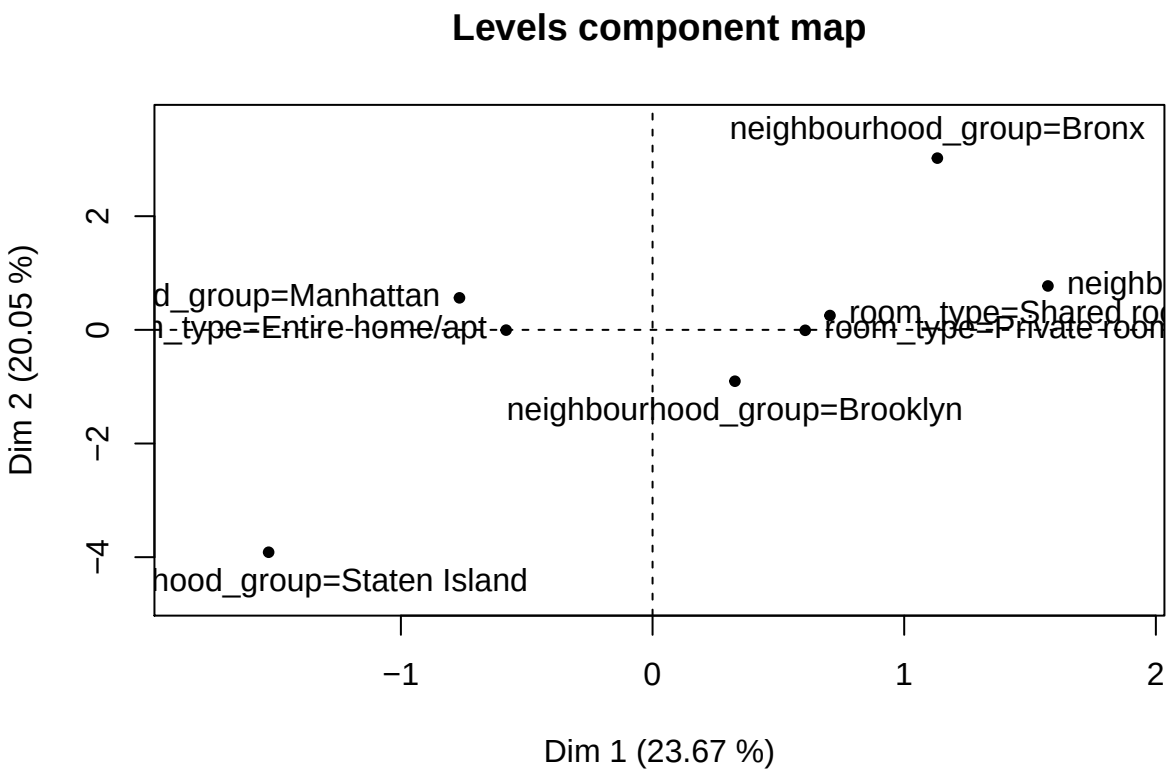


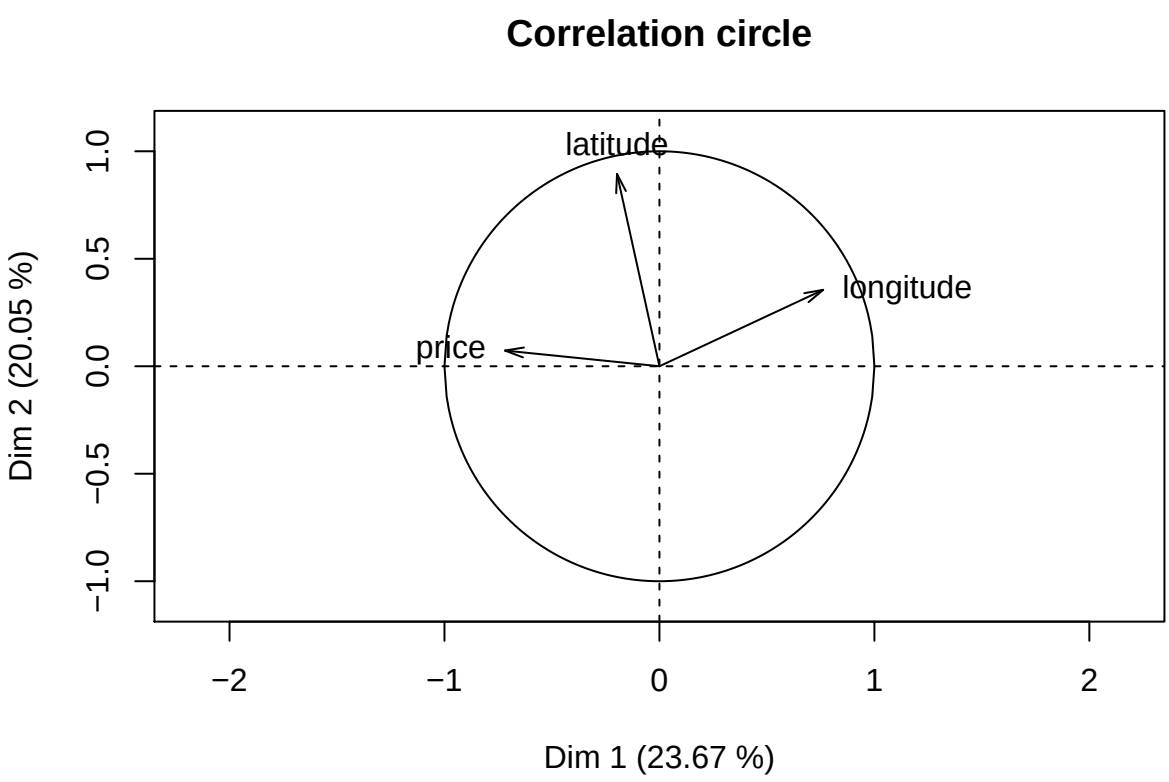
Principal Component Analysis

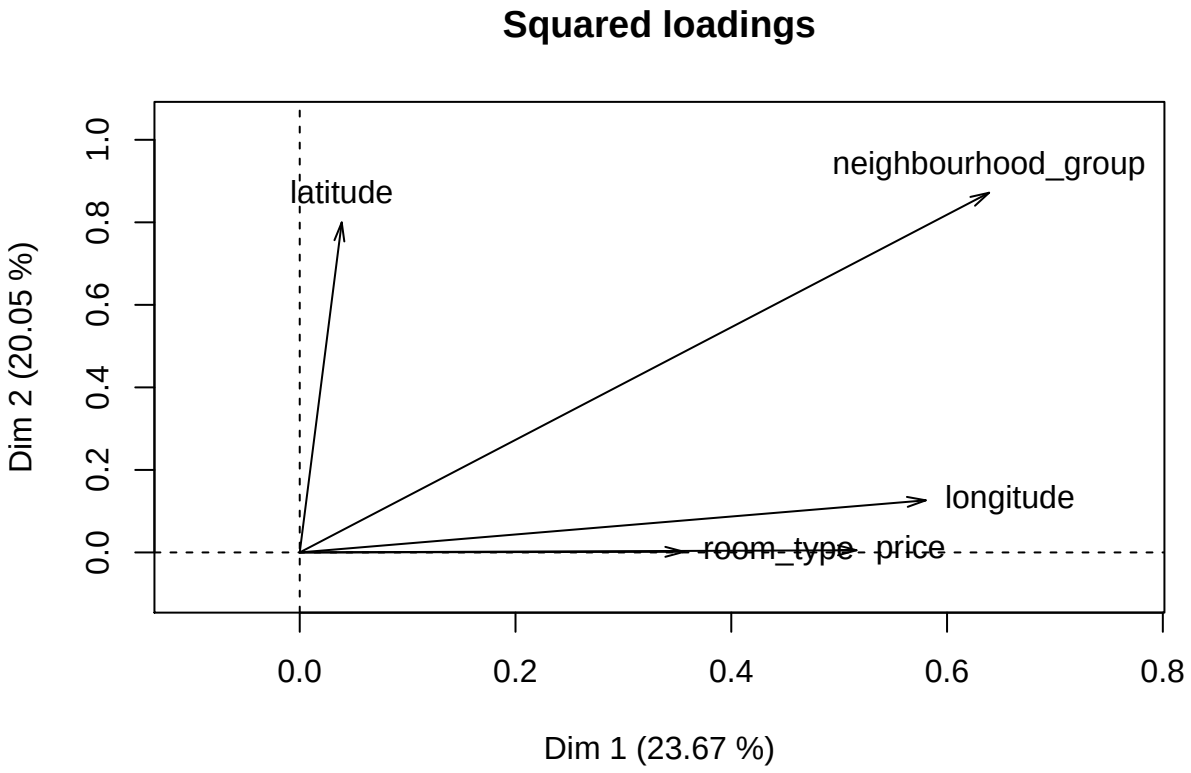
PCAmixdata

```
## Split mixed dataset into quantitative and qualitative variables
#split <- splitmix(dataset[1:5])

split = splitmix(clust_data$all)
## PCA
res.pcamix <- PCAmix(X.quanti=split$X.quanti,
                     X.quali=split$X.quali,
                     rename.level=TRUE)
```


Individuals component map
Dim 2 (20.05 %)
Dim 1 (23.67 %)
5
0
−5
−6
−4
−2
0
2
4
6


Levels component map
neighbourhood_group=Bronx
neighb
d_group=Manhattan
room_type=Shared ro
n_type=Entire home/apt
room_type=Private room
neighbourhood_group=Brooklyn
hood_group=Staten Island
Dim 2 (20.05 %)
2
0
−2
−4
−1
0
1
2
Dim 1 (23.67 %)


Correlation circle
latitude
longitude
price
Dim 2 (20.05 %)
Dim 1 (23.67 %)
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2




```
res.pcamix
```

```
##
## Call:
## PCAmix(X.quanti = split$X.quanti, X.quali = split$X.quali, rename.level = TRUE)
##
## Method = Principal Component of mixed data (PCAmix)
##
##
## "name" "description"
## "$eig" "eigenvalues of the principal components (PC) "
## "$ind" "results for the individuals (coord,contrib,cos2)"
## "$quanti" "results for the quantitative variables (coord,contrib,cos2)"
## "$levels" "results for the levels of the qualitative variables (coord,contrib,cos2)"
## "$quali" "results for the qualitative variables (contrib,relative contrib)"
## "$sqload" "squared loadings"
## "$coef" "coef of the linear combinations defining the PC"
```

```
## Inspect principal components
res.pcamix$eig
```

```
##       Eigenvalue Proportion Cumulative
## dim 1  2.1303801  23.670890   23.67089
## dim 2  1.8046238  20.051375   43.72227
## dim 3  1.2456603  13.840670   57.56294
## dim 4  1.0028761  11.143067   68.70600
## dim 5  0.9971611  11.079568   79.78557
```

```
## dim 6  0.9570021  10.633357   90.41893
## dim 7  0.4201255   4.668061   95.08699
## dim 8  0.2652652   2.947391   98.03438
## dim 9  0.1769058   1.965620  100.00000
```

```
res.pcamix$quanti.cor
```

```
##                dim 1      dim 2      dim 3       dim 4       dim 5
## latitude  -0.1972875 0.89428404 -0.2214475 -0.04100129 -0.04272996
## longitude  0.7617791 0.35537824  0.4064136  0.01097103  0.01298250
## price     -0.7183816 0.07289152  0.4855170  0.02253704  0.01357485
```

```
res.pcamix$quali.eta2
```

```
##                          dim 1       dim 2     dim 3     dim 4     dim 5
## neighbourhood_group 0.6390071 0.871714919 0.4342793 0.5273305 0.4806940
## room_type           0.3560712 0.001558062 0.3614432 0.4732362 0.5142885
```

Factor Analysis of Mixed Data (FAMD)

FAMD (base, ncp = 5, sup.var = NULL, ind.sup = NULL, graph = TRUE) - base : a data frame with n rows (individuals) and p columns (variables). - ncp: the number of dimensions kept in the results (by default 5) - sup.var: a vector indicating the indexes of the supplementary variables. - ind.sup: a vector indicating the indexes of the supplementary individuals. - graph : a logical value. If TRUE a graph is displayed.

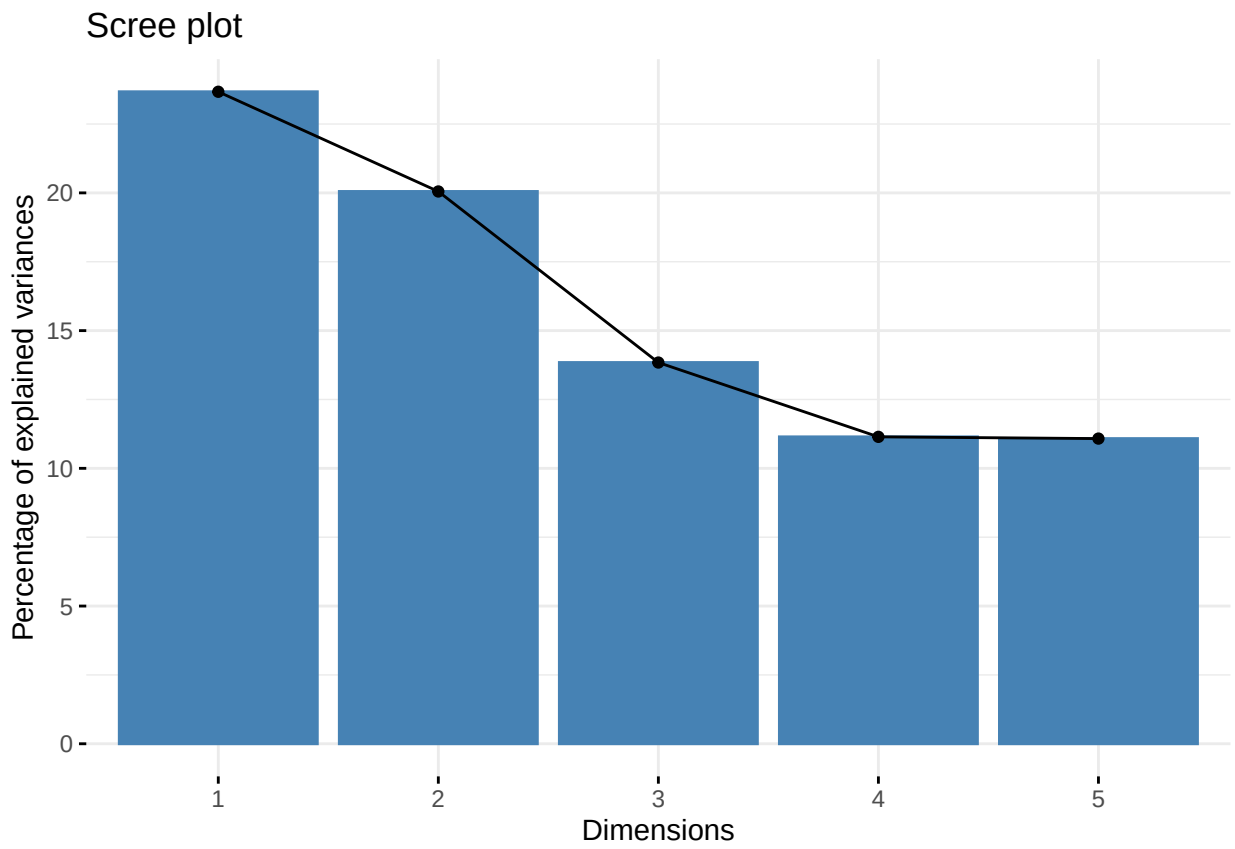
```
res.famd <- FAMD(clust_data$all, graph = F, ncp = 5)
print(res.famd)
```

```
## *The results are available in the following objects:
##
##   name          description
## 1 "$eig"        "eigenvalues and inertia"
## 2 "$var"        "Results for the variables"
## 3 "$ind"        "results for the individuals"
## 4 "$quali.var"  "Results for the qualitative variables"
## 5 "$quanti.var" "Results for the quantitative variables"
```

```
eig.val <- get_eigenvalue(res.famd)
head(eig.val)
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.1303801         23.67089                     23.67089
## Dim.2  1.8046238         20.05138                     43.72227
## Dim.3  1.2456603         13.84067                     57.56294
## Dim.4  1.0028761         11.14307                     68.70600
## Dim.5  0.9971611         11.07957                     79.78557
```

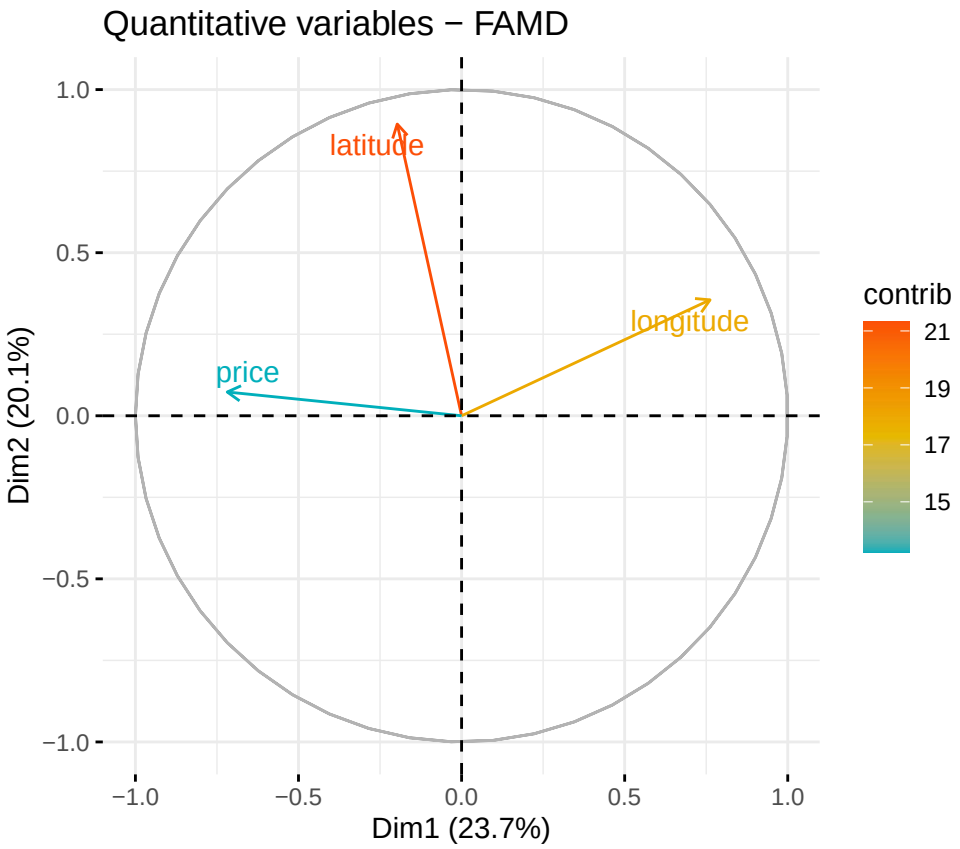
```
fviz_screeplot(res.famd)
```




```
quanti.var <- get_famd_var(res.famd, "quanti.var")
quanti.var
```

```
## FAMD results for quantitative variables
##  ==================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

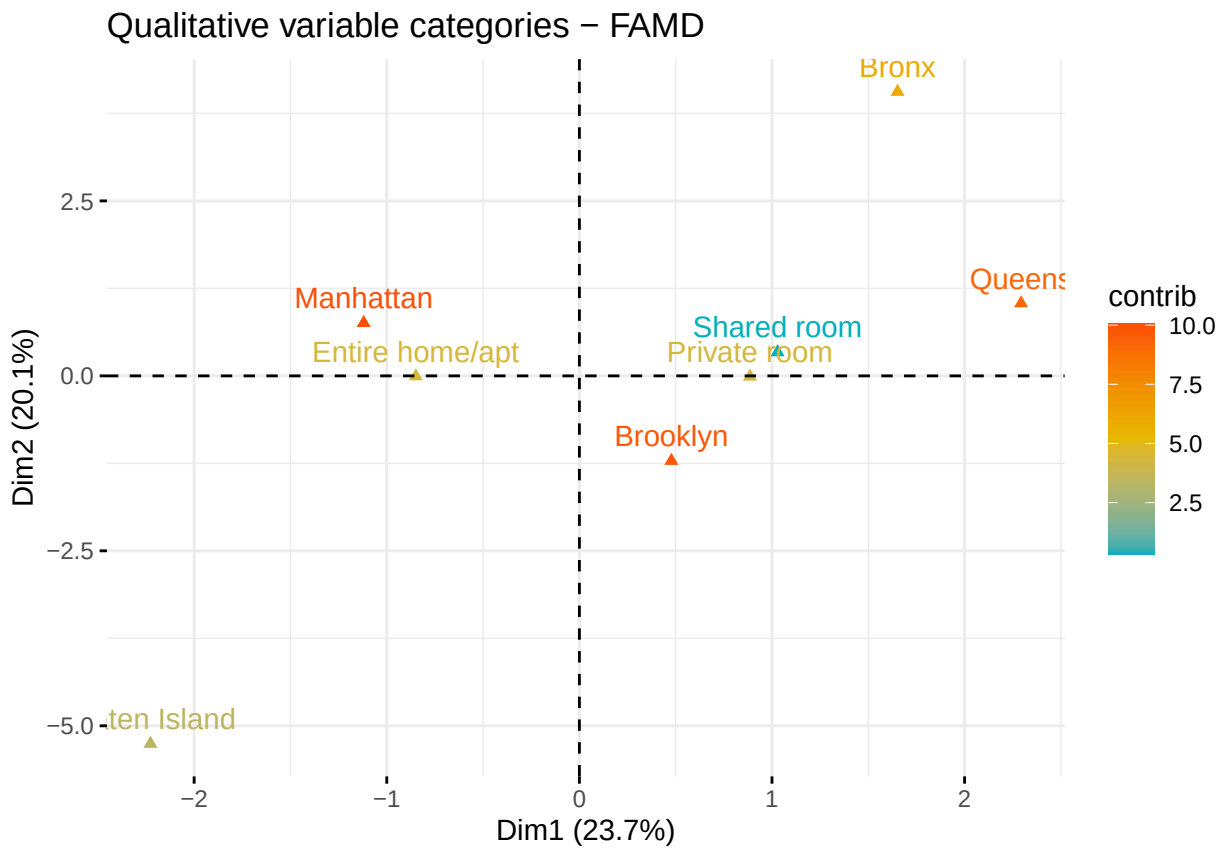
```
fviz_famd_var(res.famd, "quanti.var", col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
              repel = TRUE)
```




```
quali.var <- get_famd_var(res.famd, "quali.var")
quali.var
```

```
## FAMD results for qualitative variable categories
##  ==================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

```
fviz_famd_var(res.famd, "quali.var", col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07")
)
```




```
var <- get_famd_var(res.famd)
var
```

```
## FAMD results for variables
##  ===================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

```
# Coordinates of variables
head(var$coord)
```

```
##                          Dim.1       Dim.2      Dim.3        Dim.4        Dim.5
## latitude            0.03892236 0.799743946 0.04903898 0.0016811055 0.0018258497
## longitude           0.58030735 0.126293692 0.16517205 0.0001203636 0.0001685452
## price               0.51607212 0.005313173 0.23572672 0.0005079180 0.0001842764
## neighbourhood_group 0.63900709 0.871714919 0.43427934 0.5273304673 0.4806939587
## room_type           0.35607121 0.001558062 0.36144319 0.4732362202 0.5142884950
```

```
# Cos2: quality of representation on the factore map
head(var$cos2)
```

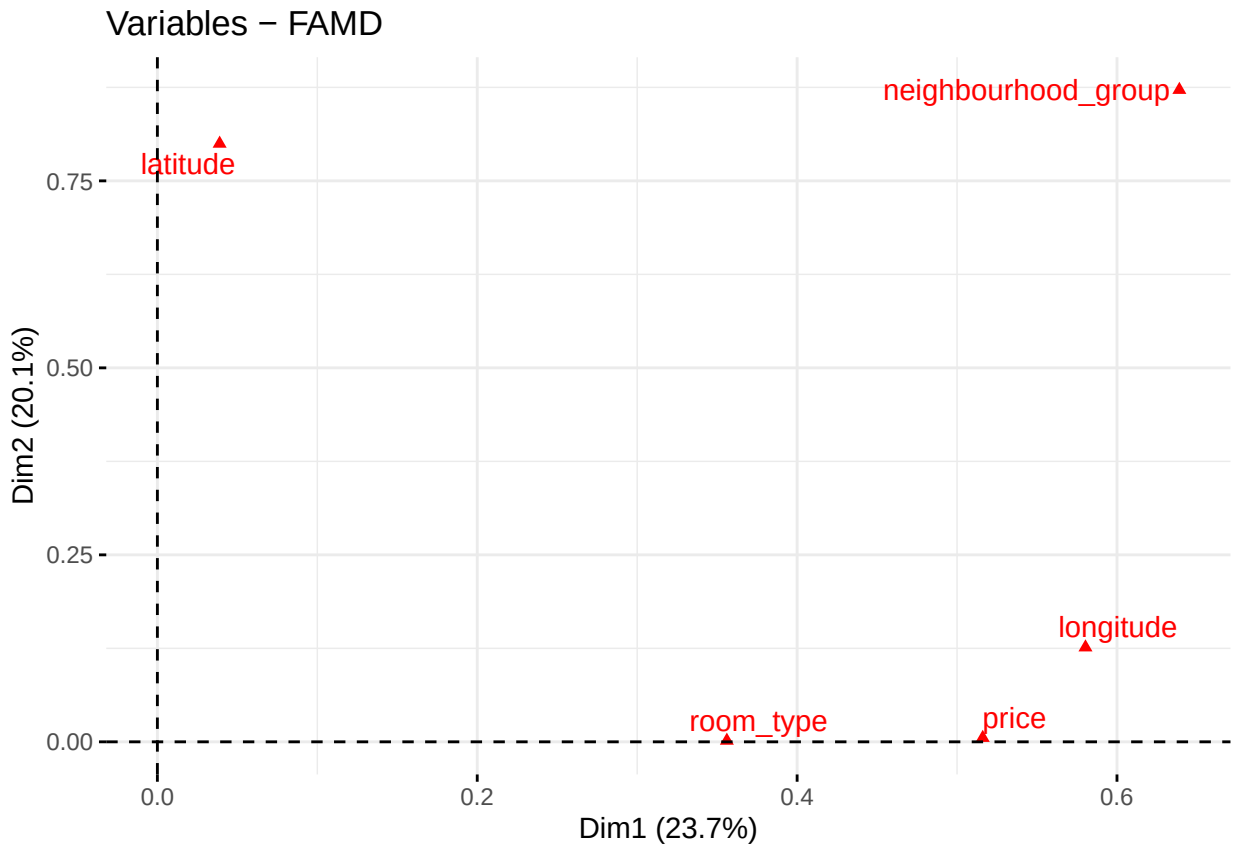
```
##                          Dim.1        Dim.2       Dim.3        Dim.4
## latitude            0.00151495 6.395904e-01 0.002404821 2.826116e-06
## longitude           0.33675663 1.595010e-02 0.027281805 1.448739e-08
## price               0.26633043 2.822981e-05 0.055567086 2.579807e-07
## neighbourhood_group 0.10208251 1.899717e-01 0.047149637 6.951936e-02
```

```
## room_type           0.06339335 1.213779e-06 0.065320588 1.119763e-01
##                            Dim.5
## latitude            3.333727e-06
## longitude           2.840749e-08
## price               3.395780e-08
## neighbourhood_group 5.776667e-02
## room_type           1.322463e-01
```

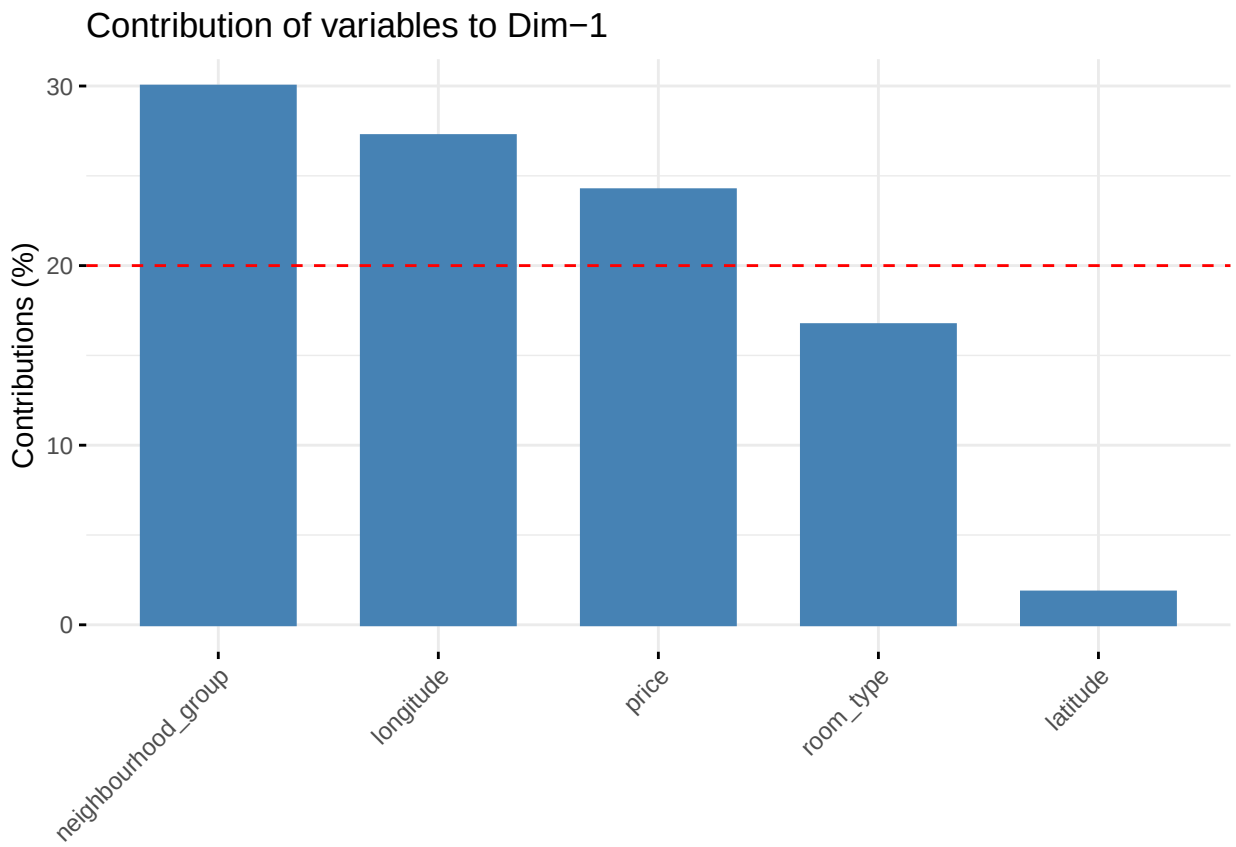
```
# Contributions to the  dimensions
head(var$contrib)
```

```
##                          Dim.1       Dim.2     Dim.3       Dim.4       Dim.5
## latitude              1.827015 44.31638047  3.936786  0.16762844  0.18310478
## longitude            27.239615  6.99833910 13.259799  0.01200184  0.01690250
## price                24.224415  0.29441998 18.923837  0.05064614  0.01848011
## neighbourhood_group 29.994980 48.30452324 34.863386 52.58181750 48.20624738
## room_type            16.713975  0.08633722 29.016193 47.18790608 51.57526523
```

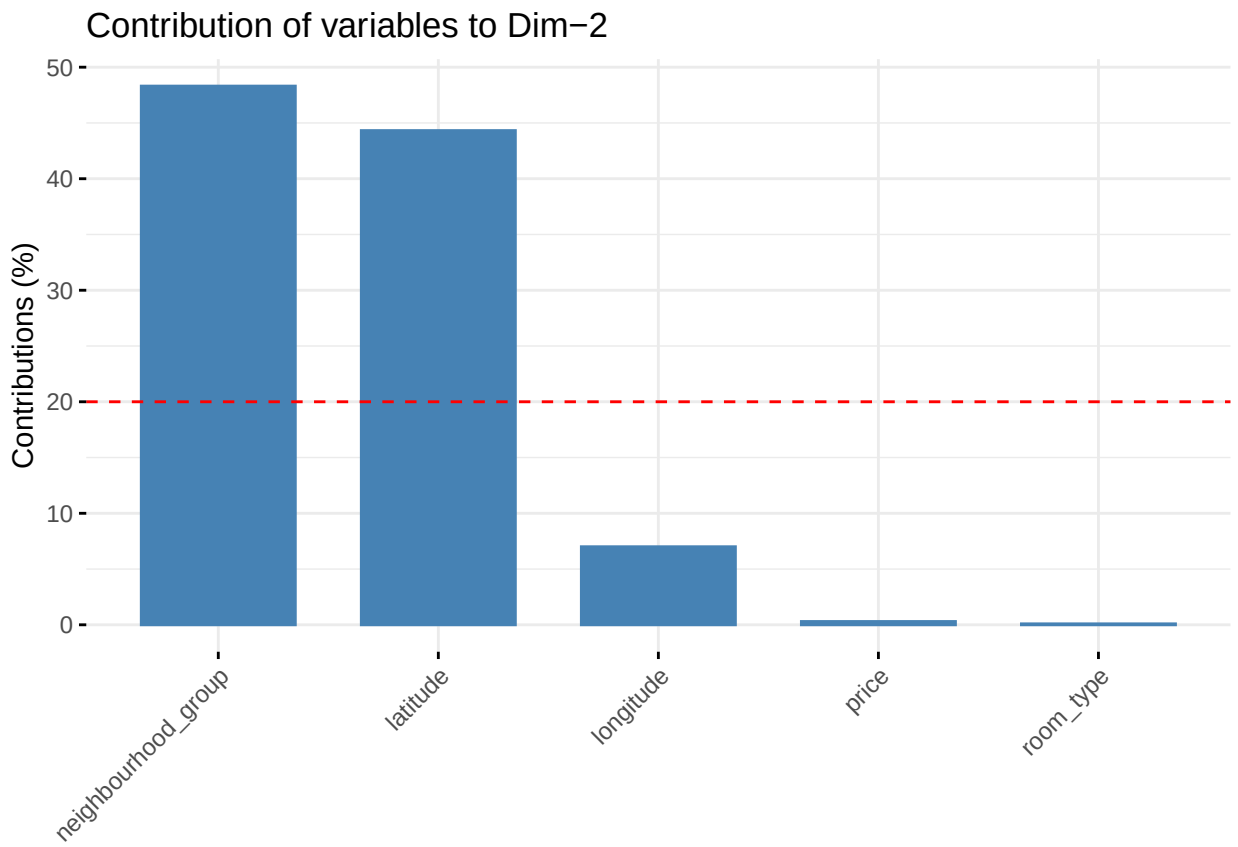
```
# Plot of variables

fviz_famd_var(res.famd, repel = TRUE)
```



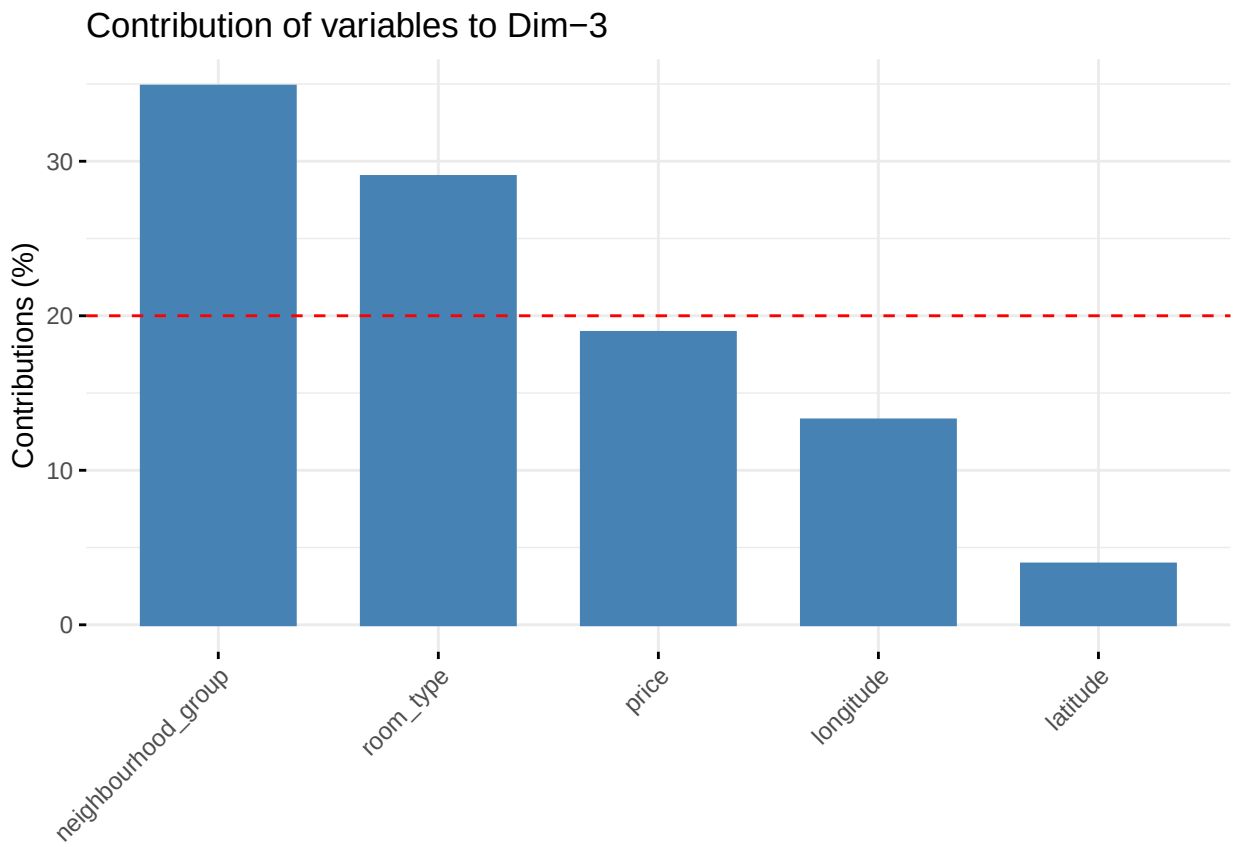

```
# Contribution to the first dimension
fviz_contrib(res.famd, "var", axes = 1)
```



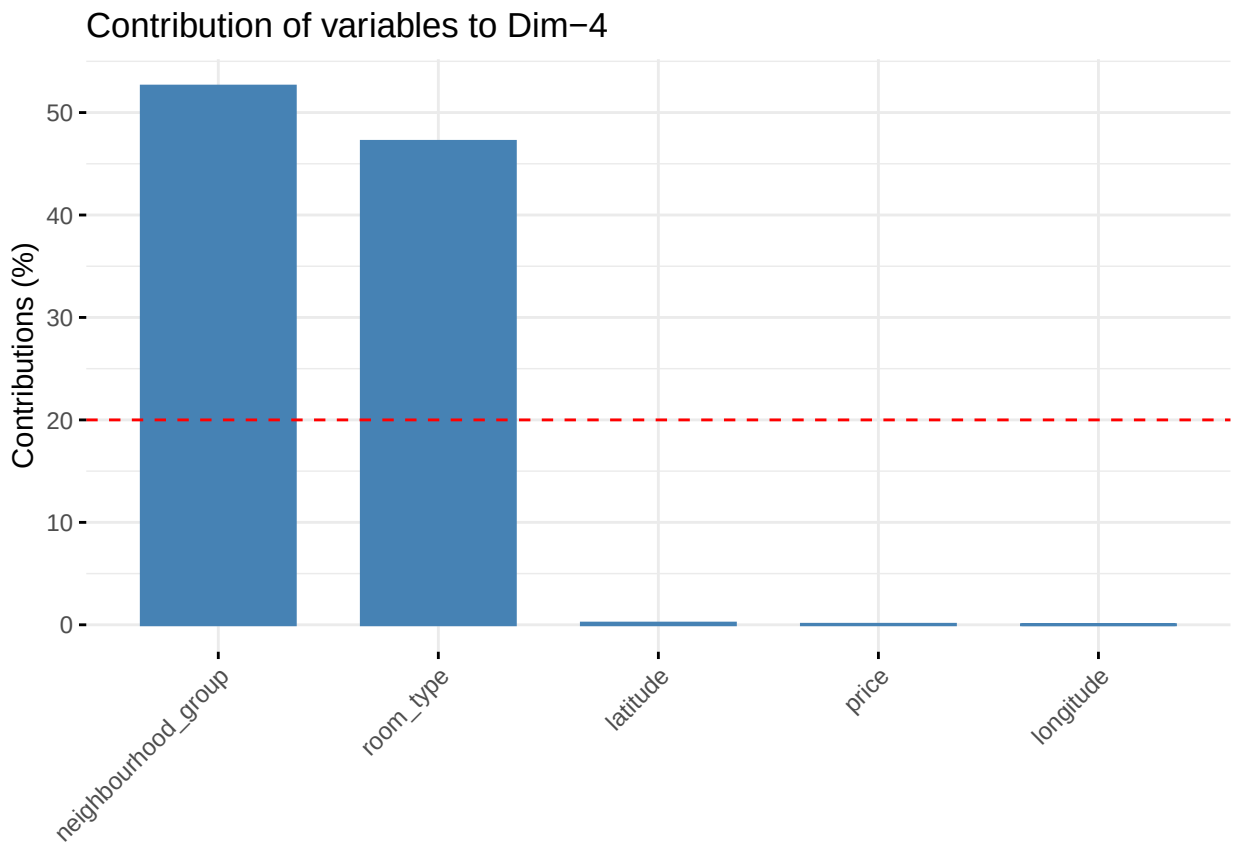

```
# Contribution to the second dimension
fviz_contrib(res.famd, "var", axes = 2)
```



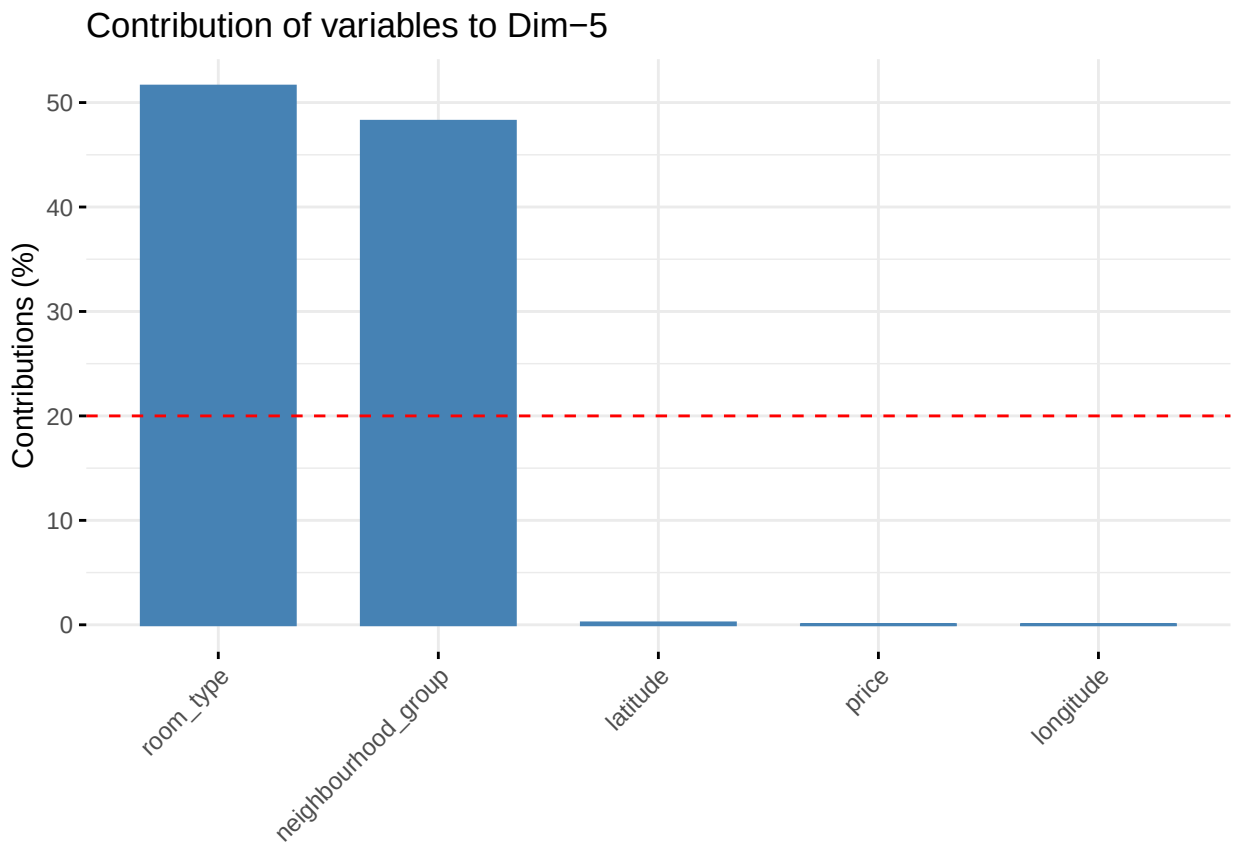

```
# Contribution to the third dimension
fviz_contrib(res.famd, "var", axes = 3)
```




```
# Contribution to the forth dimension
fviz_contrib(res.famd, "var", axes = 4)
```




```
# Contribution to the fifth dimension
fviz_contrib(res.famd, "var", axes = 5)
```


Contribution of variables to Dim−5
Contributions (%)
50
40
30
20
10
0
room_type
neighbourhood_group
latitude
price
longitude