Statistical Learning Project

Andrea Ierardi

pdf_document: default html_document: default

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 here

```
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
```

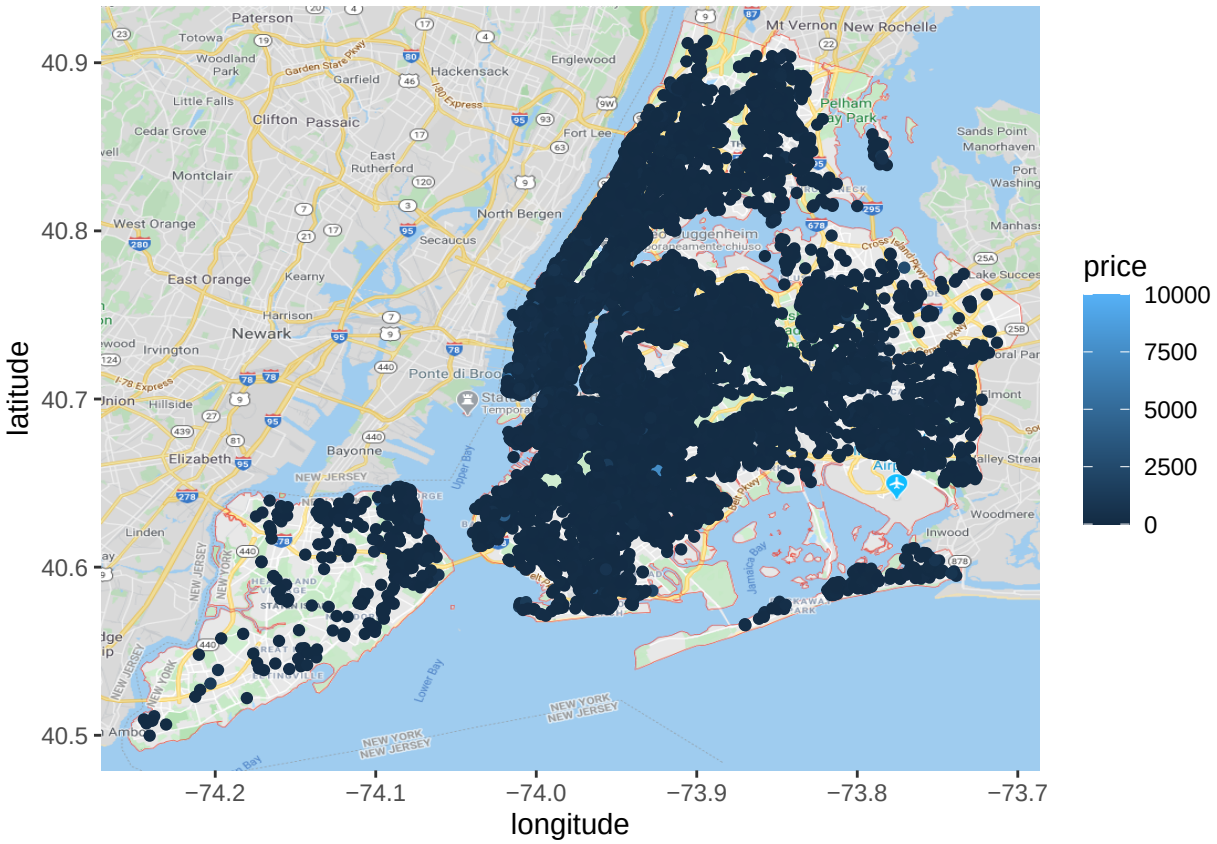
```
summary(ds)
```

```
##        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("Dimensions of the dataset:" ,dim(ds))
```

```
## Dimensions 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
```

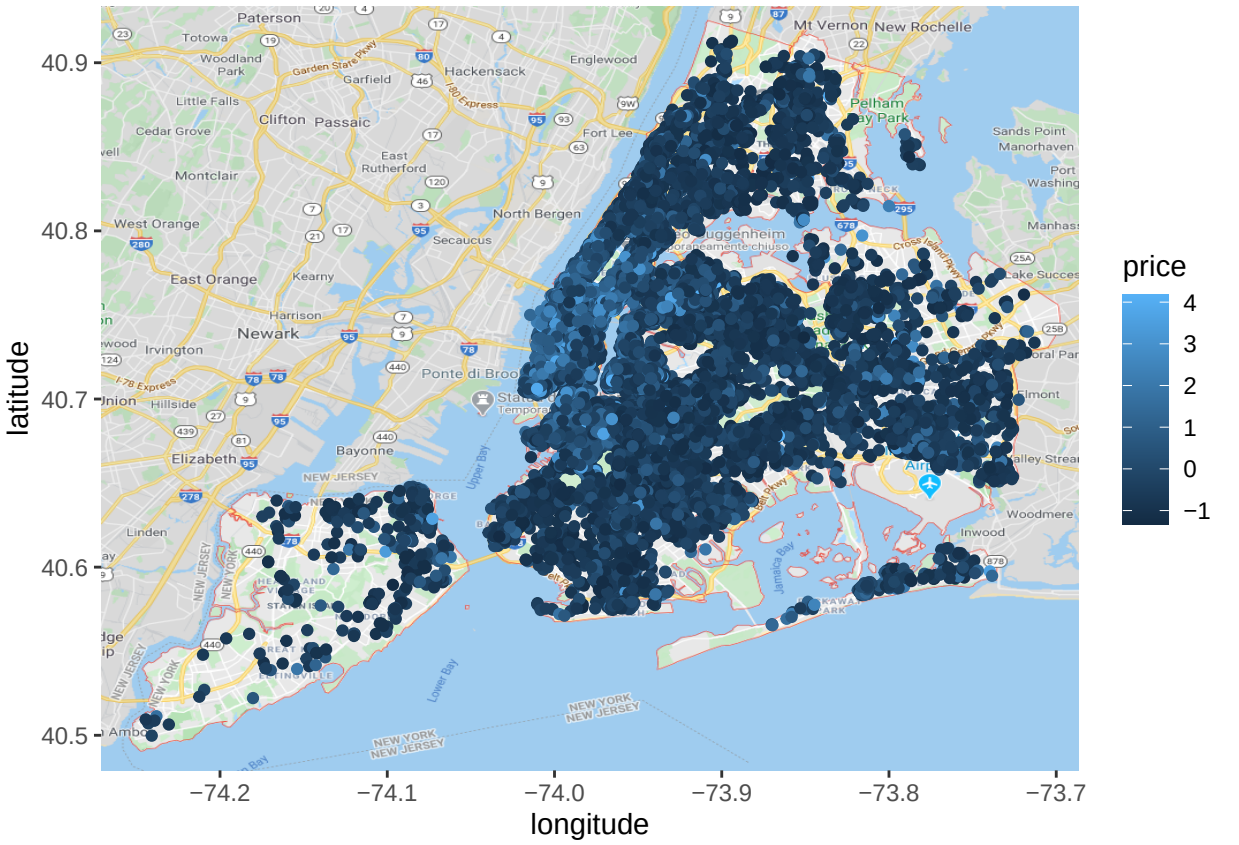
```
## 1                   1 40.64749 -73.97237         1  0.1973858
## 2                   2 40.75362 -73.98377         2  1.0607252
## 3                   2 40.80902 -73.94190         1  0.2087456
## 4                   1 40.68514 -73.95976         2 -0.4841979
## 5                   2 40.79851 -73.94399         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]
  # print(clust_data[[n]]$room_type )
  clust_data[[n]]$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))
  #lin_reg[[sub]]$plt=   plot(lm.fit)
  cat("\n\n")
}
```

```
## [1] "========== 1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8345 -0.3464 -0.1178  0.1731  4.9573
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -548.77851   20.08969 -27.316  < 2e-16 ***
## latitude       5.37557    0.20584  26.115  < 2e-16 ***
## longitude     -4.45432    0.22293 -19.980  < 2e-16 ***
## room_type2     0.97147    0.01122  86.566  < 2e-16 ***
## room_type3    -0.17381    0.03864  -4.499 6.89e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6676 on 14888 degrees of freedom
## Multiple R-squared:  0.3904, Adjusted R-squared:  0.3902
## F-statistic:  2383 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
## -2.3997 -0.5291 -0.1913  0.2865  4.7705
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -843.16382   58.37960 -14.443  < 2e-16 ***
## latitude      -0.36939    0.35427  -1.043    0.297
## longitude    -11.59833    0.61630 -18.819  < 2e-16 ***
## room_type2     1.02020    0.01497  68.163  < 2e-16 ***
## room_type3    -0.22761    0.04842  -4.701 2.62e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.88 on 15652 degrees of freedom
## Multiple R-squared:  0.3338, Adjusted R-squared:  0.3336
## F-statistic:  1961 on 4 and 15652 DF,  p-value: < 2.2e-16
##
##
##
## [1] "========== 3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3764 -0.2972 -0.1184  0.1345  5.0659
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.68623   12.19323  -1.450  0.14699
## latitude     -0.34810    0.27817  -1.251  0.21086
## longitude    -0.42135    0.19522  -2.158  0.03096 *
## room_type2    0.81239    0.01920  42.319  < 2e-16 ***
## room_type3   -0.15461    0.05031  -3.073  0.00213 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.593 on 4219 degrees of freedom
## Multiple R-squared:  0.3106, Adjusted R-squared:  0.3099
## F-statistic: 475.2 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
## -0.6346 -0.2358 -0.0916  0.1118  5.1399
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -360.5784    19.4062  -18.58   <2e-16 ***
## latitude       2.7783     0.1990   13.96   <2e-16 ***
## longitude     -3.3383     0.2114  -15.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.422 on 6721 degrees of freedom
## Multiple R-squared:  0.04984,    Adjusted R-squared:  0.04956
## F-statistic: 176.3 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
## -1.9301 -0.5690 -0.2020  0.2865  4.4665
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -756.5694    39.3500  -19.23   <2e-16 ***
## latitude       7.5474     0.4052   18.63   <2e-16 ***
## longitude     -6.0823     0.4433  -13.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8619 on 6239 degrees of freedom
## Multiple R-squared:  0.06807,    Adjusted R-squared:  0.06777
## F-statistic: 227.8 on 2 and 6239 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
## -0.5419 -0.2240 -0.0917  0.1024  4.9175
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.2460    12.3016  -1.646   0.0999 .
## latitude     -0.1631     0.2852  -0.572   0.5674
## longitude    -0.3540     0.1887  -1.876   0.0608 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4257 on 2239 degrees of freedom
## Multiple R-squared:  0.001704,   Adjusted R-squared:  0.0008119
## F-statistic: 1.911 on 2 and 2239 DF,  p-value: 0.1482
##
##
##
## [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
## -0.4119 -0.1682 -0.1020  0.1206  1.6368
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.4977    46.3031   0.162   0.8716
## latitude      2.0226     1.0411   1.943   0.0543 .
## longitude     1.2298     0.6844   1.797   0.0748 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3313 on 125 degrees of freedom
## Multiple R-squared:  0.03576,    Adjusted R-squared:  0.02033
## F-statistic: 2.318 on 2 and 125 DF,  p-value: 0.1027
##
##
##
## [1] "========== n4-r1 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5556 -0.2506 -0.1124  0.1700  1.5552
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -144.407    117.860  -1.225    0.223
## latitude       1.311      1.245   1.054    0.294
## longitude     -1.219      1.185  -1.029    0.305
##
## Residual standard error: 0.3531 on 122 degrees of freedom
## Multiple R-squared:  0.01247,    Adjusted R-squared:  -0.003723
## F-statistic:  0.77 on 2 and 122 DF,  p-value: 0.4652
```

```
##
##
##
## [1] "========== n4-r2 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7905 -0.4979 -0.2084  0.3115  2.0357
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  135.664    232.330   0.584    0.560
## latitude      -1.502      2.369  -0.634    0.527
## longitude      1.009      2.113   0.478    0.634
##
## Residual standard error: 0.6894 on 110 degrees of freedom
## Multiple R-squared:  0.003687,   Adjusted R-squared:  -0.01443
## F-statistic: 0.2035 on 2 and 110 DF,  p-value: 0.8161
##
##
##
## [1] "========== n4-r3 =========="
##
## Call:
## lm(formula = price ~ ., data = trains[[sub]])
##
## Residuals:
##         2         3         5         6         8
## -0.091989 -0.392716  0.499432 -0.008522 -0.006204
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3437.87    4707.43  -0.730    0.541
## latitude       48.28      24.56   1.966    0.188
## longitude     -19.93      64.79  -0.308    0.787
##
## Residual standard error: 0.454 on 2 degrees of freedom
## Multiple R-squared:  0.6857, Adjusted R-squared:  0.3713
## F-statistic: 2.181 on 2 and 2 DF,  p-value: 0.3143
##
##
##
## [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
## -1.1884 -0.5191 -0.2658  0.1793  4.2156
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -53.3410   175.2402  -0.304    0.761
## latitude      1.6975     2.1824   0.778    0.437
## longitude     0.2177     1.6531   0.132    0.895
##
## Residual standard error: 0.9039 on 248 degrees of freedom
## Multiple R-squared:  0.003215,   Adjusted R-squared:  -0.004823
## F-statistic:   0.4 on 2 and 248 DF,  p-value: 0.6708
##
##
##
## [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
```

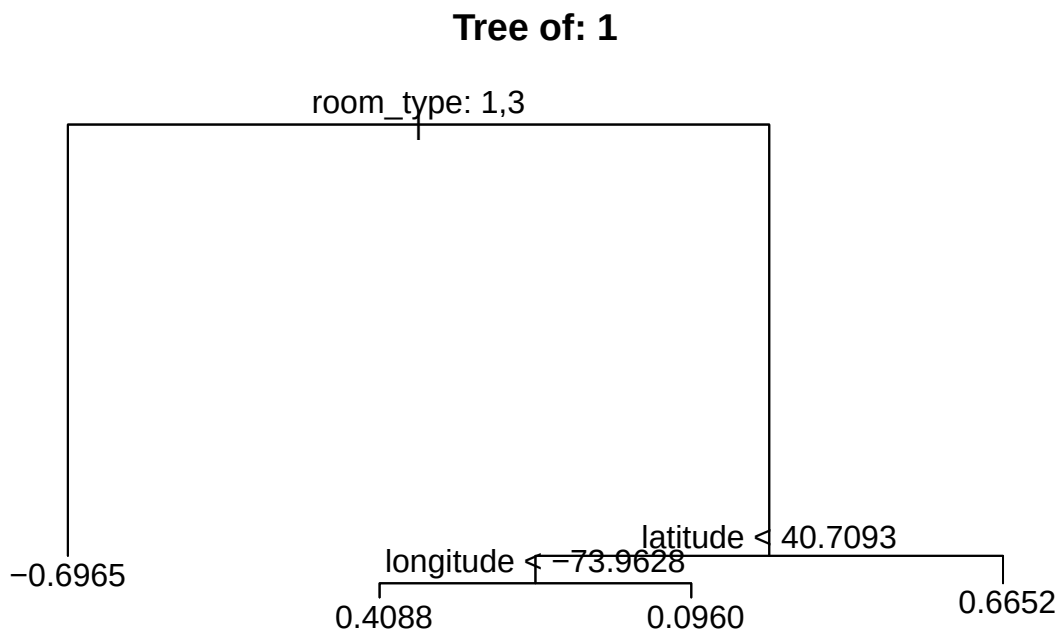
```
lin_reg$name = "Linear Regression"
model_lis$linear_regression=  lin_reg
```

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
```

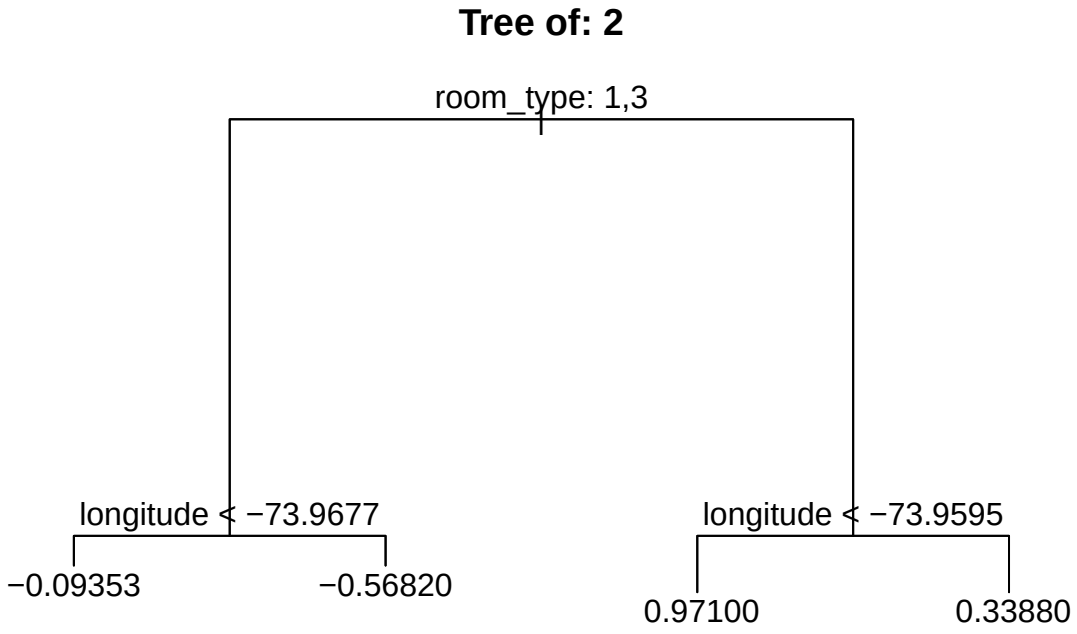
```
  {
    cat("Not possible to plot tree: ", sub)
  }

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

  dec_tree[[sub]]$MSE = mse =  sum((pred - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  #dec_tree[[sub]]$plt=   plot(tree_res)+text(tree_res,pretty=0)
  print(mse)
  cat("\n\n")
}
```

```
## [1] "========== 1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.4488 = 6683 / 14890
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9330 -0.3443 -0.1171  0.0000  0.1669  4.8810
```

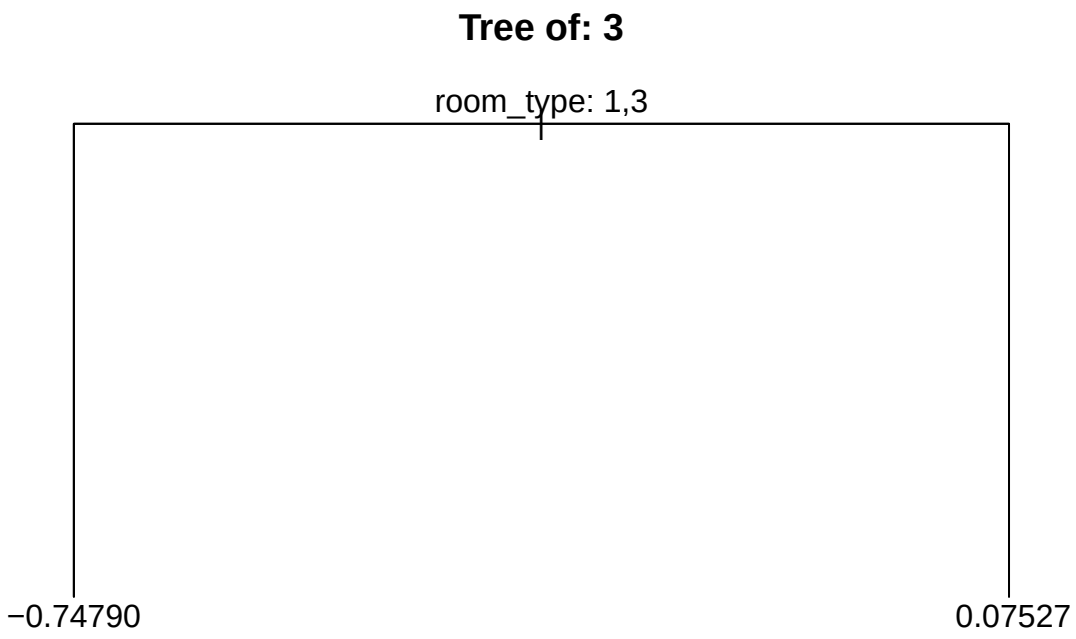



```
## [1] 0.478872
##
##
## [1] "========== 2 =========="
##
```

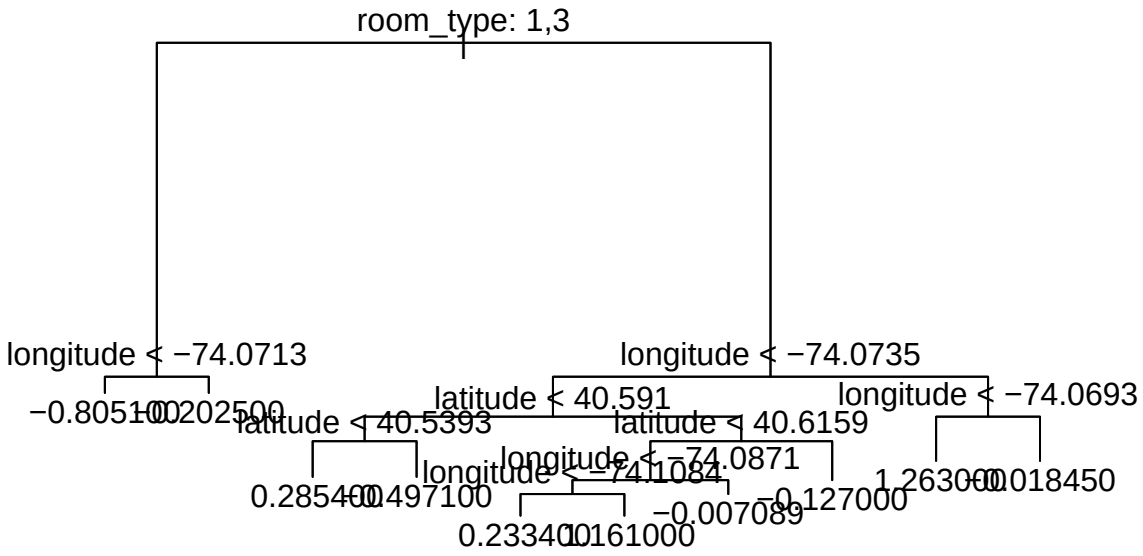
```
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type" "longitude"
## Number of terminal nodes:  4
## Residual mean deviance:  0.7795 = 12200 / 15650
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.2960 -0.5156 -0.1942  0.0000  0.3023  4.7530
```



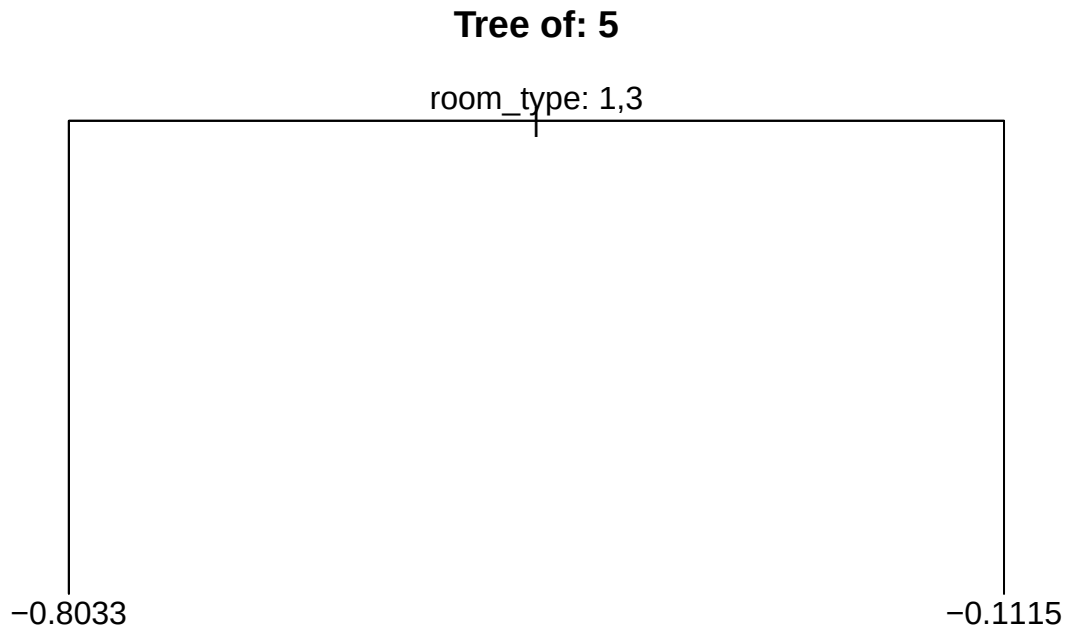

```
## [1] 0.7851166
##
##
## [1] "========== 3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type"
## Number of terminal nodes:  2
## Residual mean deviance:  0.3526 = 1488 / 4222
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.4000 -0.2929 -0.1225  0.0000  0.1335  4.9330
```



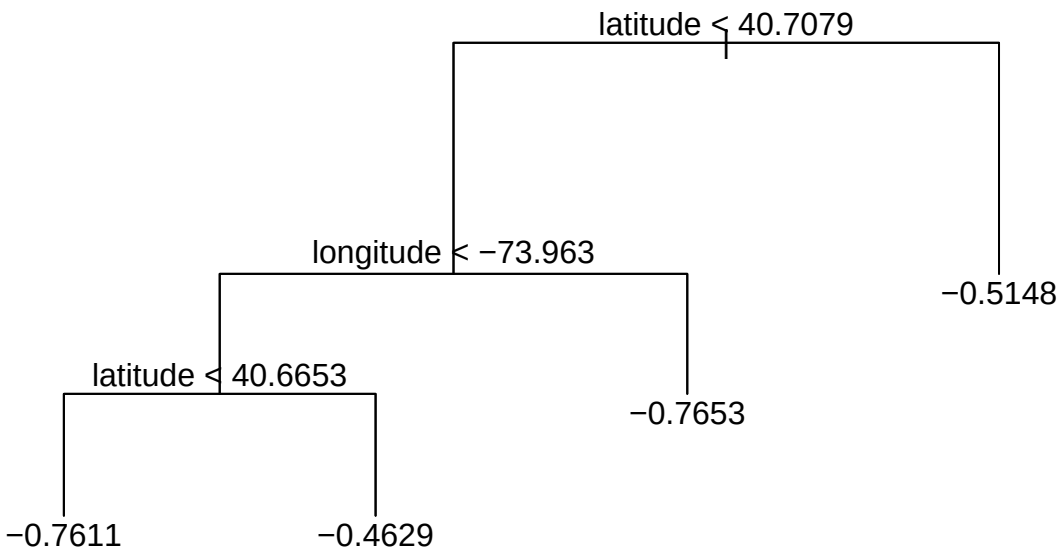

```
## [1] 0.4228161
##
##
## [1] "========== 4 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  10
## Residual mean deviance:  0.3202 = 84.54 / 264
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.3500 -0.3039 -0.1221  0.0000  0.2187  2.3540
```



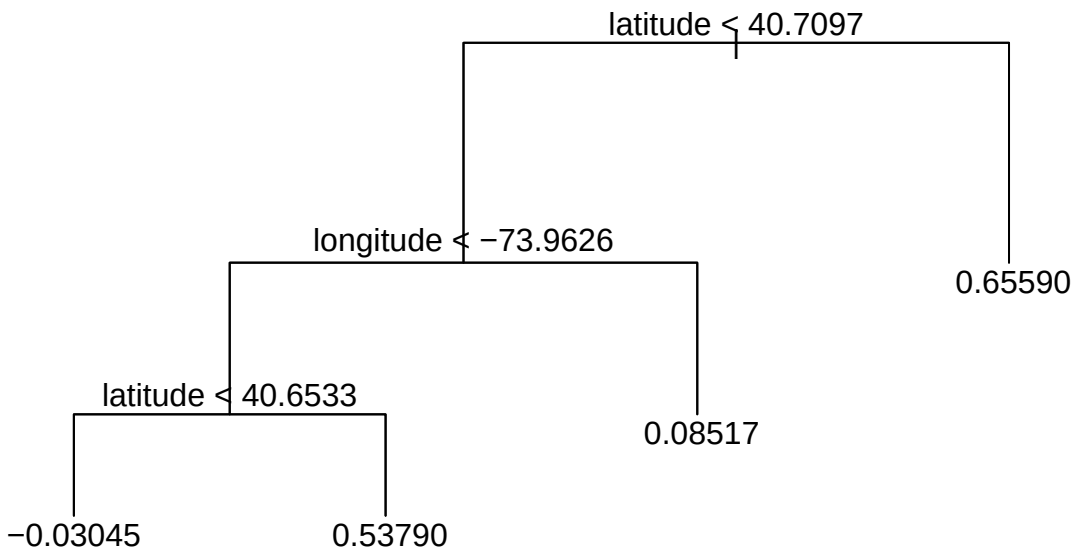

```
## [1] 0.3471879
##
##
## [1] "========== 5 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "room_type"
## Number of terminal nodes:  2
## Residual mean deviance:  0.3927 = 317.7 / 809
## Distribution of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



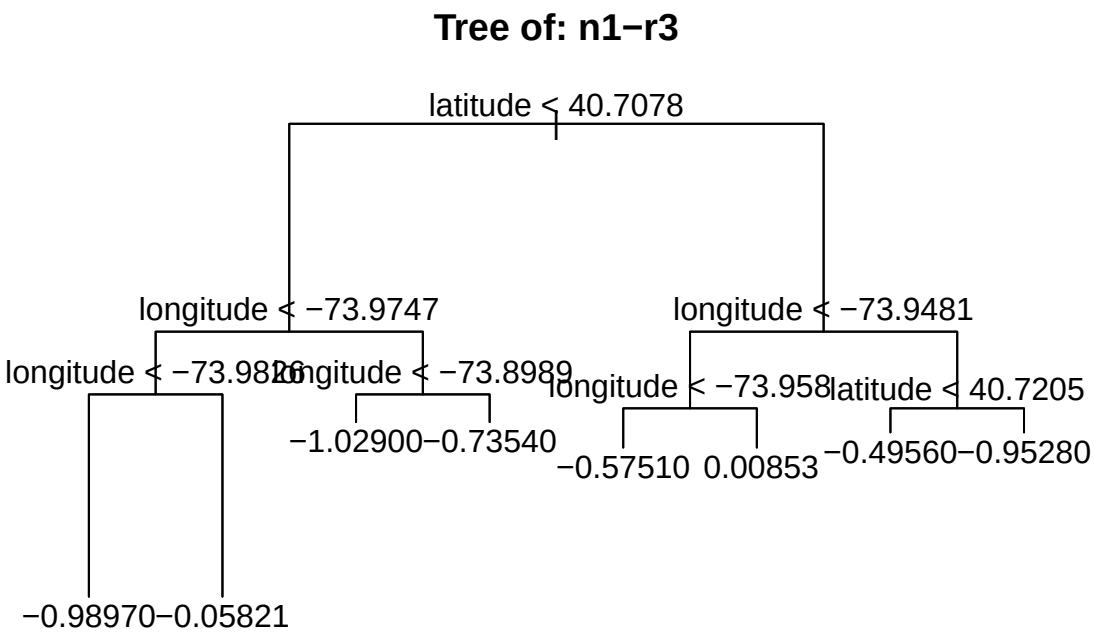

```
## [1] 0.2744778
##
##
## [1] "========== n1-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.1724 = 1158 / 6720
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.69640 -0.22300 -0.07884  0.00000  0.12210  4.95000
```



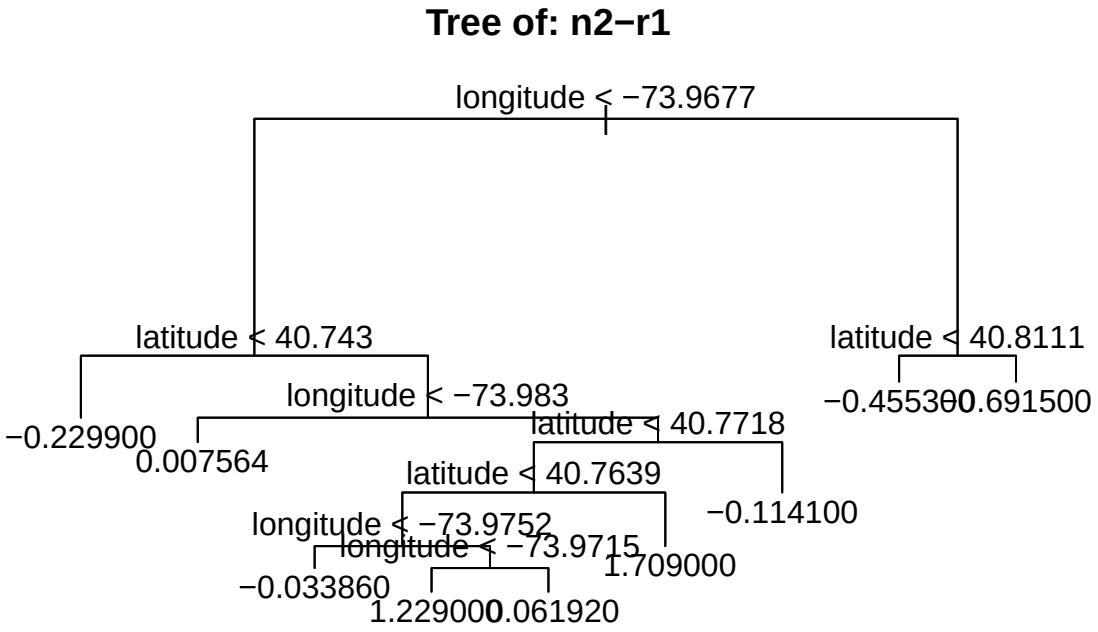

```
## [1] 0.1806116
##
##
## [1] "========== n1-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.7274 = 4538 / 6238
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9240 -0.5563 -0.2155  0.0000  0.2388  4.2040
```



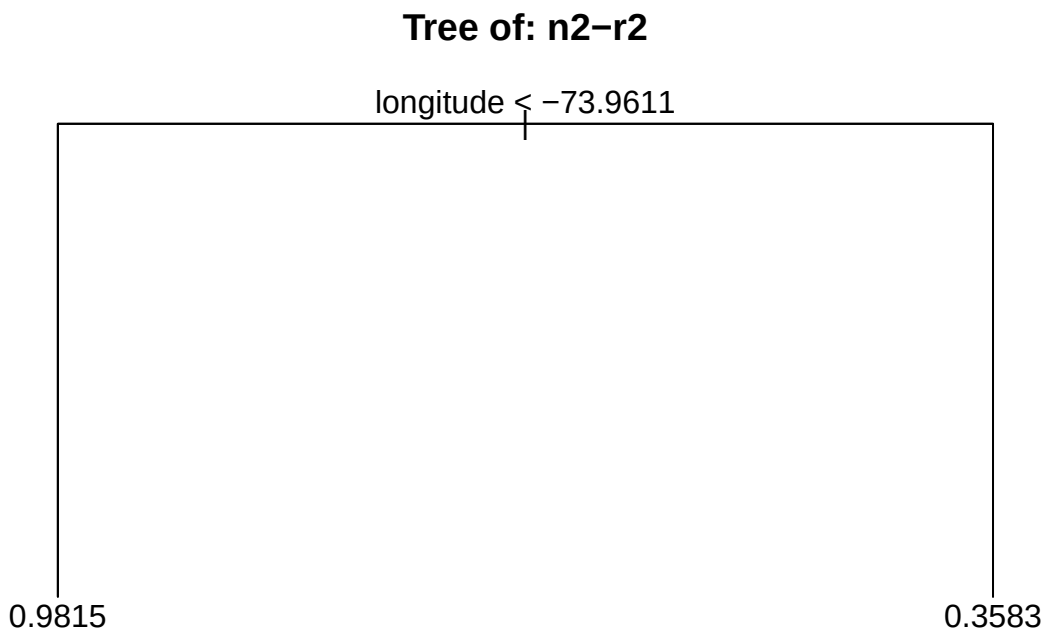

```
## [1] 0.7483236
##
##
## [1] "========== n1-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  8
## Residual mean deviance:  0.1442 = 38.21 / 265
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.02700 -0.15190 -0.06911  0.00000  0.04448  2.05000
```



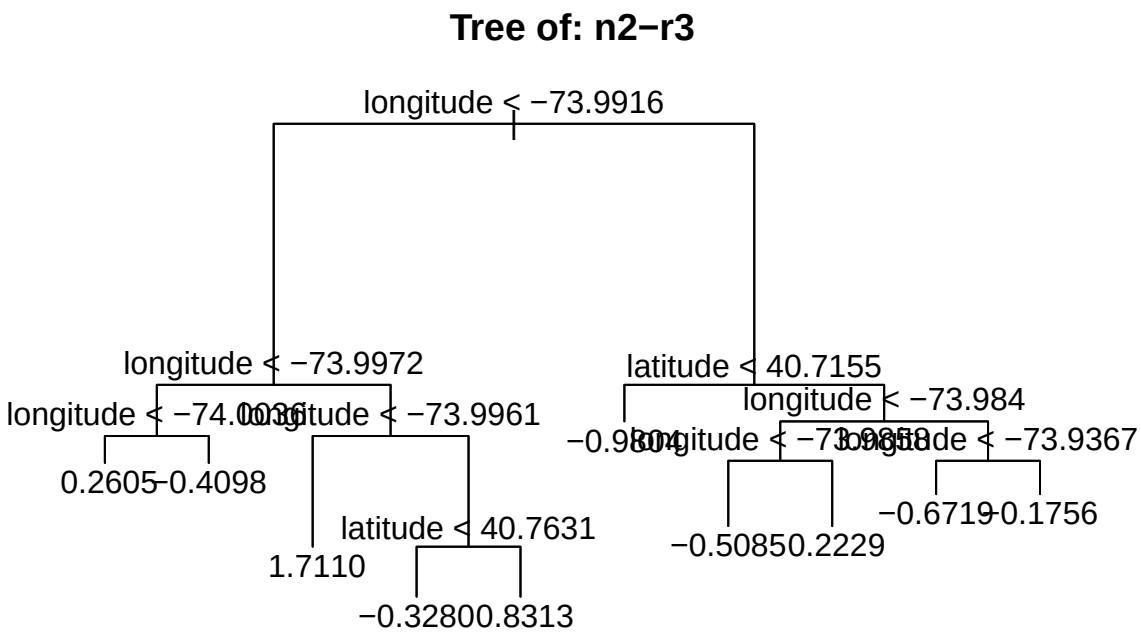

```
## [1] 0.2505339
##
##
## [1] "========== n2-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  9
## Residual mean deviance:  0.3895 = 2043 / 5246
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##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



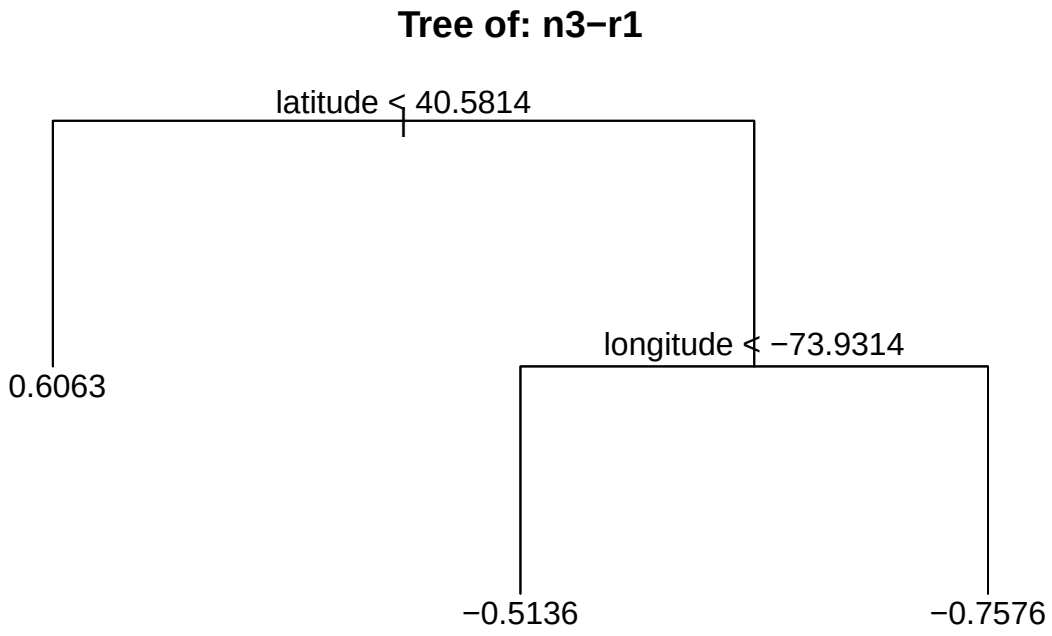

```
## [1] 0.3753113
##
##
## [1] "========== n2-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## [1] "longitude"
## Number of terminal nodes:  2
## Residual mean deviance:  1.01 = 8431 / 8344
## Distribution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



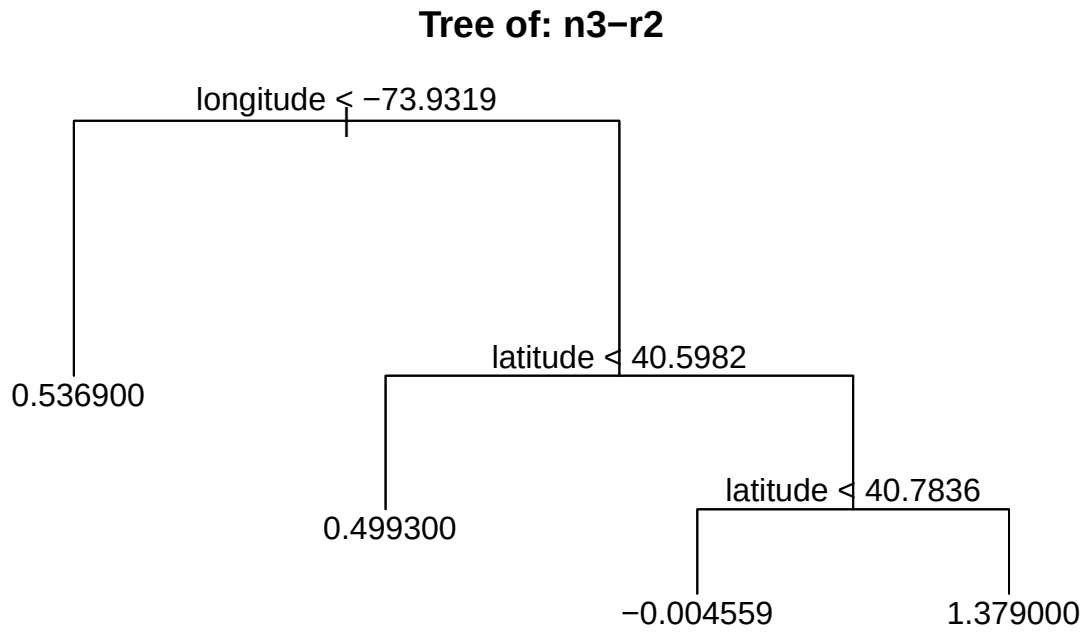

```
## [1] 0.9585893
##
##
## [1] "========== n2-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  10
## Residual mean deviance:  0.4952 = 151.5 / 306
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.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



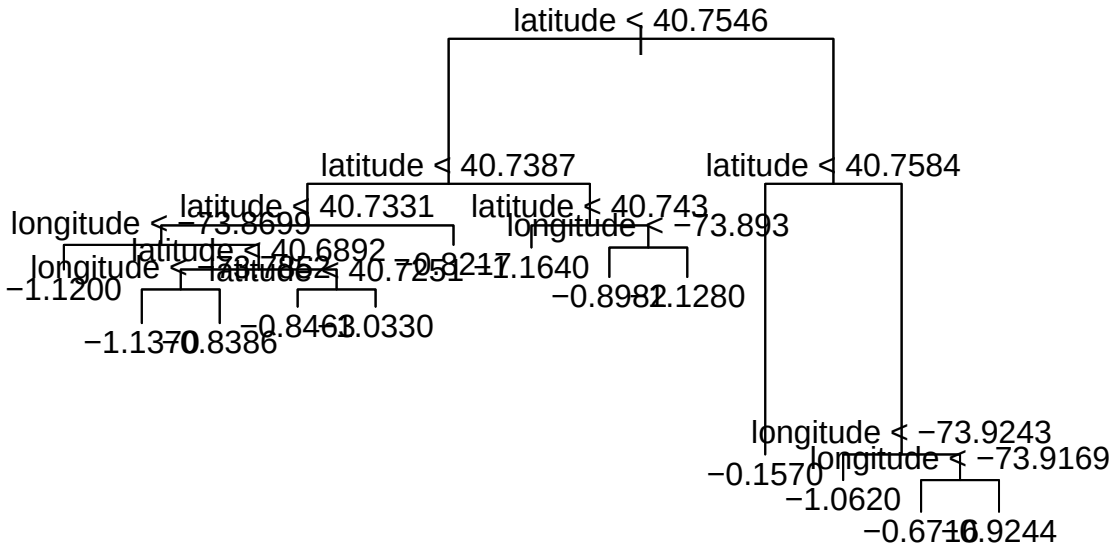

```
## [1] 0.4992158
##
##
## [1] "========== n3-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  3
## Residual mean deviance:  0.1738 = 389.1 / 2239
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.85200 -0.22640 -0.08418  0.00000  0.11440  4.94200
```



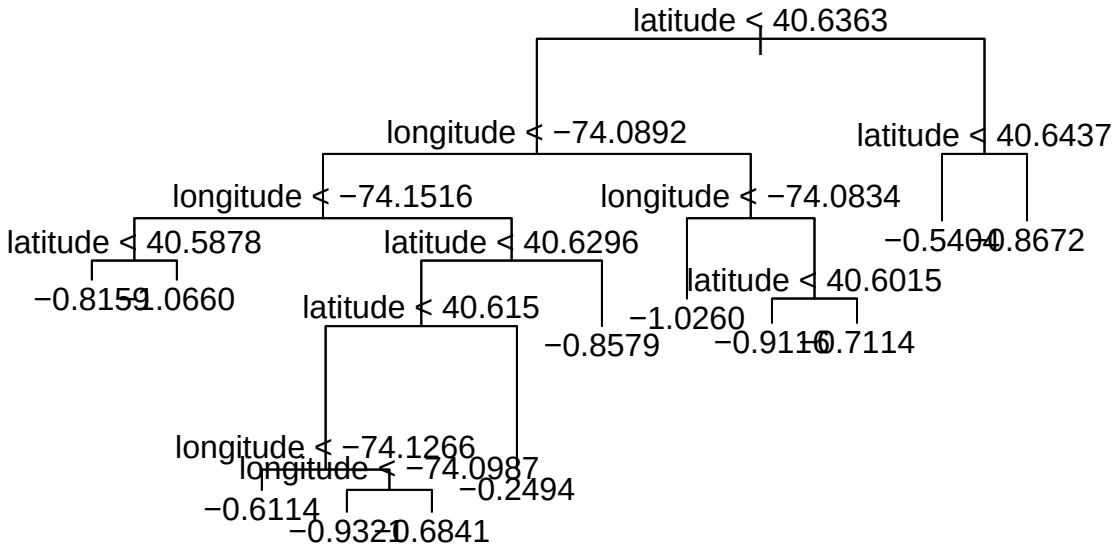

```
## [1] 0.1600965
##
##
## [1] "========== n3-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  4
## Residual mean deviance:  0.6455 = 890.9 / 1380
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.1360 -0.4910 -0.2411  0.0000  0.2133  4.1890
```



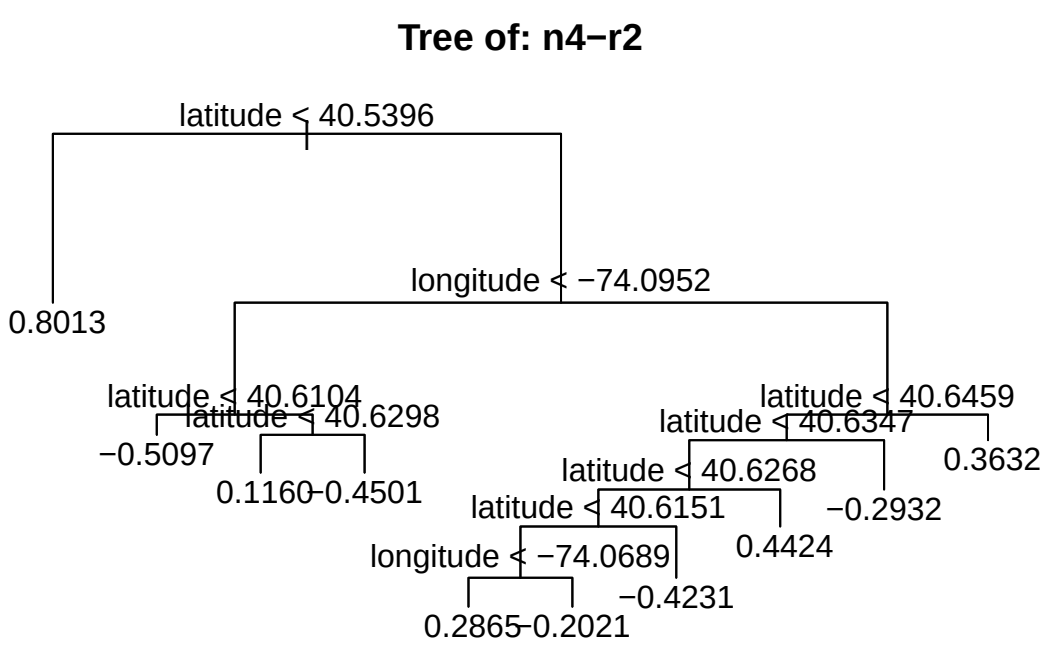

```
## [1] 0.6995247
##
##
## [1] "========== n3-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  13
## Residual mean deviance:  0.07148 = 8.221 / 115
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.82700 -0.12060 -0.02272  0.00000  0.06309  0.93380
```



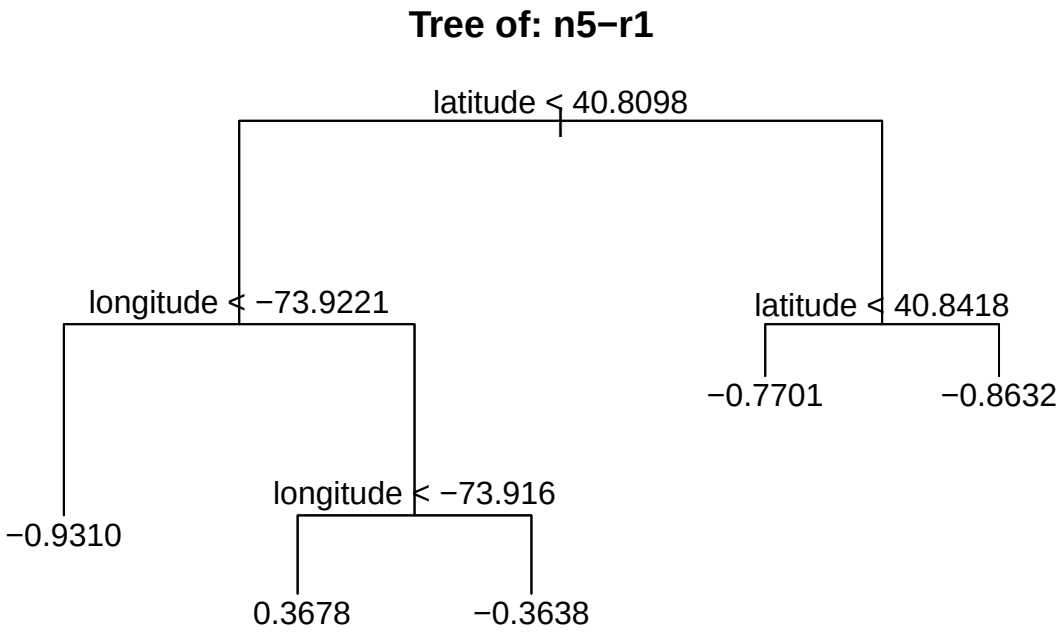

```
## [1] 0.6218172
##
##
## [1] "========== n4-r1 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  12
## Residual mean deviance:  0.08816 = 9.962 / 113
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.56420 -0.14540 -0.04604  0.00000  0.12440  1.31700
```



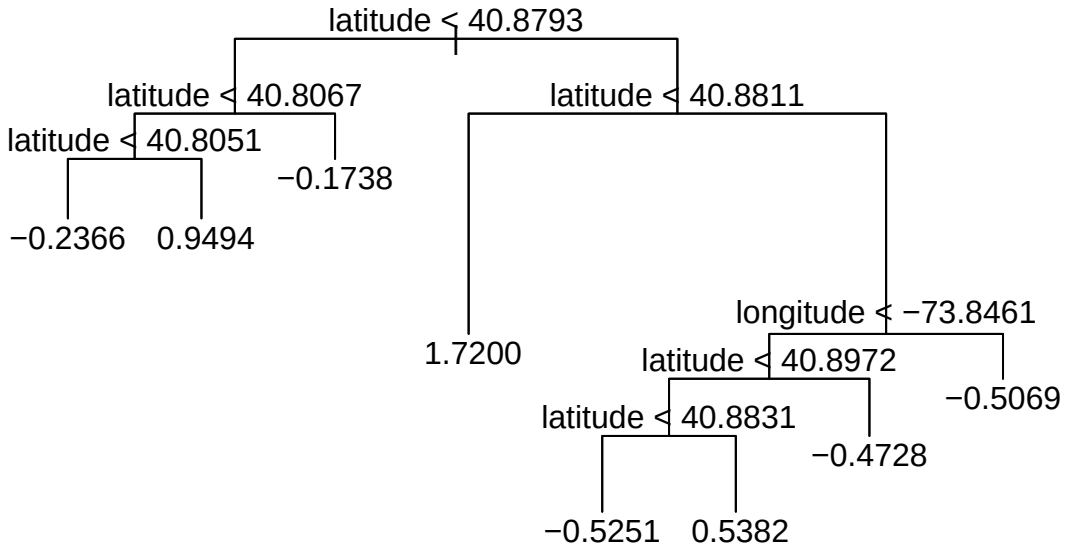

```
## [1] 0.2339327
##
##
## [1] "========== n4-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  10
## Residual mean deviance:  0.3348 = 34.49 / 103
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.44500 -0.32670 -0.07762  0.00000  0.27450  1.62700
```



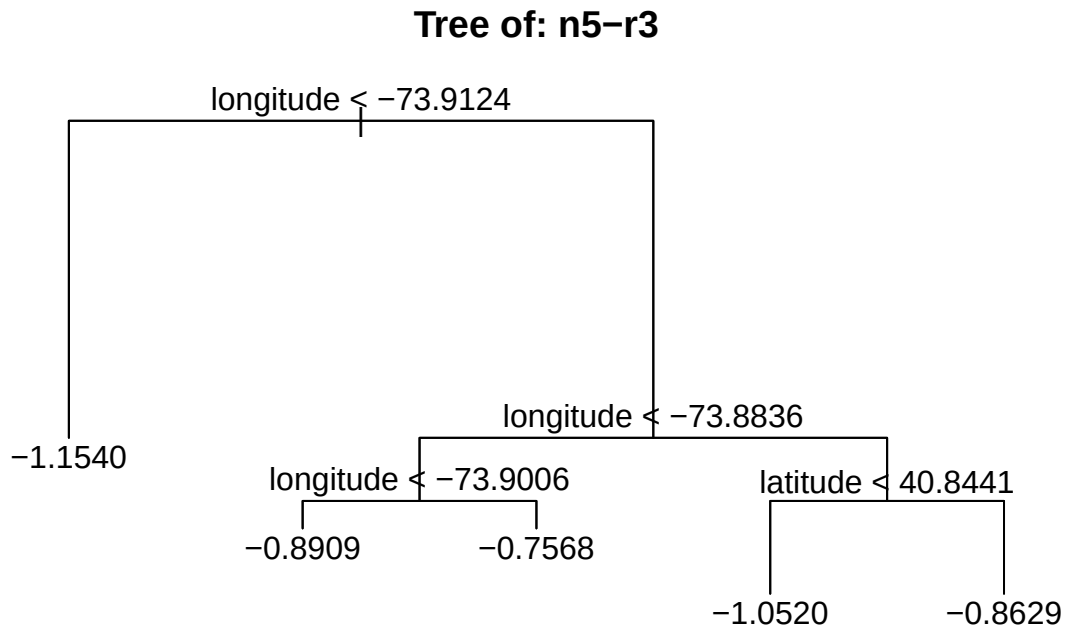

```
## [1] 1.189299
##
##
## [1] "========== n4-r3 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Variables actually used in tree construction:
## character(0)
## Number of terminal nodes:  1
## Residual mean deviance:  0.3279 = 1.311 / 4
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.531600 -0.520300 -0.009088  0.000000  0.218100  0.842900
## Not possible to plot tree:  n4-r3[1] 0.224188
##
##
## [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
```



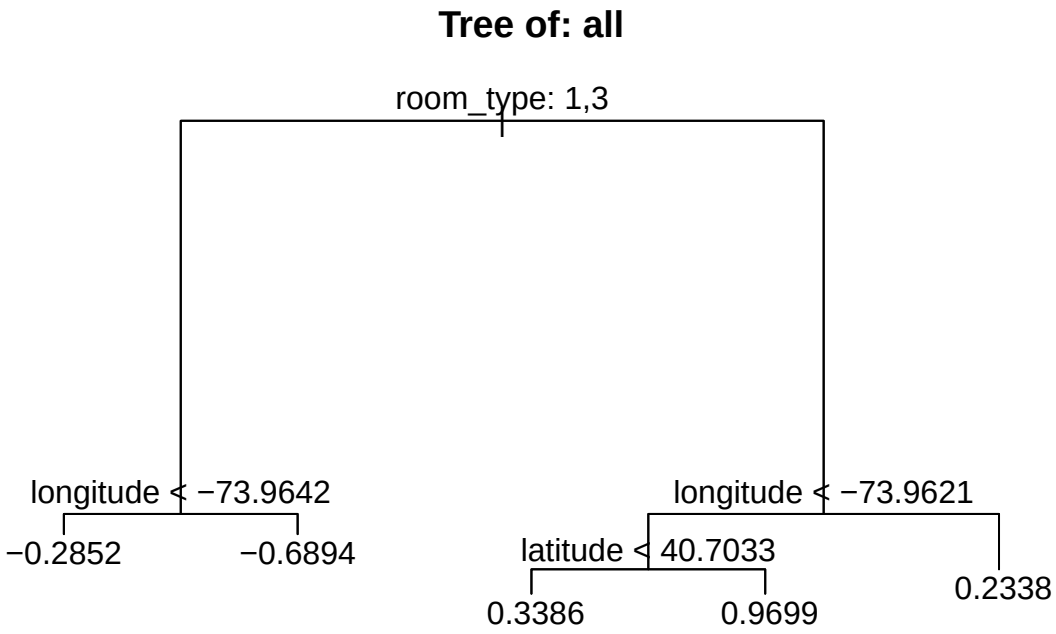

```
## [1] 0.1611146
##
##
## [1] "========== n5-r2 =========="
##
## Regression tree:
## tree(formula = price ~ ., data = trains[[sub]])
## Number of terminal nodes:  8
## Residual mean deviance:  0.6961 = 169.1 / 243
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.2490 -0.4638 -0.1854  0.0000  0.2079  3.7900
```




```
## [1] 0.7622148
##
##
## [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)

  #  varImpPlot(res)
  #  plot(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: 0.4264125
##                     % Var explained: 41.65
## [1] "MSE: 0.454012374017494"
##
##
## [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: 0.7317802
##                     % Var explained: 37.03
## [1] "MSE: 0.735595108176858"
##
##
## [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: 0.3368627
##                     % Var explained: 33.87
## [1] "MSE: 0.396276824161383"
##
##
## [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: 0.4185199
##                     % Var explained: 21.55
## [1] "MSE: 0.336750091255294"
##
##
## [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: 0.3910493
##                     % Var explained: 21.81
## [1] "MSE: 0.271528940234555"
##
##
## [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: 0.178747
##                     % Var explained: 4.61
## [1] "MSE: 0.193214029929856"
##
##
## [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: 0.7899636
##                     % Var explained: 0.84
## [1] "MSE: 0.792951540112952"
##
##
## [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: 0.1945029
##                     % Var explained: 4
## [1] "MSE: 0.242160186139755"
##
##
## [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: 0.3996346
##                     % Var explained: 23.52
## [1] "MSE: 0.380591824585714"
##
##
## [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: 1.036225
##                     % Var explained: 4.26
## [1] "MSE: 0.977070559784507"
##
##
## [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: 0.7066876
##                     % Var explained: -7.71
## [1] "MSE: 0.55878633568084"
##
##
## [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: 0.1717809
##                     % Var explained: 5.26
## [1] "MSE: 0.163004804603044"
##
##
## [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: 0.6692033
##                     % Var explained: 1.91
## [1] "MSE: 0.711645545321903"
##
##
## [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: 0.1177455
##                     % Var explained: -5.91
## [1] "MSE: 0.631748073062785"
##
##
## [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: 0.1314448
##                     % Var explained: -6.64
## [1] "MSE: 0.226995788547981"
##
##
## [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: 0.5184733
##                     % Var explained: -11.66
## [1] "MSE: 1.0488941704488"
## 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: 0.3306639
##                     % Var explained: -26.07
## [1] "MSE: 0.0726859561516477"
##
##
## [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: 0.2160591
##                     % Var explained: -18.81
## [1] "MSE: 0.134365991758027"
##
##
## [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: 0.8916108
##                     % Var explained: -10.1
## [1] "MSE: 0.662059001453434"
##
##
## [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: 0.11065
##                     % Var explained: -25.34
## [1] "MSE: 0.174921807893669"
##
##
## [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: 0.5681364
```

```
##                       % Var explained: 42.68
## [1] "MSE: 0.582992437184199"
```

```
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, classificat

  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):       0.4257217
## R squared (OOB):                  0.4174863
## [1] "MSE: 0.451708225857914"
##
##
## [1] "========== 2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      15657
## Number of independent variables:  3
## Mtry:                             1
```

```
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7340816
## R squared (OOB):                  0.368355
## [1] "MSE: 0.740157062922684"
##
##
## [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):       0.3375653
## R squared (OOB):                  0.3375071
## [1] "MSE: 0.395339302379045"
##
##
## [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):       0.4252956
## R squared (OOB):                  0.2057241
## [1] "MSE: 0.347611399151743"
##
##
## [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):       0.3939942
## R squared (OOB):                  0.213139
## [1] "MSE: 0.272555779669846"
##
##
## [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):       0.1780698
## R squared (OOB):                  0.04983137
## [1] "MSE: 0.193773040355645"
##
##
## [1] "========== n1-r2 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      6242
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.7908384
## R squared (OOB):                  0.0074946
## [1] "MSE: 0.795616140713617"
##
##
## [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):       0.1942968
## R squared (OOB):                  0.04449433
## [1] "MSE: 0.238557396679357"
##
##
## [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):       0.3997818
## R squared (OOB):                  0.2351081
## [1] "MSE: 0.377671573066202"
##
##
## [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):       1.033592
## R squared (OOB):                  0.04510787
## [1] "MSE: 0.976905962011513"
##
##
## [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):       0.7180809
## R squared (OOB):                  -0.09100462
## [1] "MSE: 0.560614095079643"
##
##
## [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):       0.1709925
## R squared (OOB):                  0.05740717
## [1] "MSE: 0.161917430805412"
##
##
## [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):       0.6731786
## R squared (OOB):                  0.01399647
## [1] "MSE: 0.711883307878391"
##
```

```
##
## [1] "========== n3-r3 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      128
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1162408
## R squared (OOB):                  -0.03737978
## [1] "MSE: 0.631597850921341"
##
##
## [1] "========== n4-r1 =========="
## Ranger result
##
## Call:
##  ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      125
## Number of independent variables:  2
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.129391
## R squared (OOB):                  -0.0413836
## [1] "MSE: 0.229438107276058"
##
##
## [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):       0.5321167
```

```
## R squared (OOB):                  -0.1358403
## [1] "MSE: 1.06613824189643"
##
##
## [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):       0.4024009
## R squared (OOB):                  -0.2273535
## [1] "MSE: 0.22875012071697"
##
##
## [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):       0.2144384
## R squared (OOB):                  -0.1764572
## [1] "MSE: 0.134103174218686"
##
##
## [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):       0.8949554
## R squared (OOB):                  -0.10071
## [1] "MSE: 0.661771200765559"
##
##
## [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):       0.1066221
## R squared (OOB):                  -0.1767763
## [1] "MSE: 0.173822949859432"
##
##
## [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):       0.5432278
## R squared (OOB):                  0.4519412
## [1] "MSE: 0.555208346242052"
```

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

NEURAL NETWORKS

```
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
## _________________________________________________________________
## ........................
```

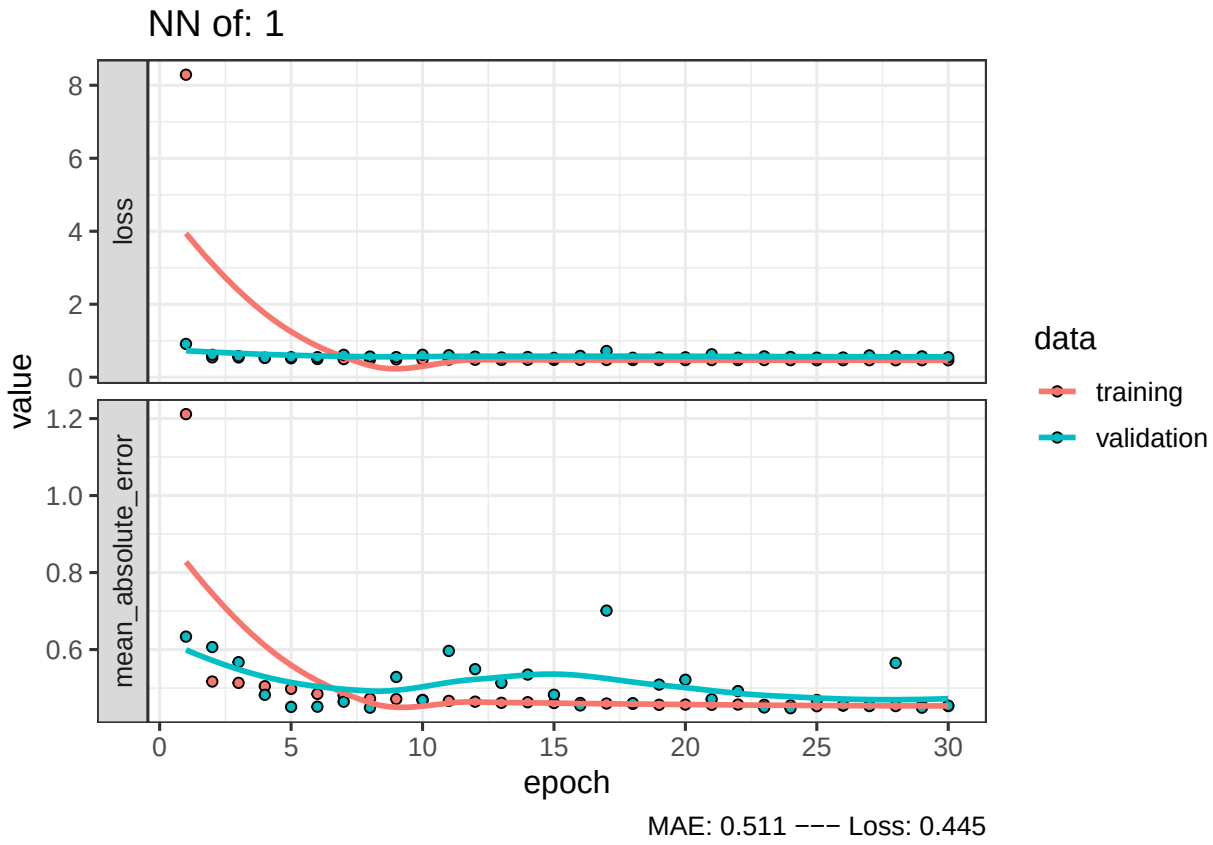
```
nn$name = "Neural Networks"
model_lis$neural_networks = nn
```

NN plots

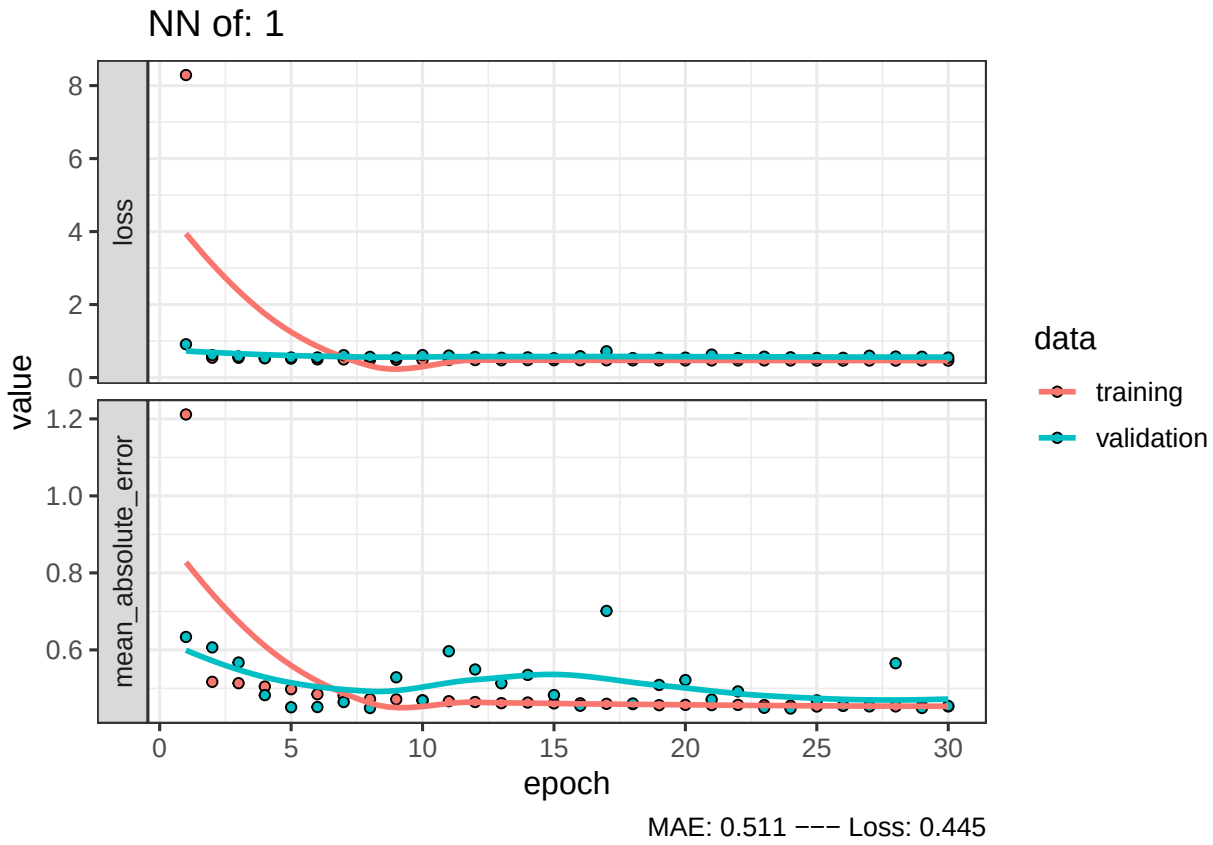
```
for (sub in names(nn))
{
  if(sub != "name")
  {
    m = nn[[sub]]
    hist = m$history
    print(paste0("========== ",sub, " =========="))
    str = paste0("MAE: ",round(m$mae,3)," --- Loss: ",round(m$loss,3))
    p = plot(hist, y ~ x) +  theme_bw(base_size = 12)  +ggtitle(paste0("NN of: ",sub))+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

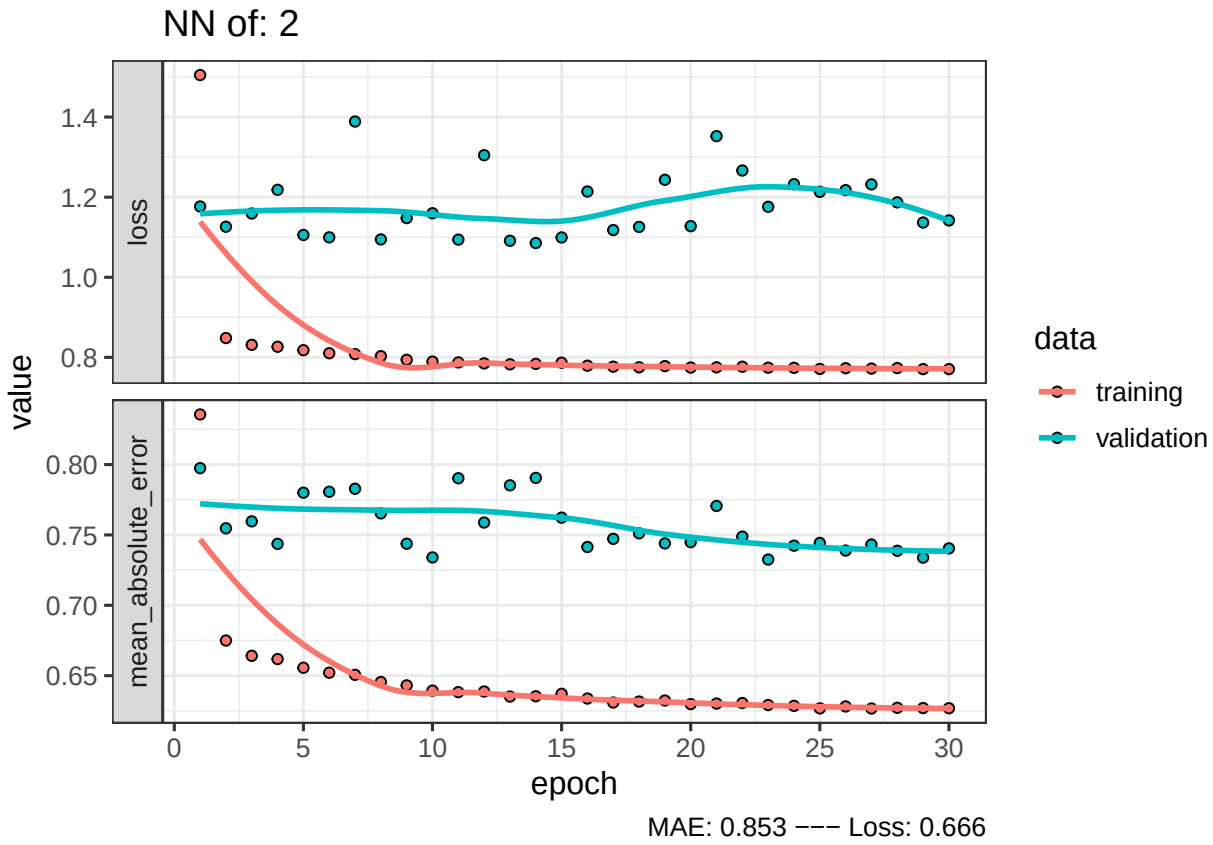



```
## `geom_smooth()` using formula 'y ~ x'
```

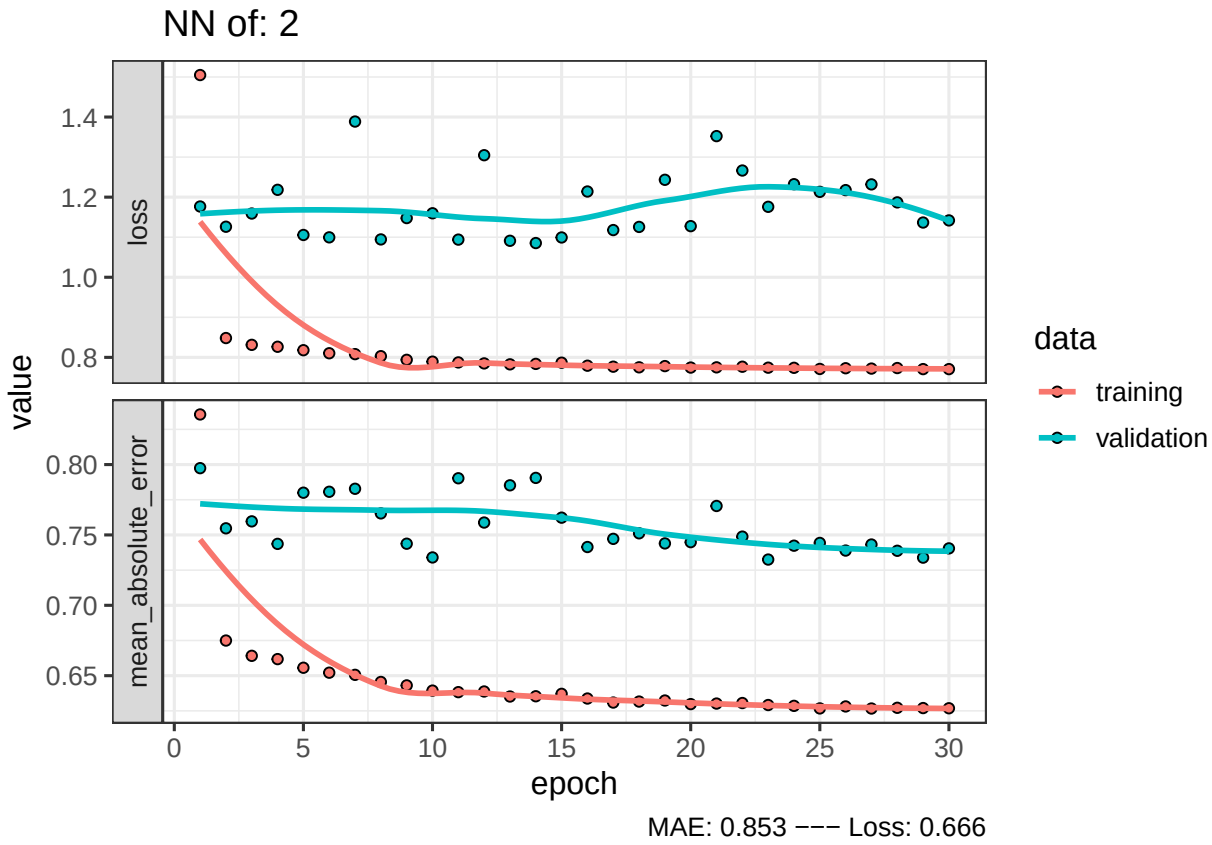



```
## MAE:  0.5105356
## Loss:  0.4452797
##
## [1] "========== 2 =========="
```

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

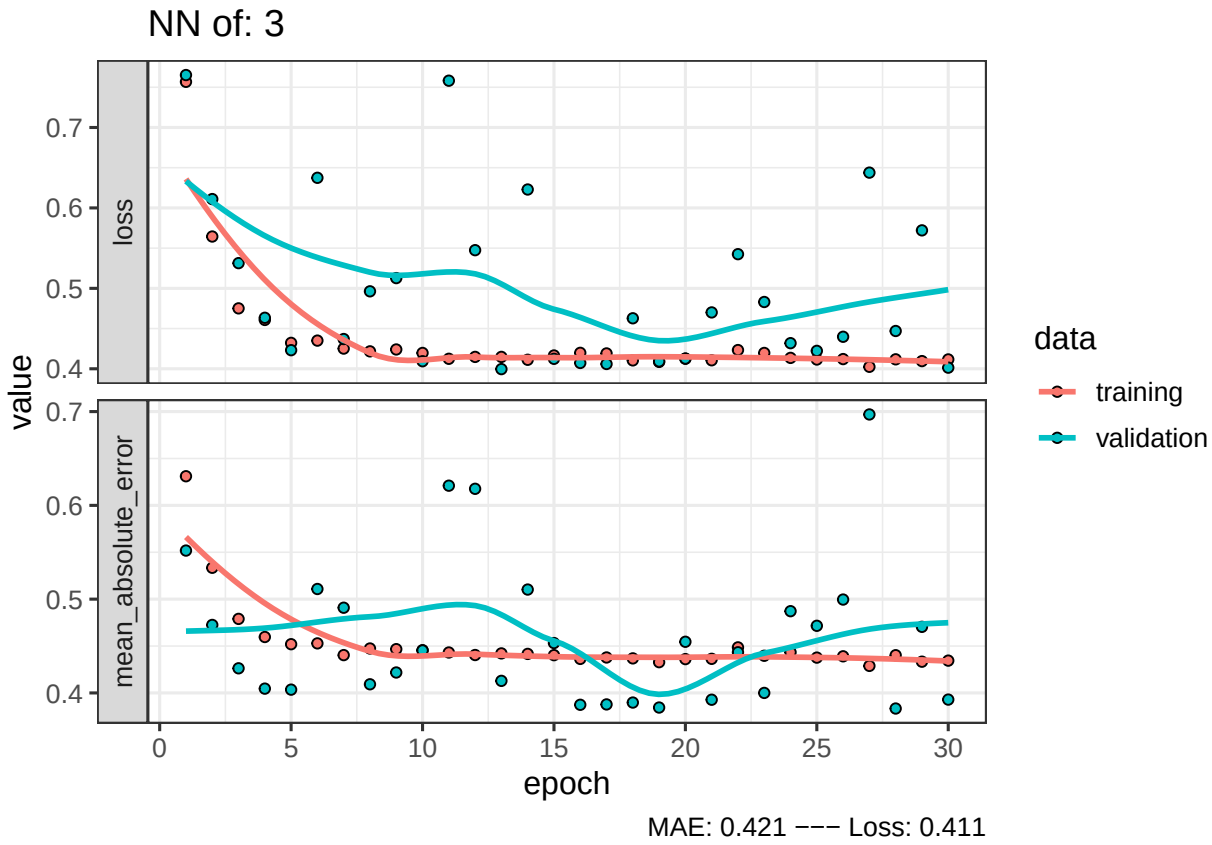



```
## `geom_smooth()` using formula 'y ~ x'
```

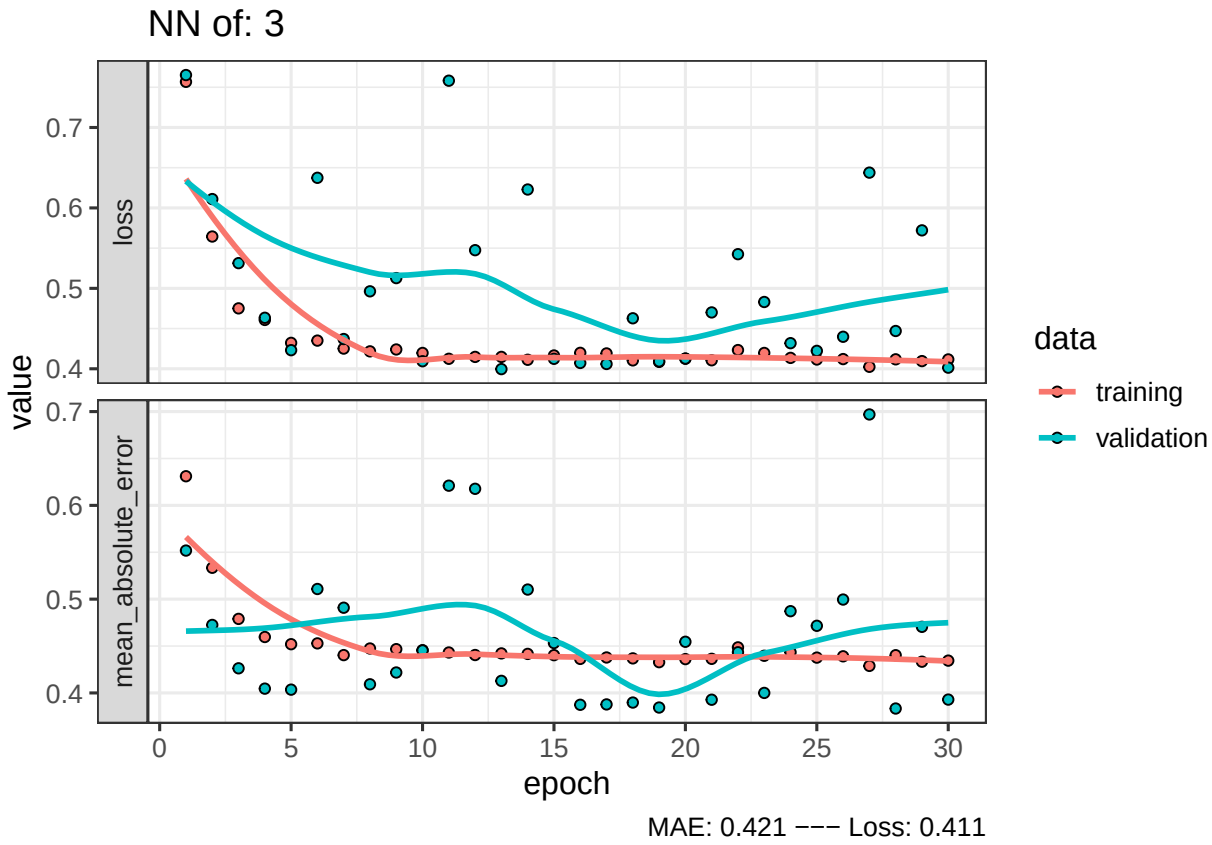



```
## MAE:  0.8534026
## Loss:  0.6659468
##
## [1] "========== 3 =========="
```

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

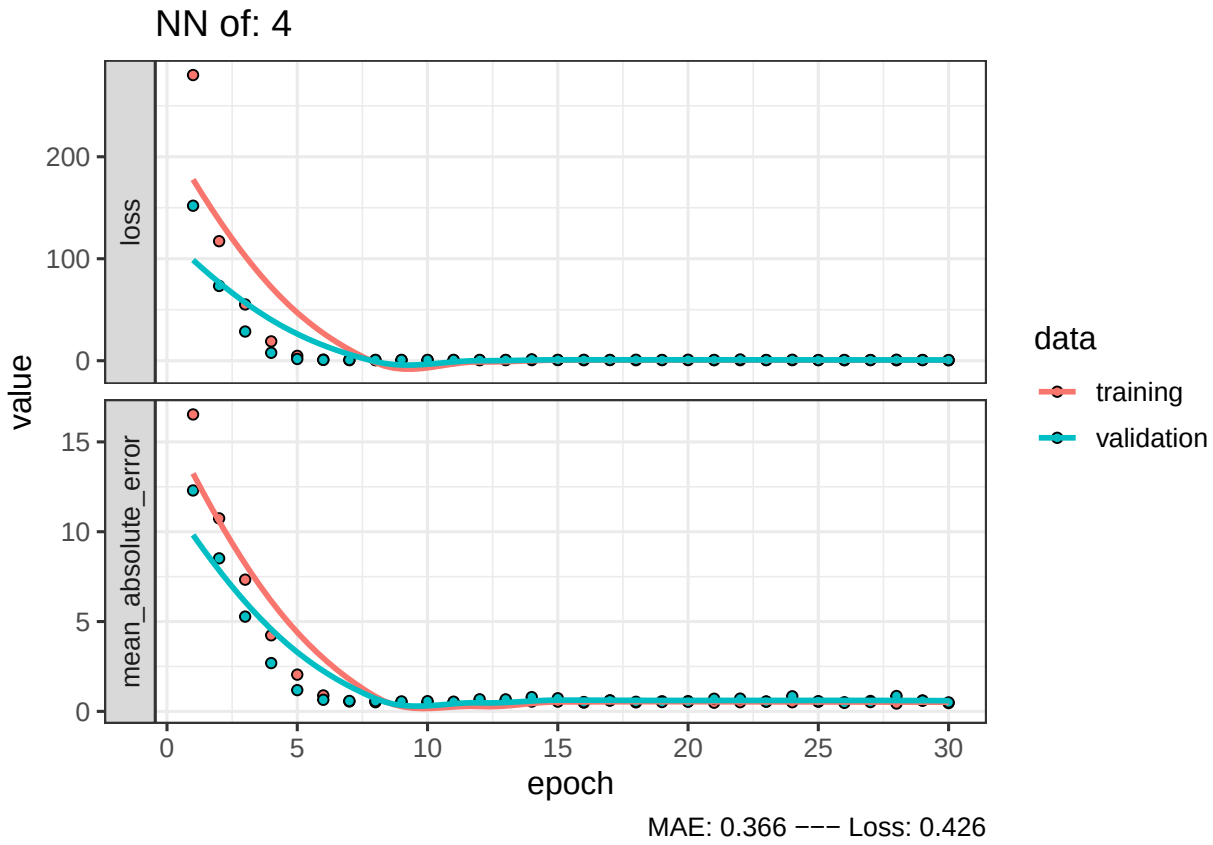



```
## `geom_smooth()` using formula 'y ~ x'
```

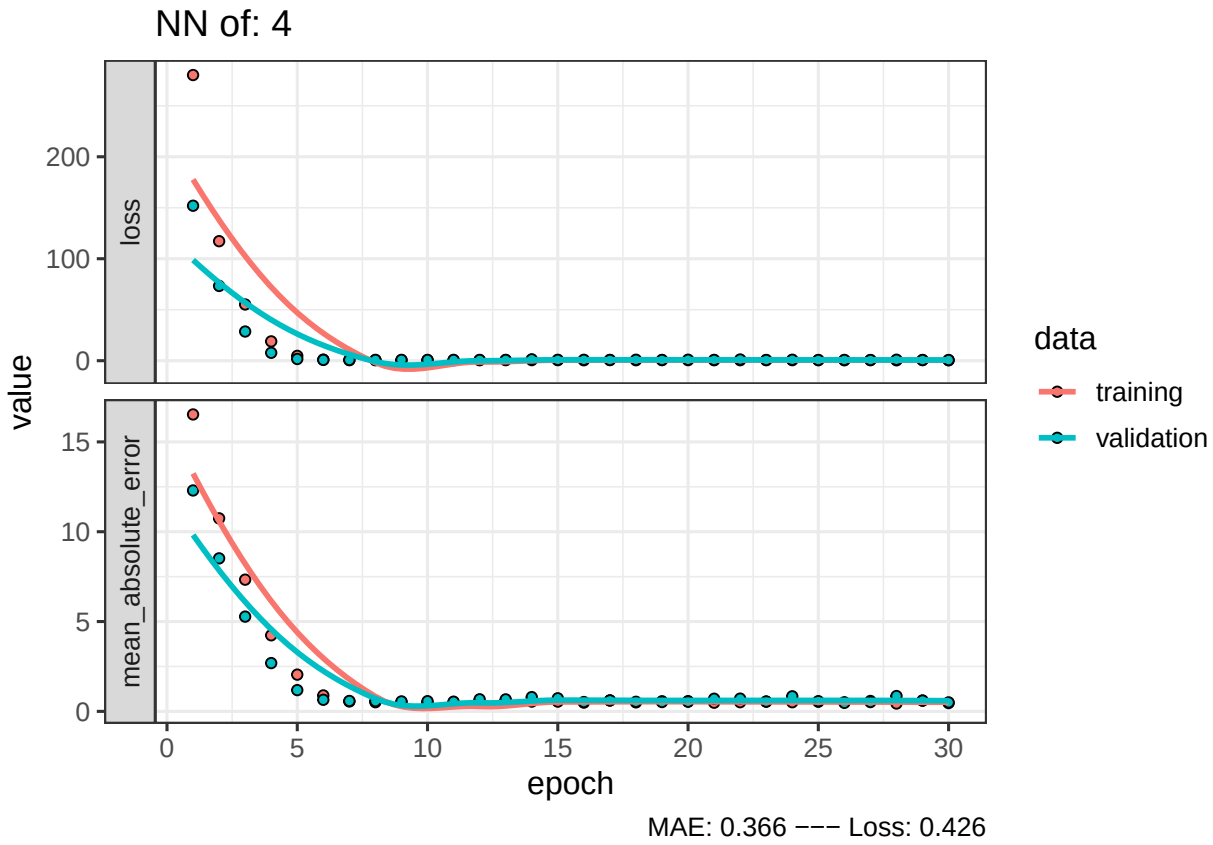



```
## MAE:  0.4212551
## Loss:  0.4113655
##
## [1] "========== 4 =========="
```

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

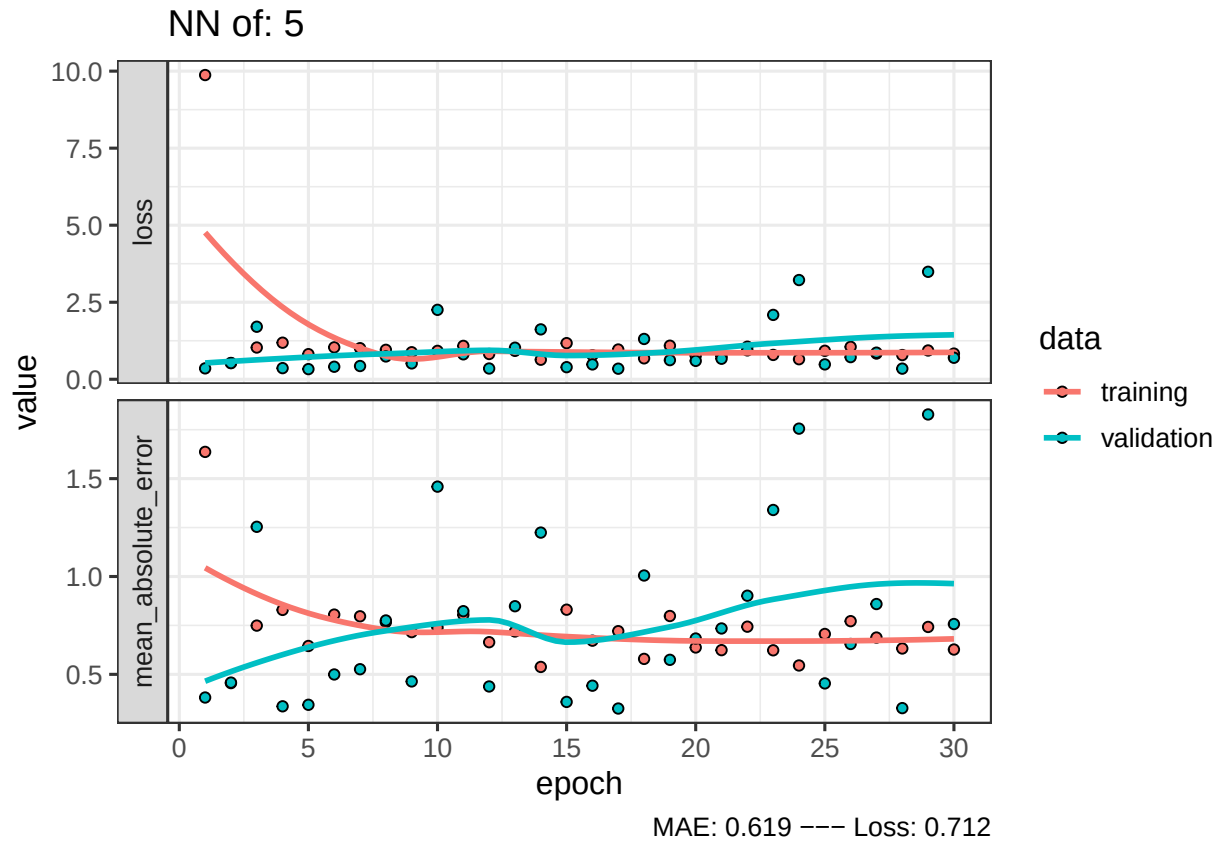



```
## `geom_smooth()` using formula 'y ~ x'
```

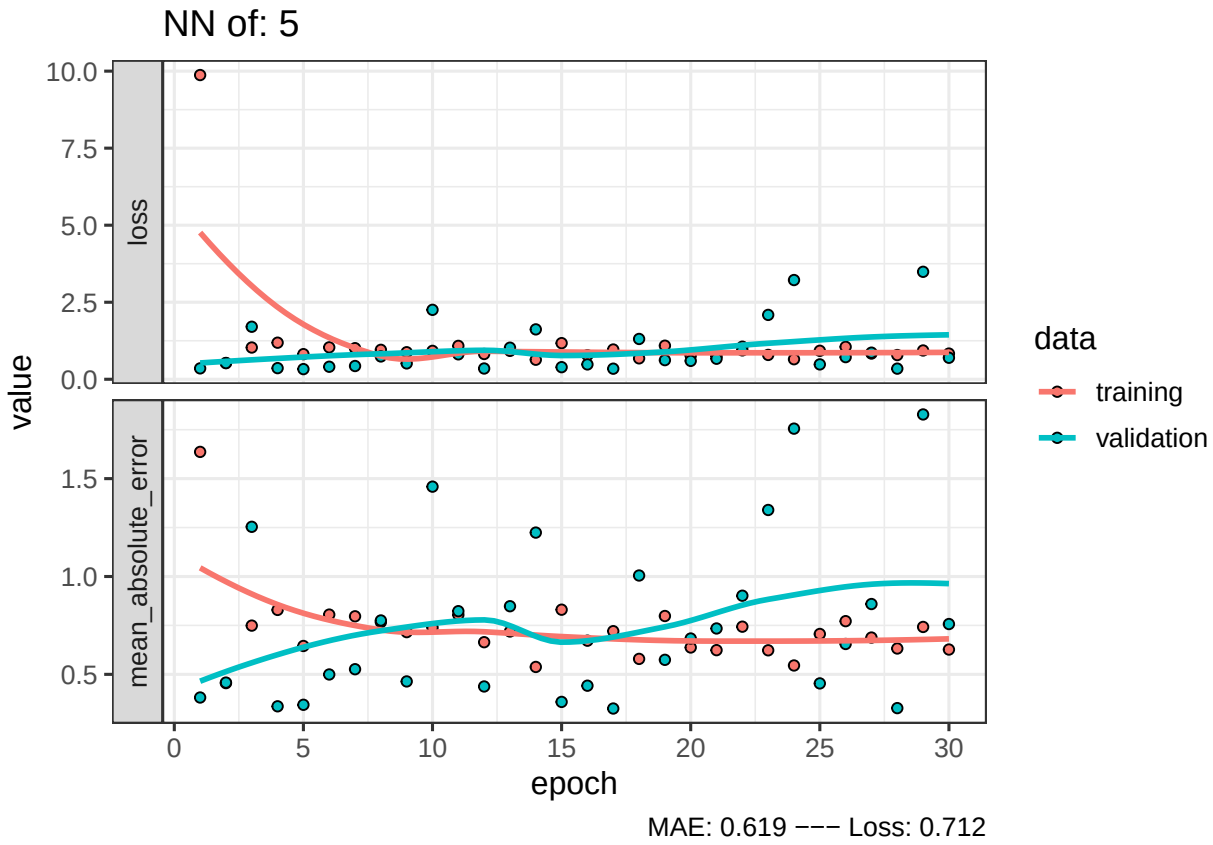



```
## MAE:  0.3655377
## Loss:  0.4257745
##
## [1] "========== 5 =========="
```

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

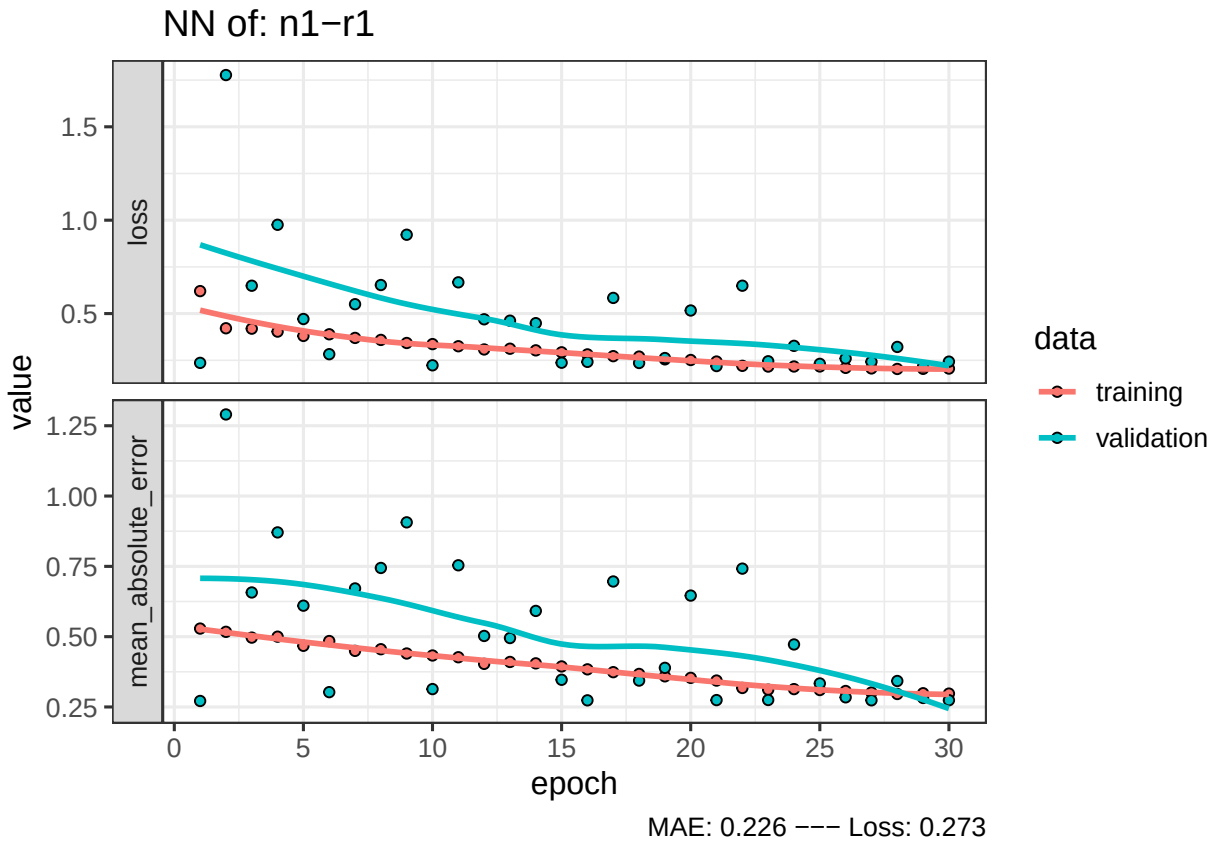



```
## `geom_smooth()` using formula 'y ~ x'
```

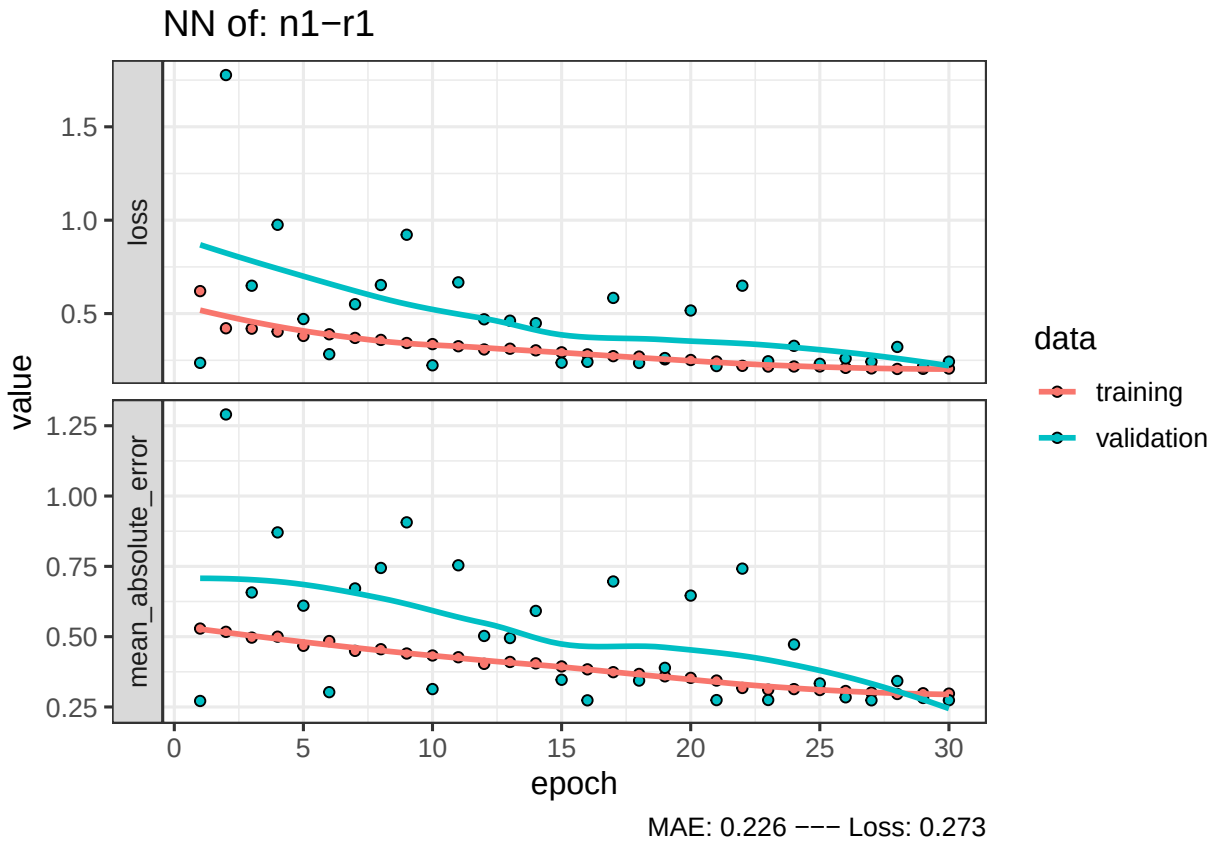



```
## MAE:  0.6189408
## Loss:  0.7115626
##
## [1] "========== n1-r1 =========="
```

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

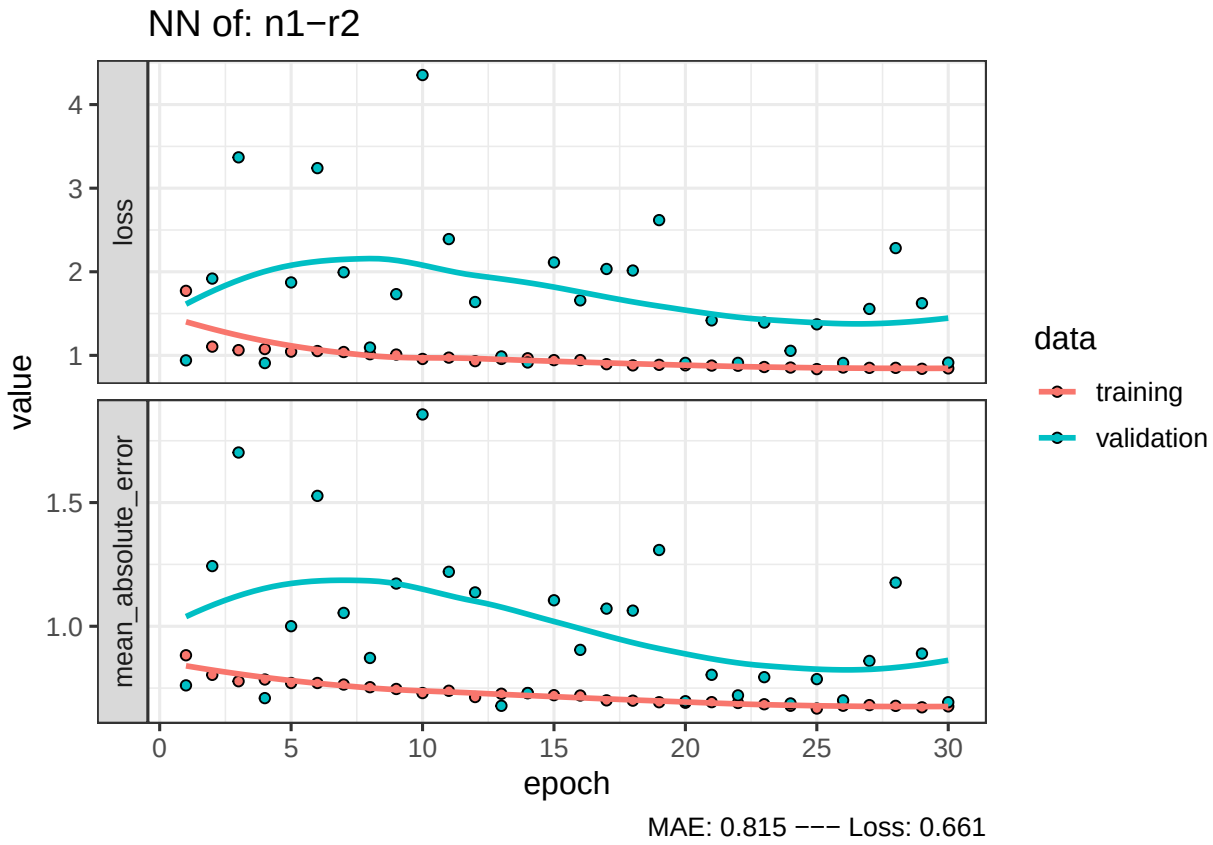



```
## `geom_smooth()` using formula 'y ~ x'
```

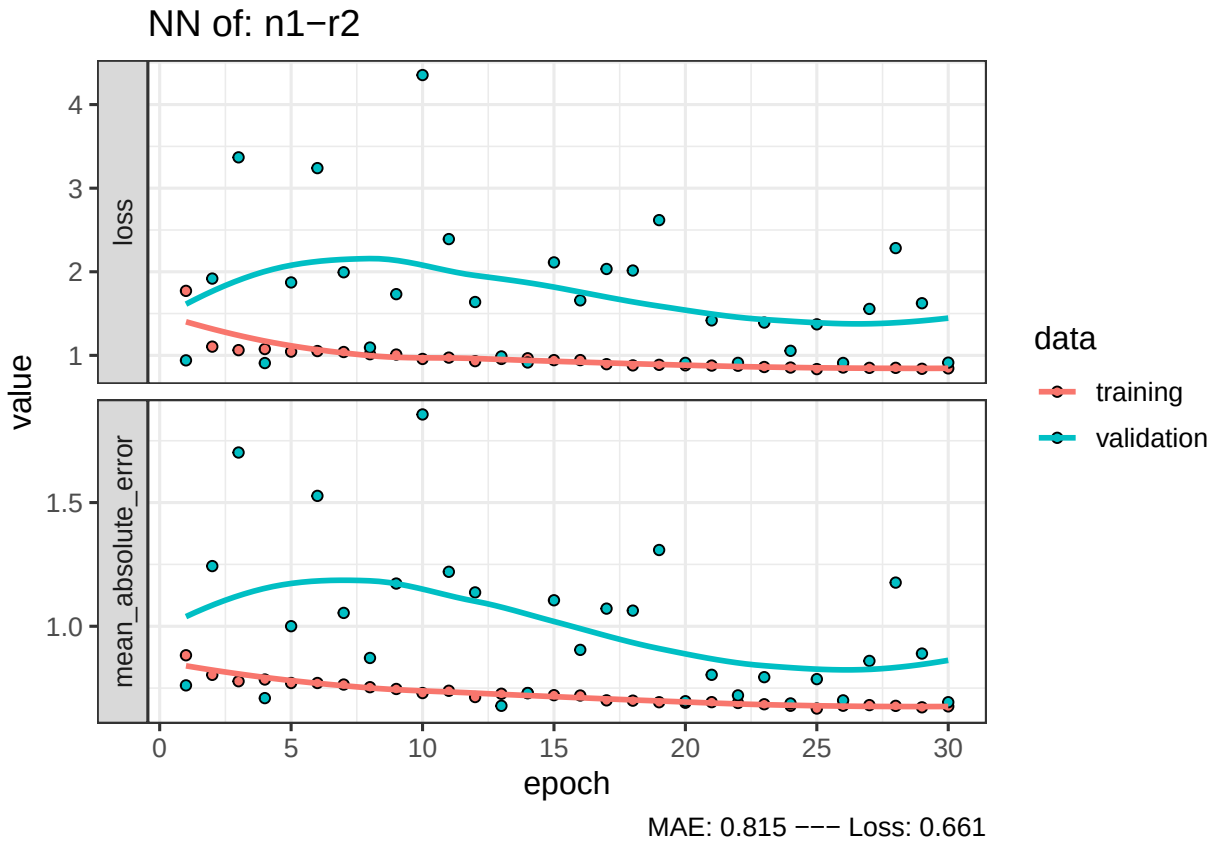



```
## MAE:  0.2255452
## Loss:  0.2728455
##
## [1] "========== n1-r2 =========="
```

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

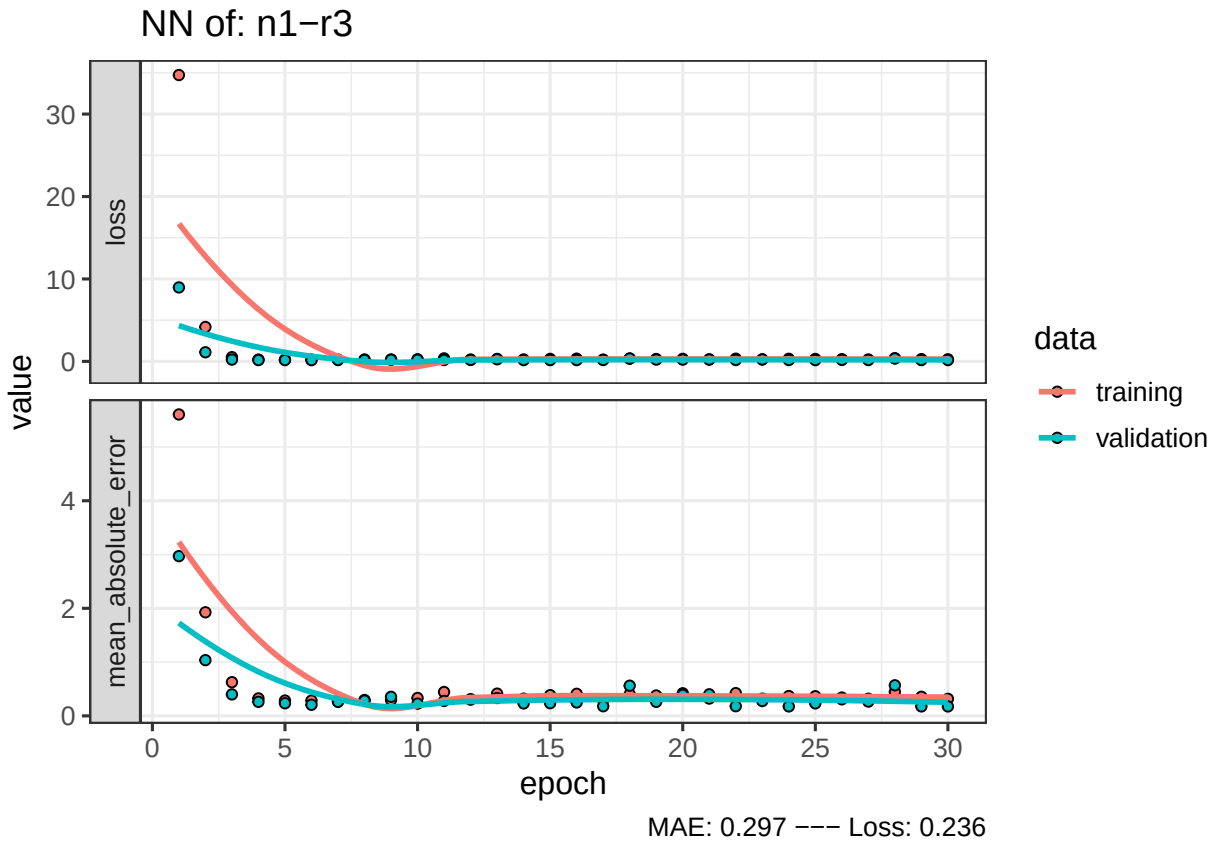



```
## `geom_smooth()` using formula 'y ~ x'
```

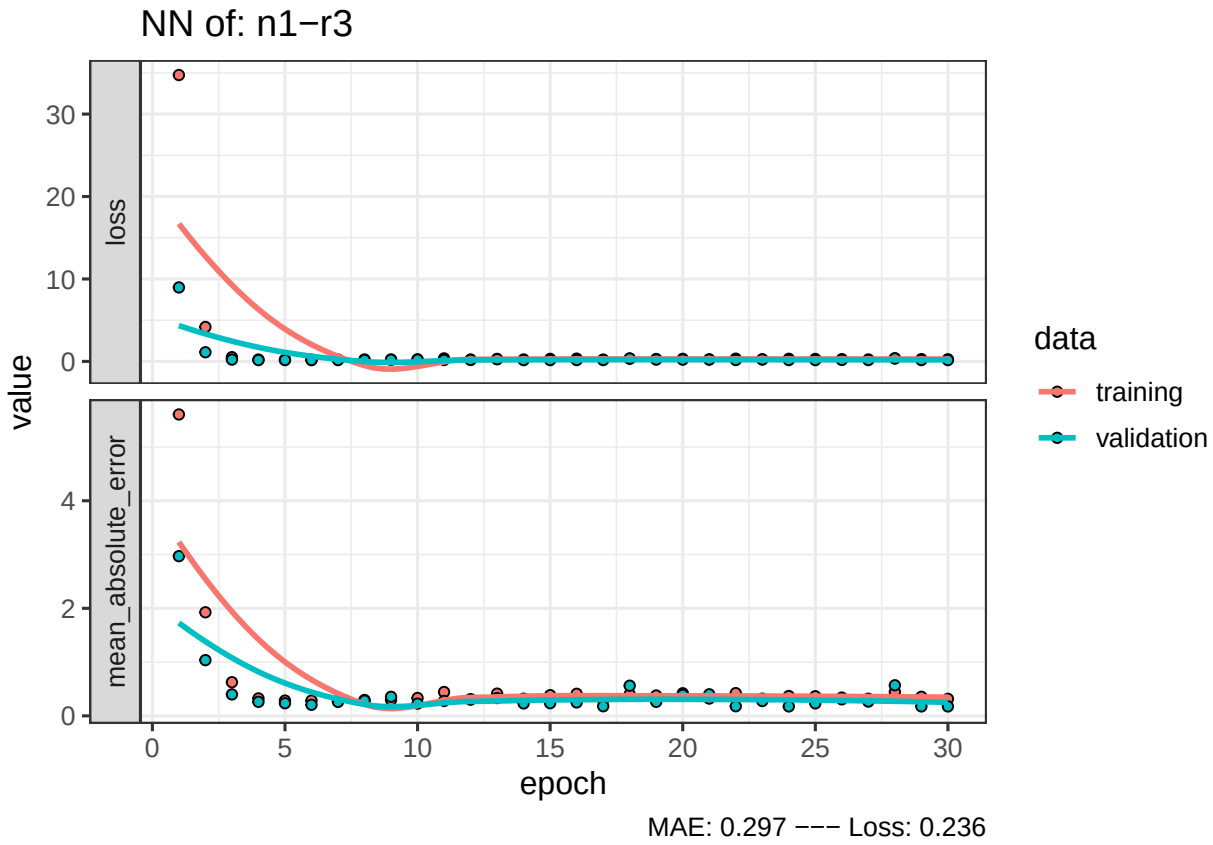



```
## MAE:  0.815325
## Loss:  0.6606796
##
## [1] "========== n1-r3 =========="
```

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

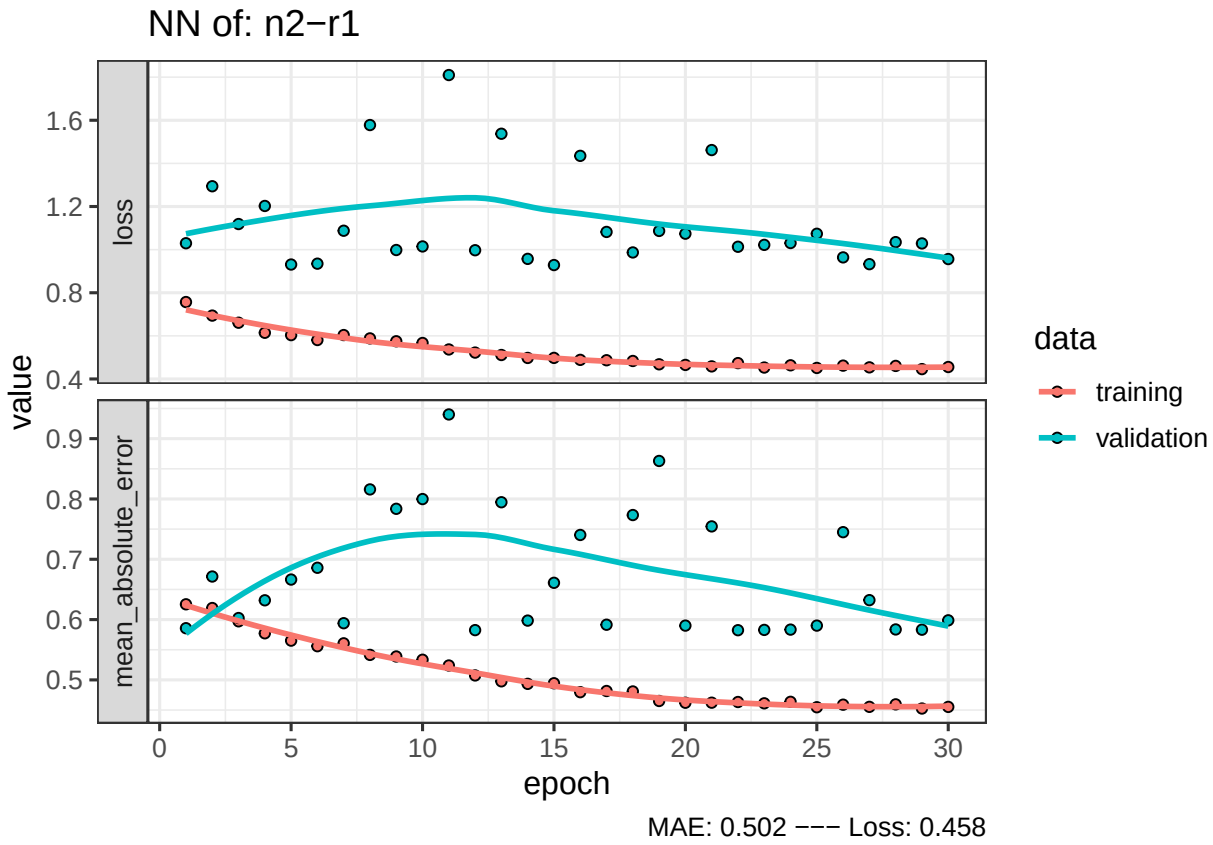



```
## `geom_smooth()` using formula 'y ~ x'
```

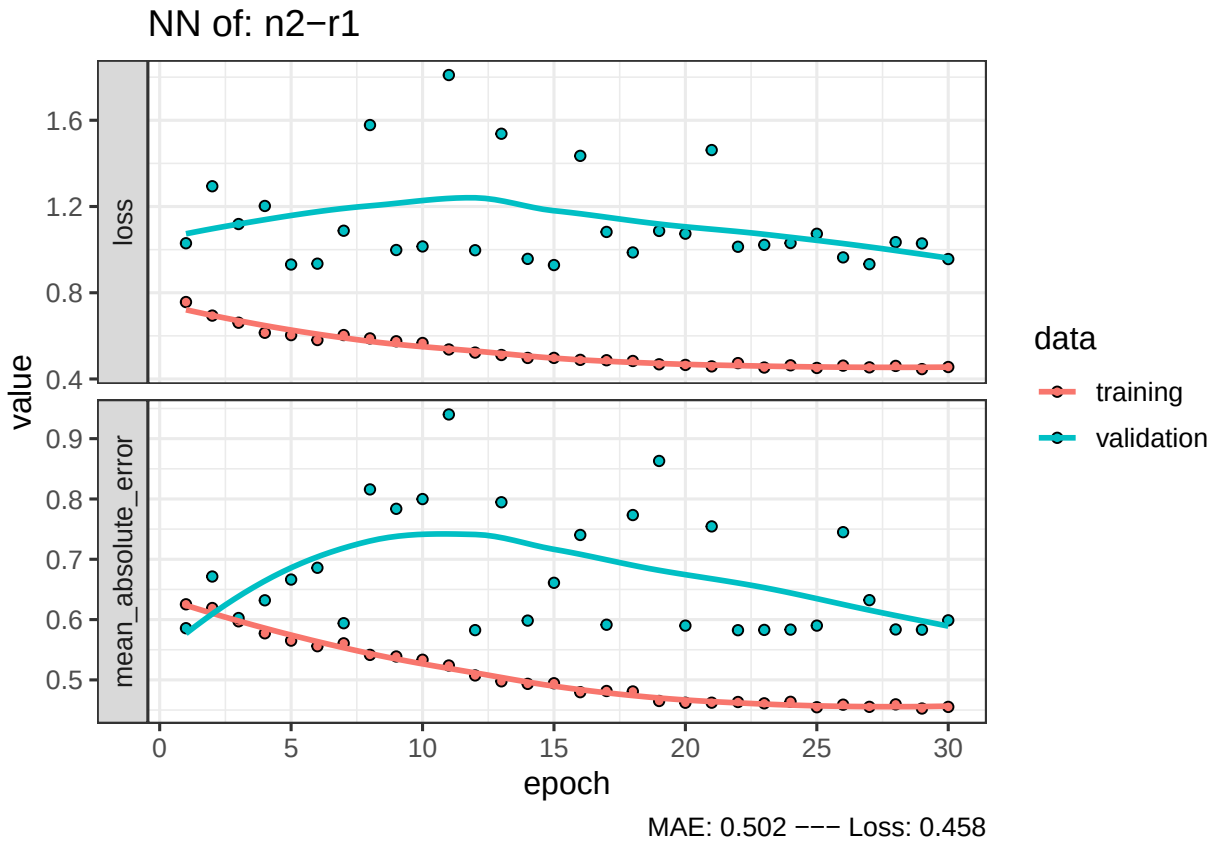



```
## MAE:  0.297305
## Loss:  0.2364705
##
## [1] "========== n2-r1 =========="
```

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

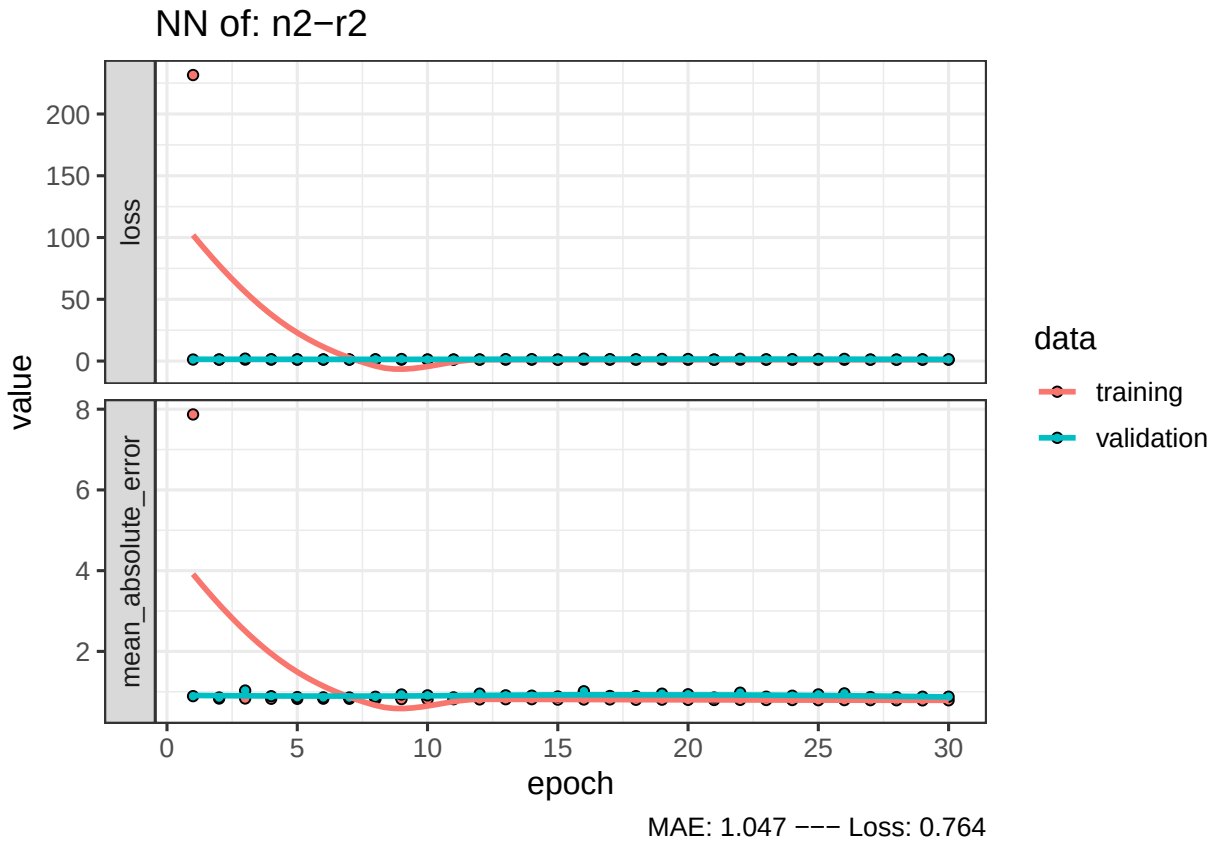



```
## `geom_smooth()` using formula 'y ~ x'
```

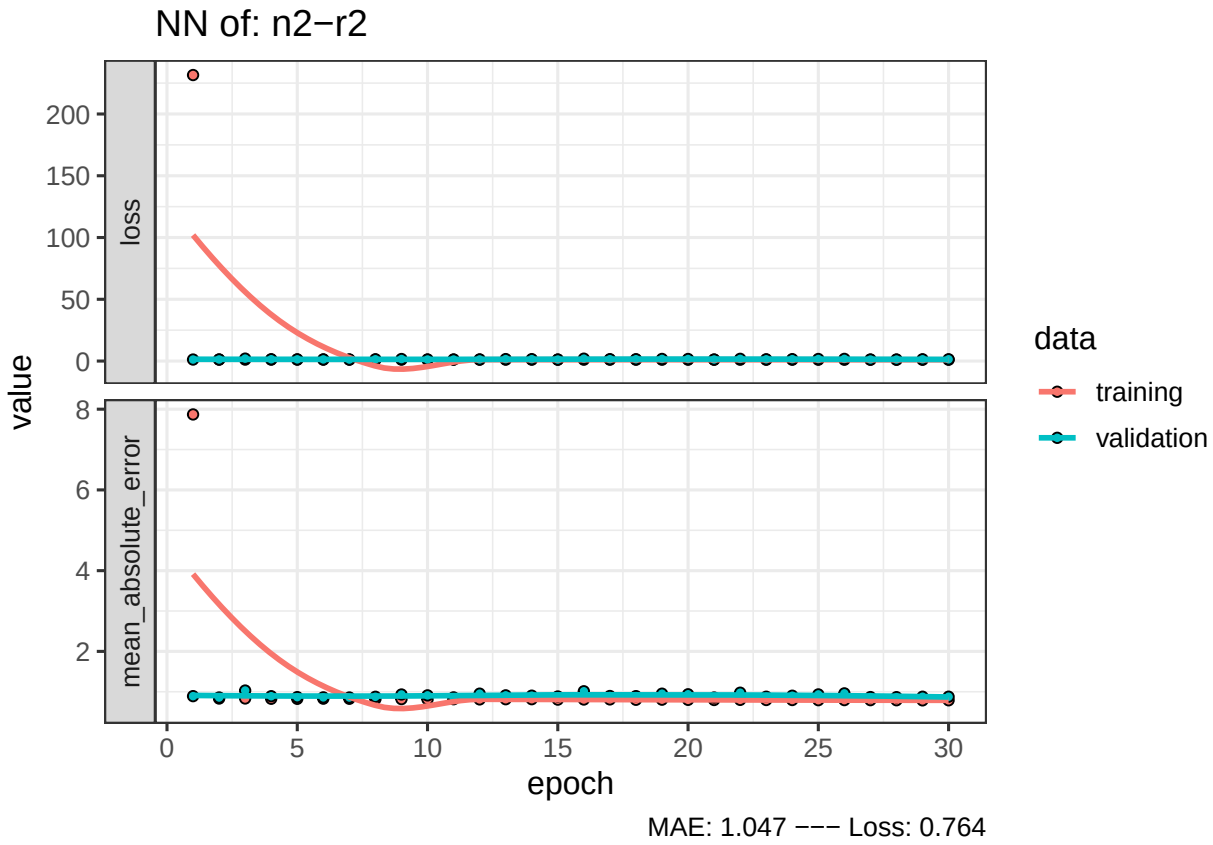



```
## MAE:  0.5021501
## Loss:  0.4582768
##
## [1] "========== n2-r2 =========="
```

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

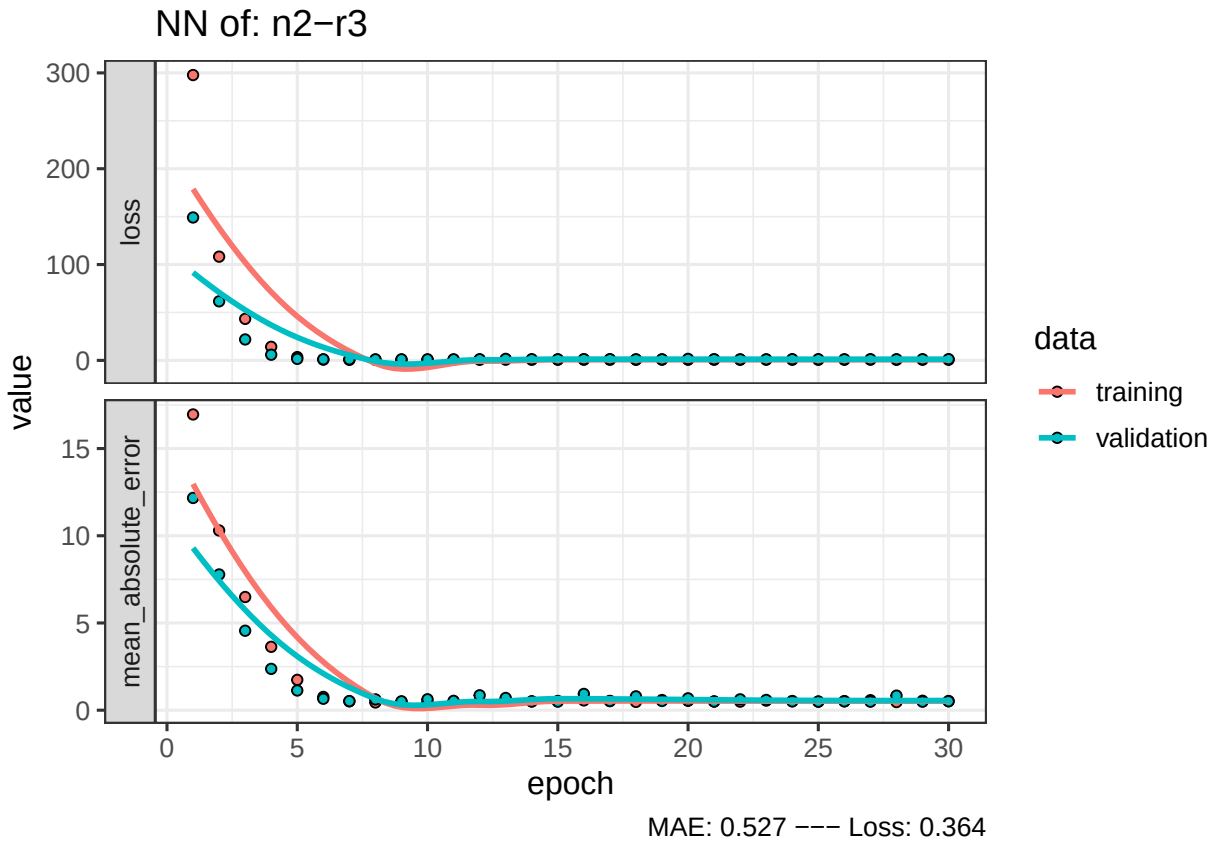



```
## `geom_smooth()` using formula 'y ~ x'
```

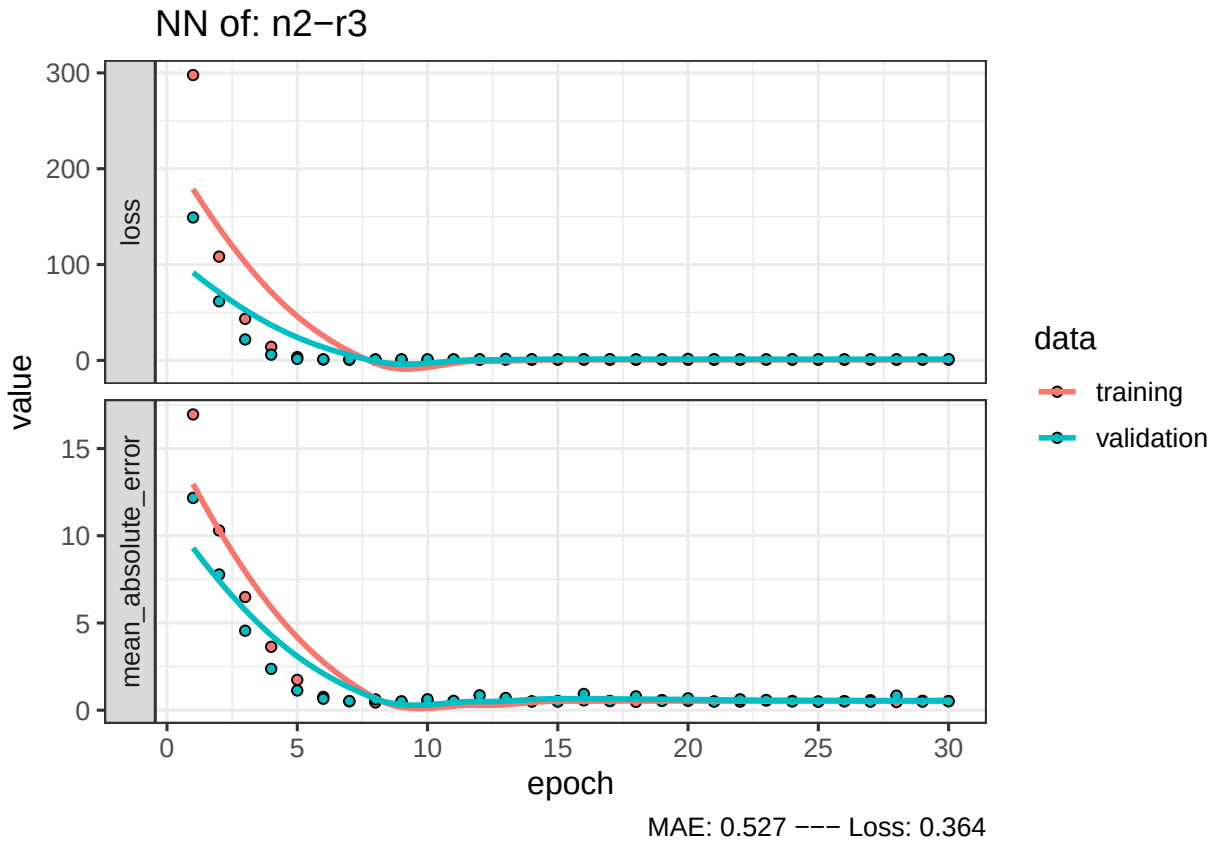



```
## MAE:  1.04709
## Loss:  0.763576
##
## [1] "========== n2-r3 =========="
```

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

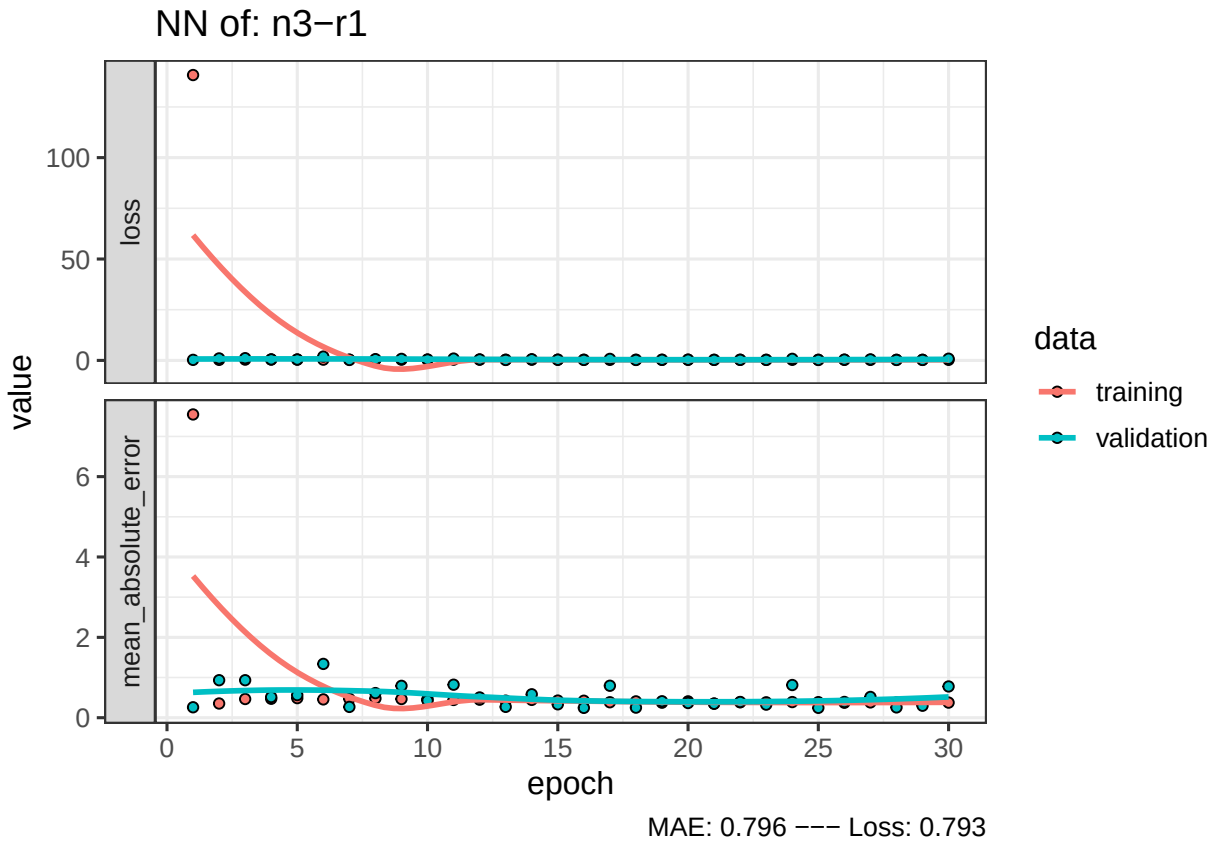



```
## `geom_smooth()` using formula 'y ~ x'
```

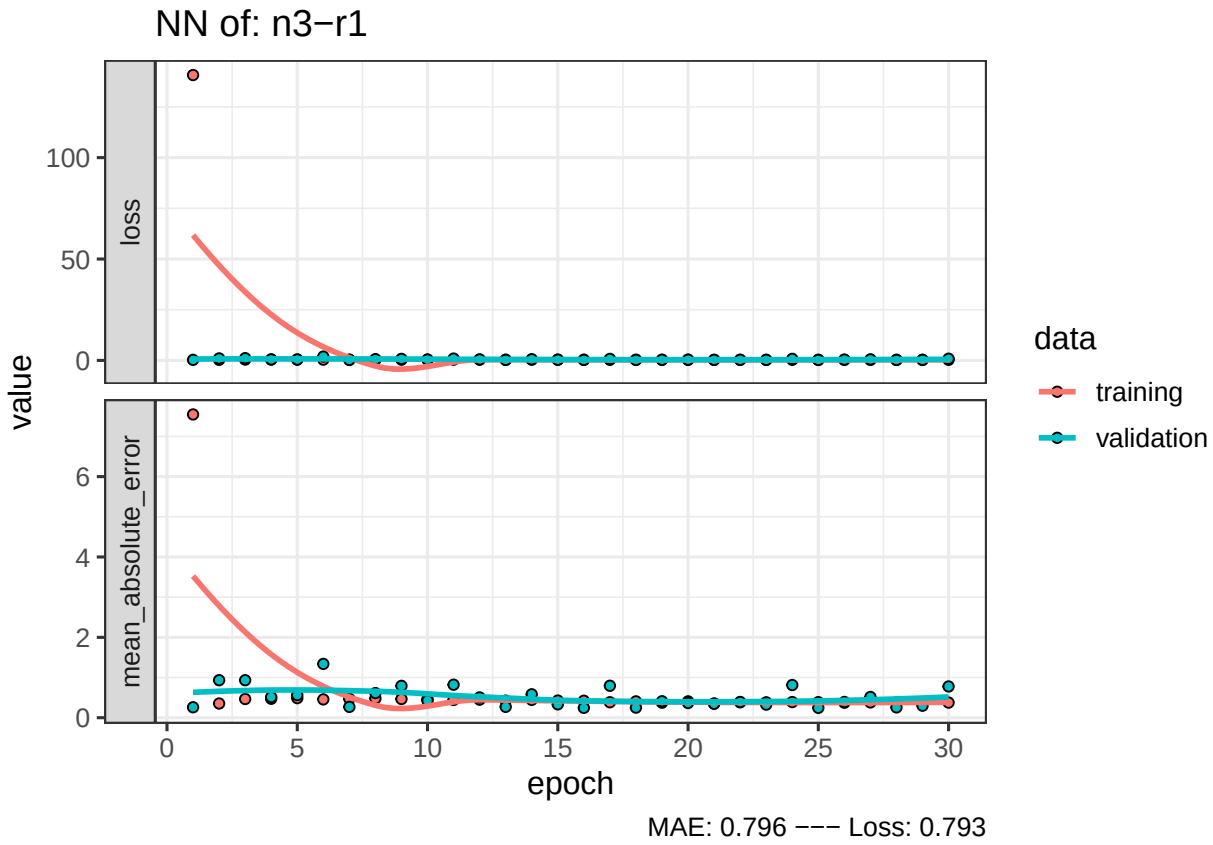



```
## MAE:  0.5272431
## Loss:  0.364036
##
## [1] "========== n3-r1 =========="
```

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

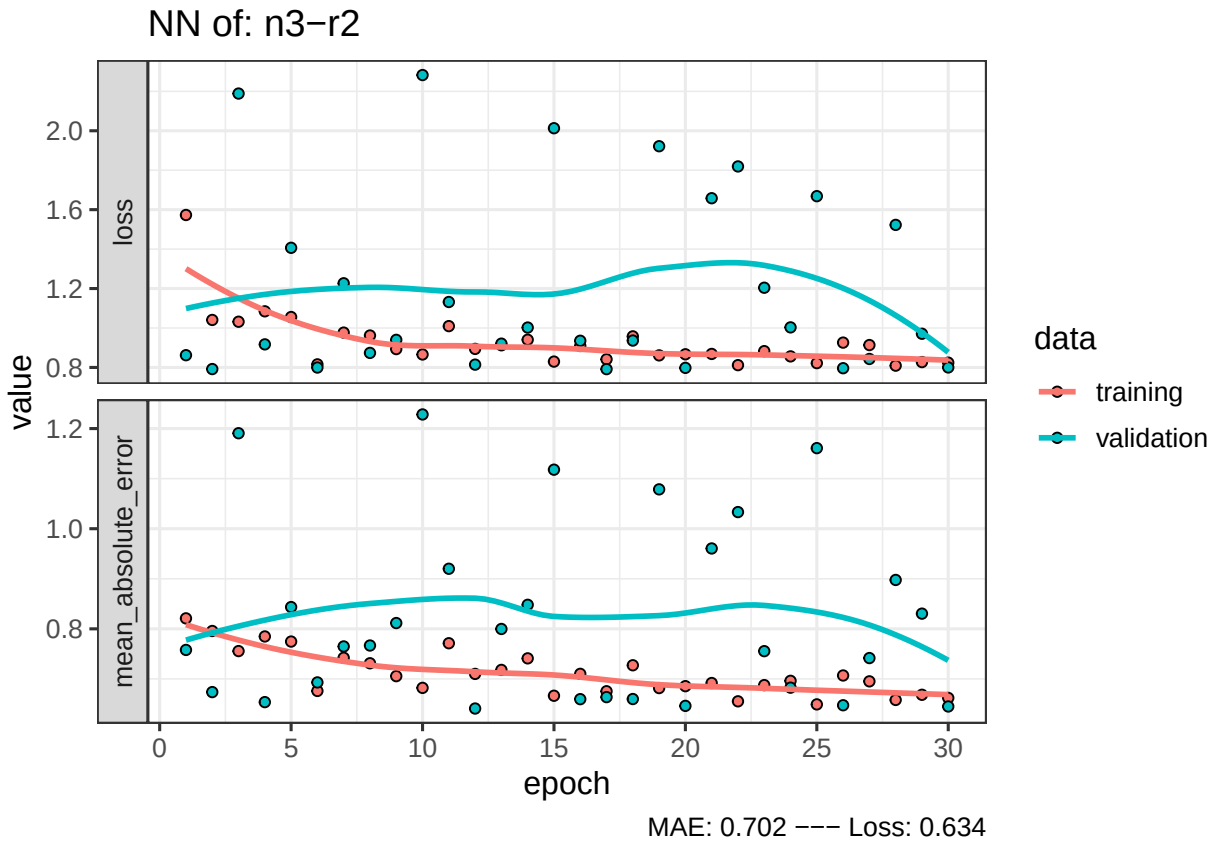



```
## `geom_smooth()` using formula 'y ~ x'
```

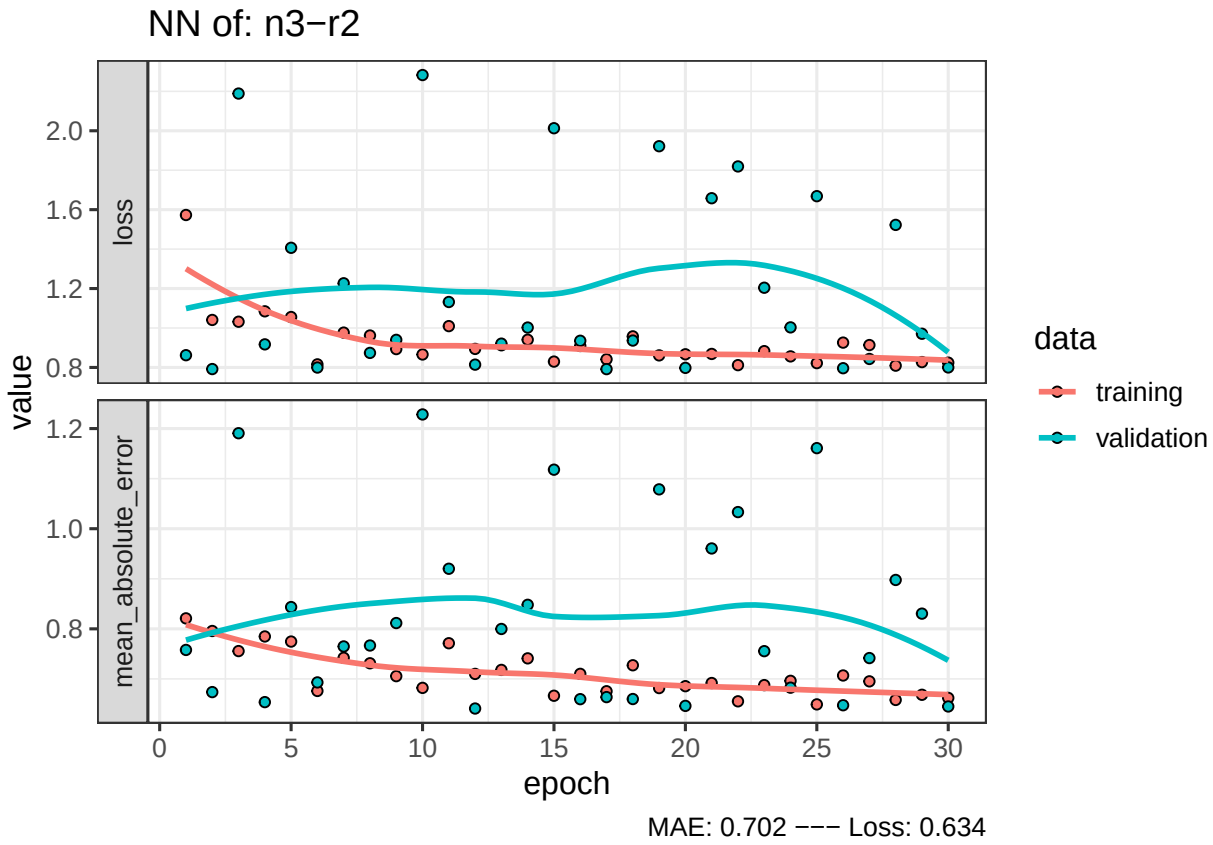



```
## MAE:  0.796416
## Loss:  0.7929425
##
## [1] "========== n3-r2 =========="
```

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

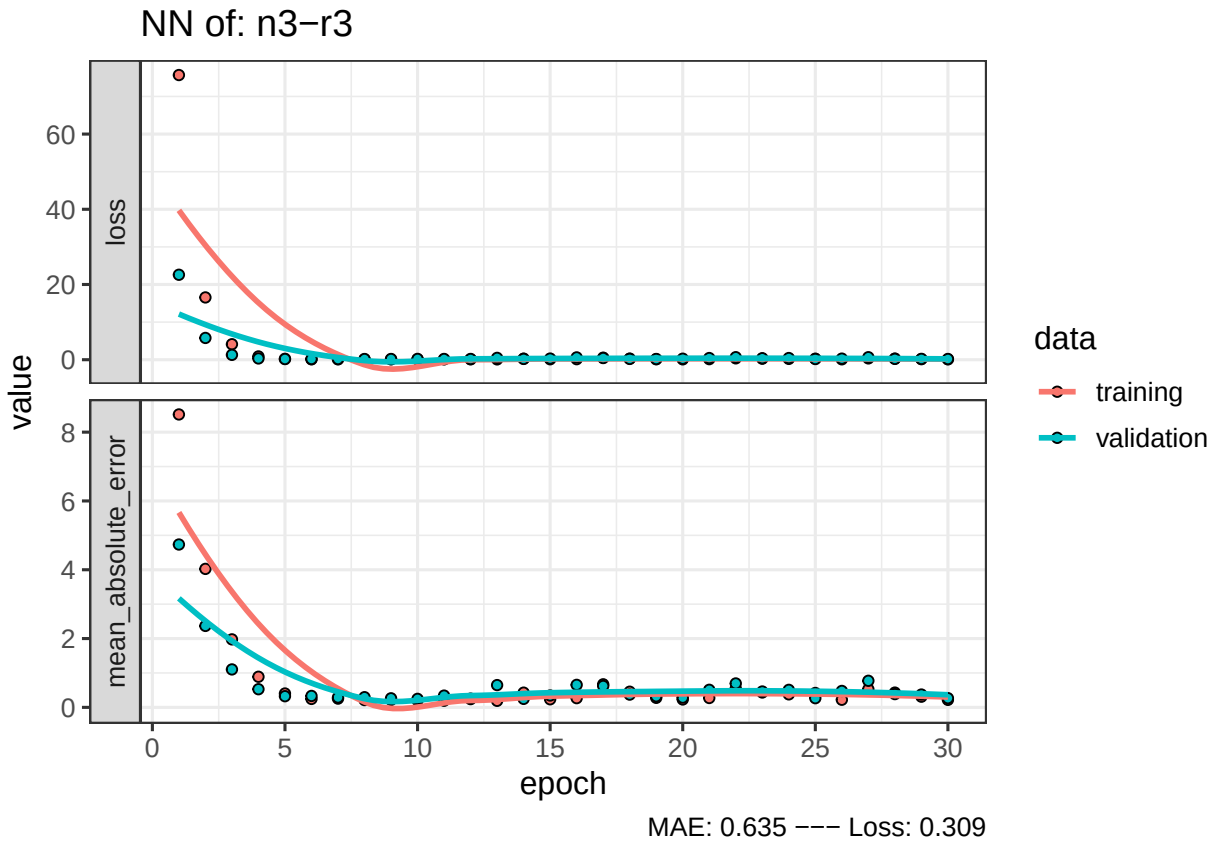



```
## `geom_smooth()` using formula 'y ~ x'
```

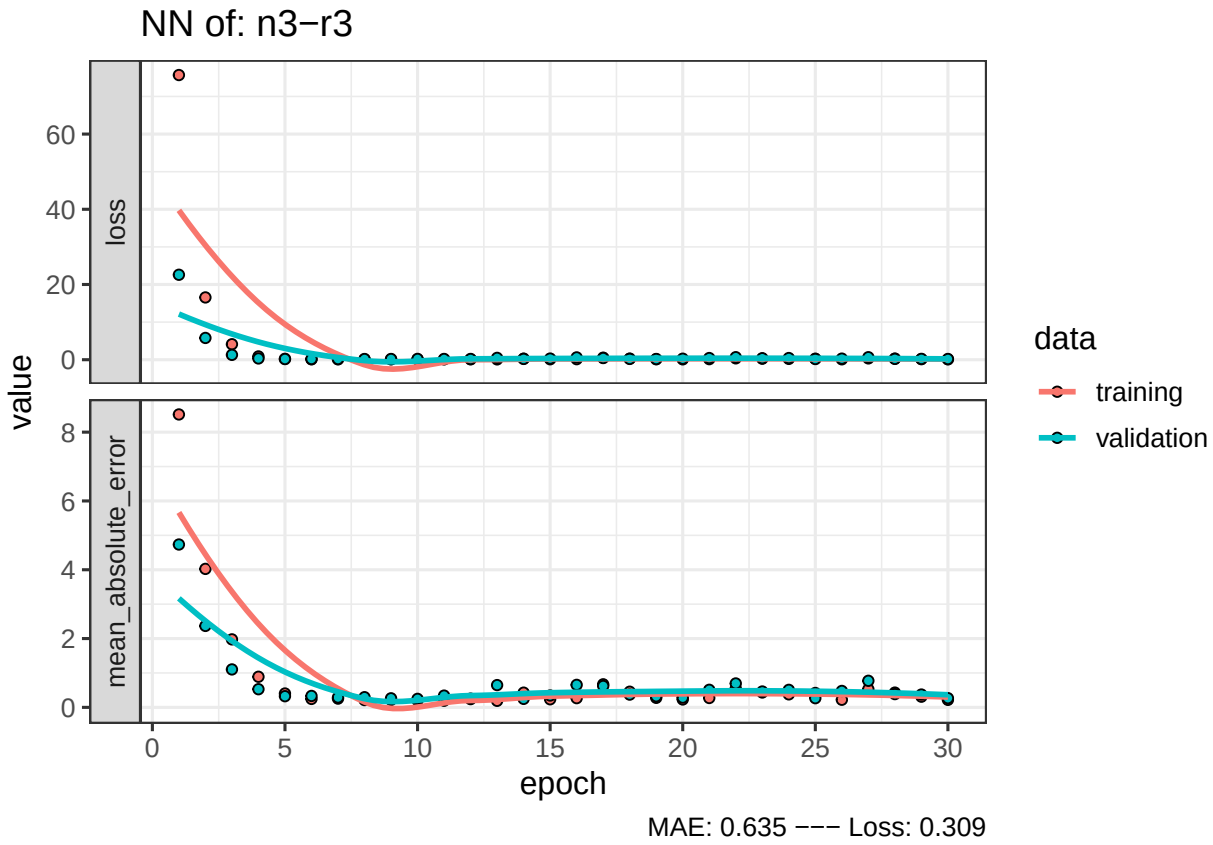



```
## MAE:  0.7023411
## Loss:  0.6338332
##
## [1] "========== n3-r3 =========="
```

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

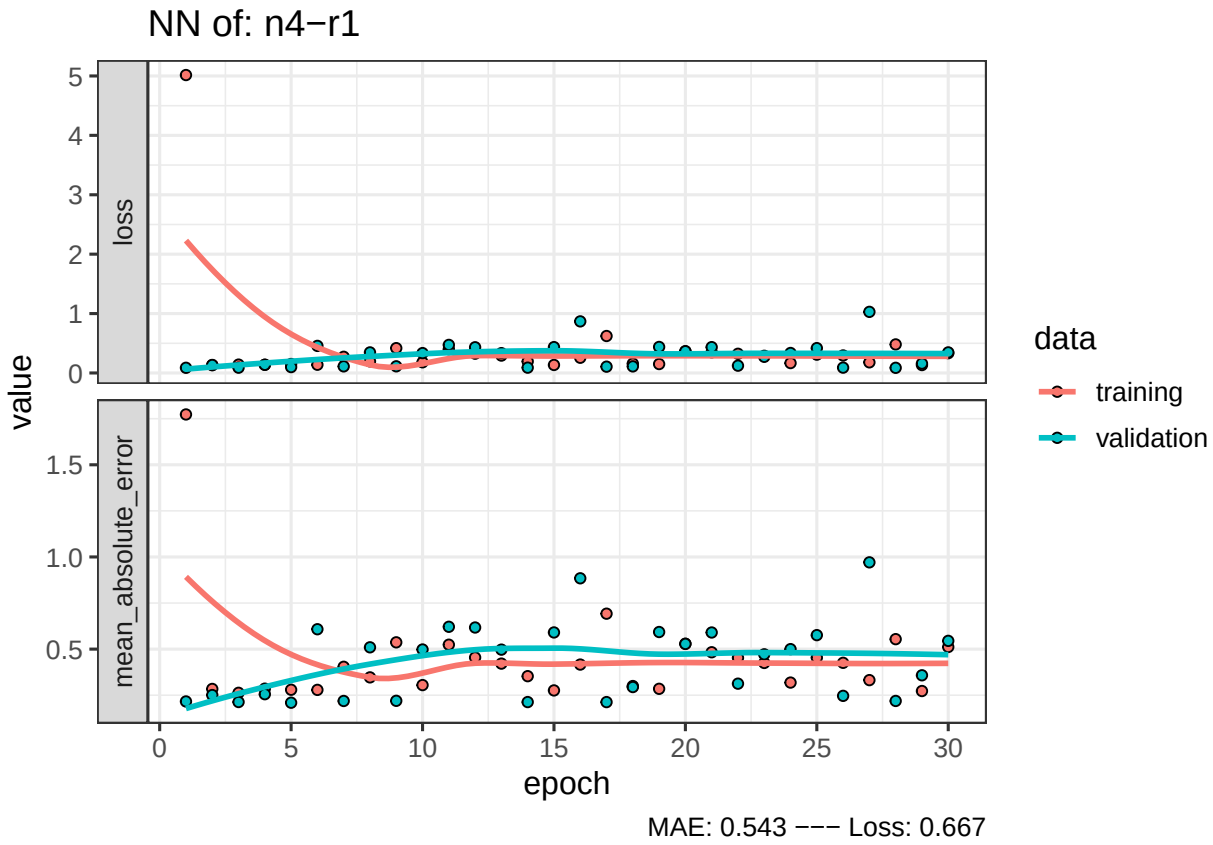



```
## `geom_smooth()` using formula 'y ~ x'
```

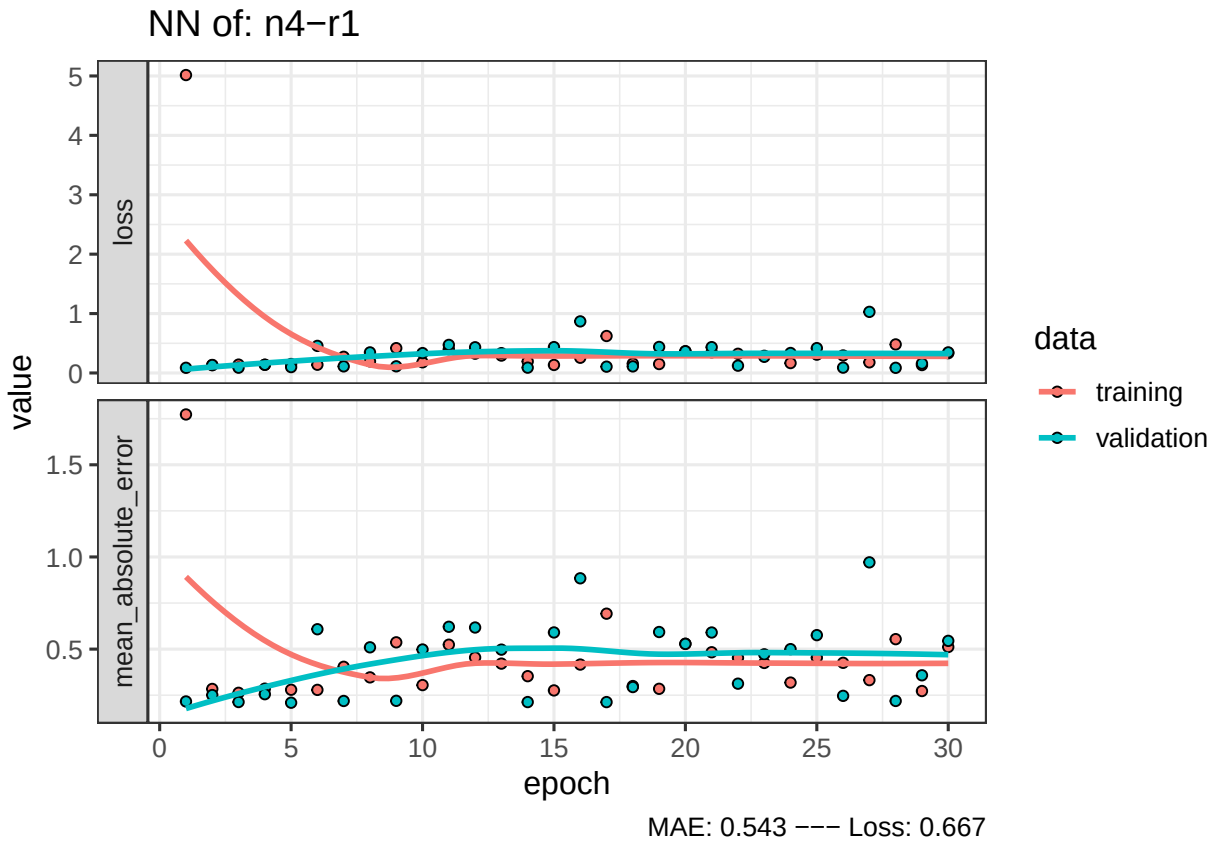



```
## MAE:  0.6350197
## Loss:  0.309325
##
## [1] "========== n4-r1 =========="
```

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

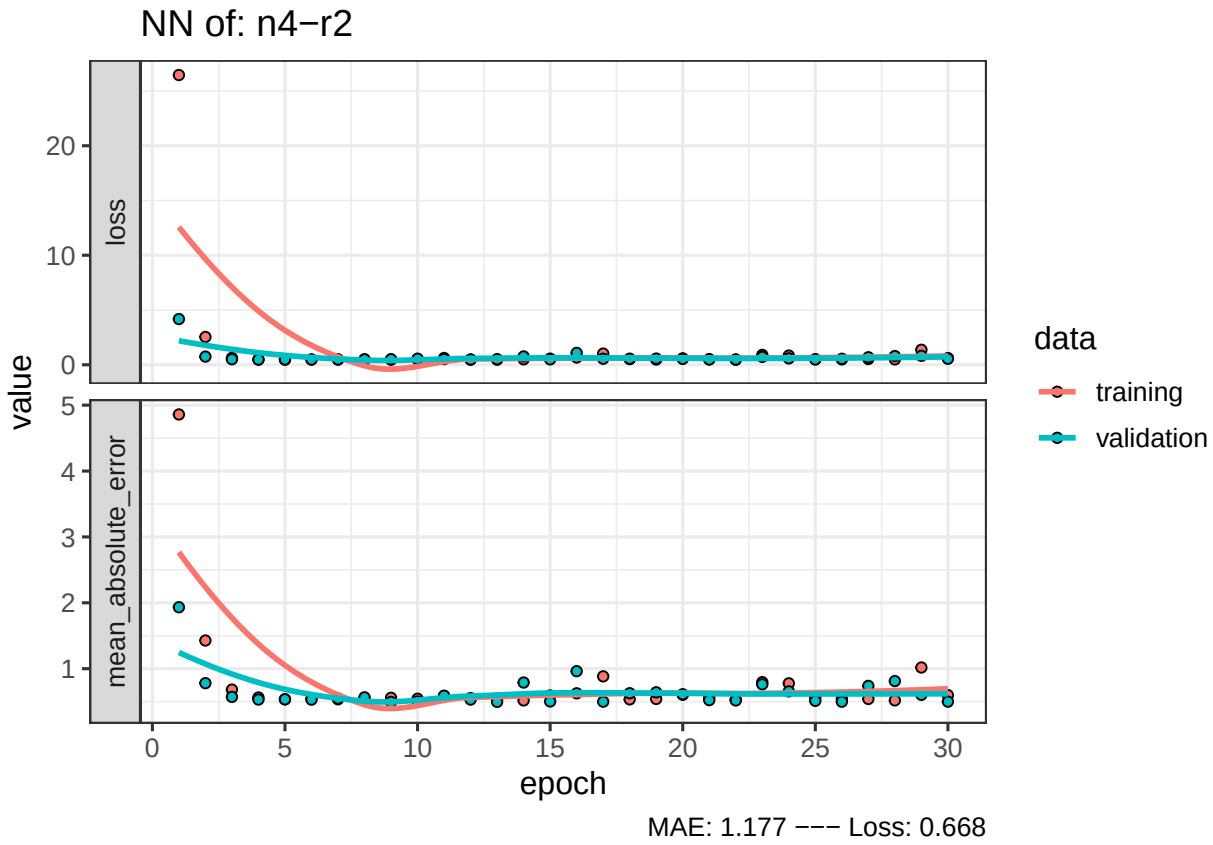



```
## `geom_smooth()` using formula 'y ~ x'
```

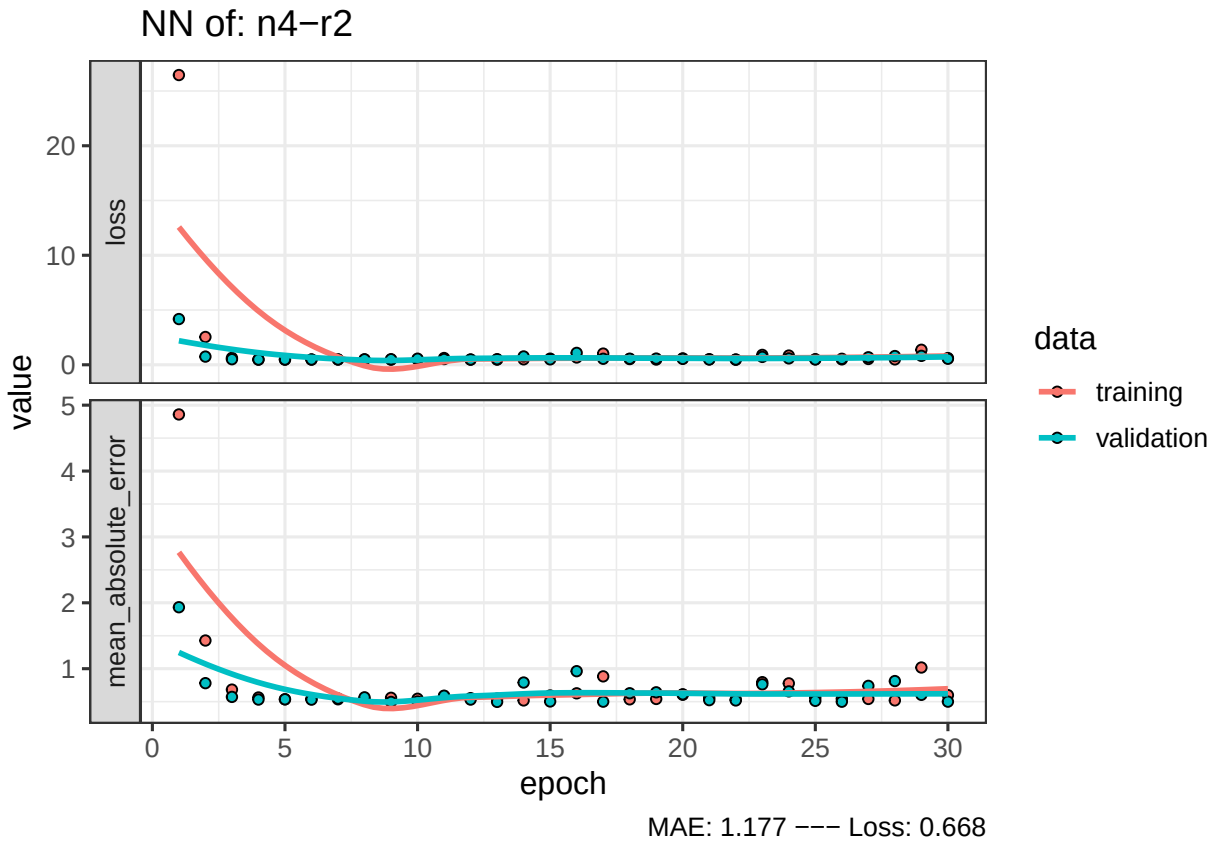



```
## MAE:  0.5431977
## Loss:  0.6666863
##
## [1] "========== n4-r2 =========="
```

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

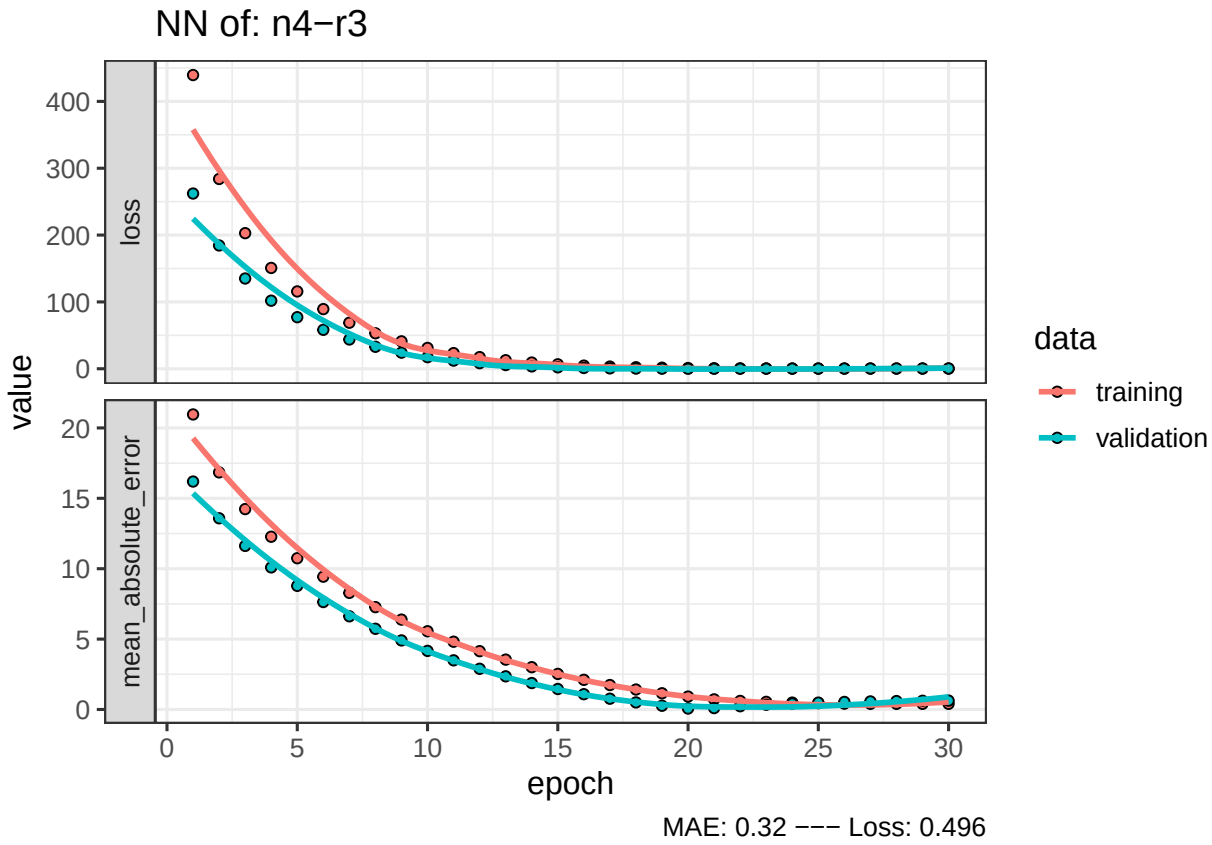



```
## `geom_smooth()` using formula 'y ~ x'
```

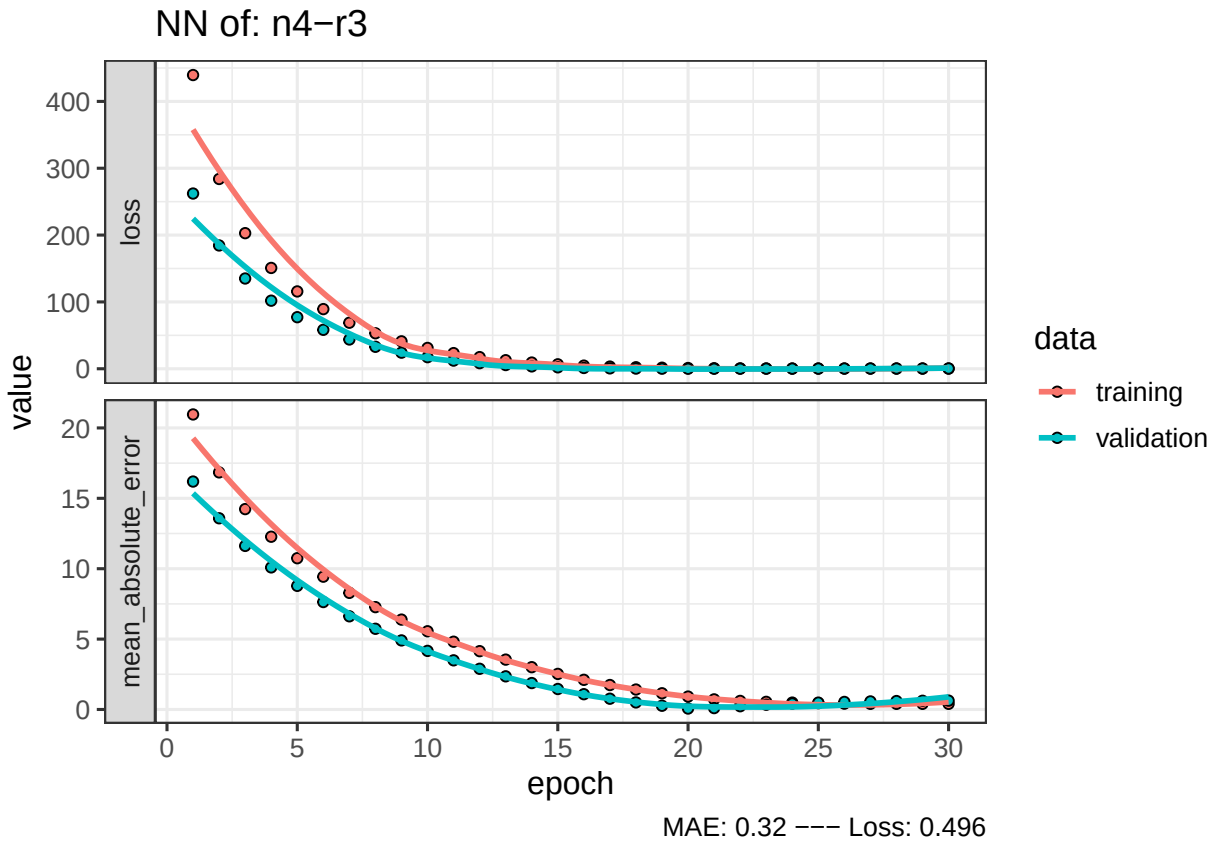



```
## MAE:  1.177381
## Loss:  0.6681005
##
## [1] "========== n4-r3 =========="
```

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

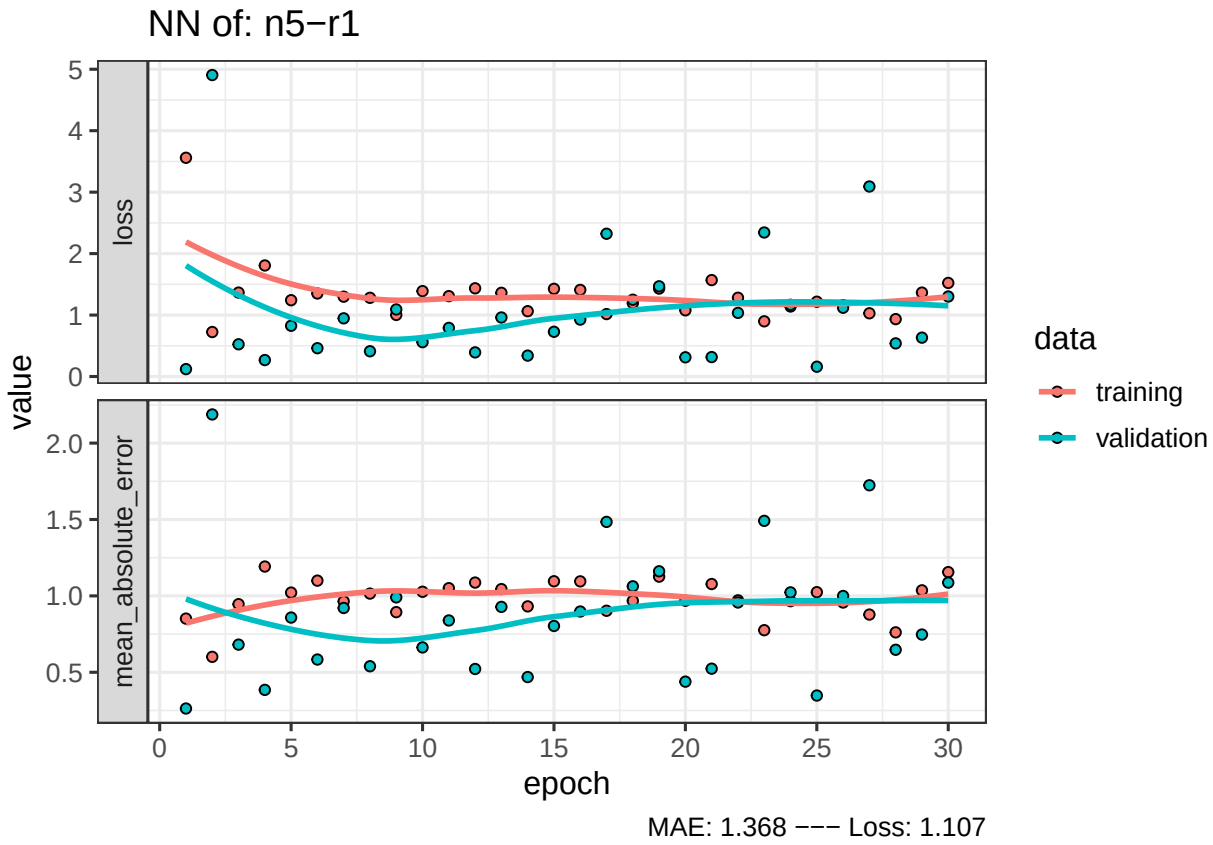



```
## `geom_smooth()` using formula 'y ~ x'
```

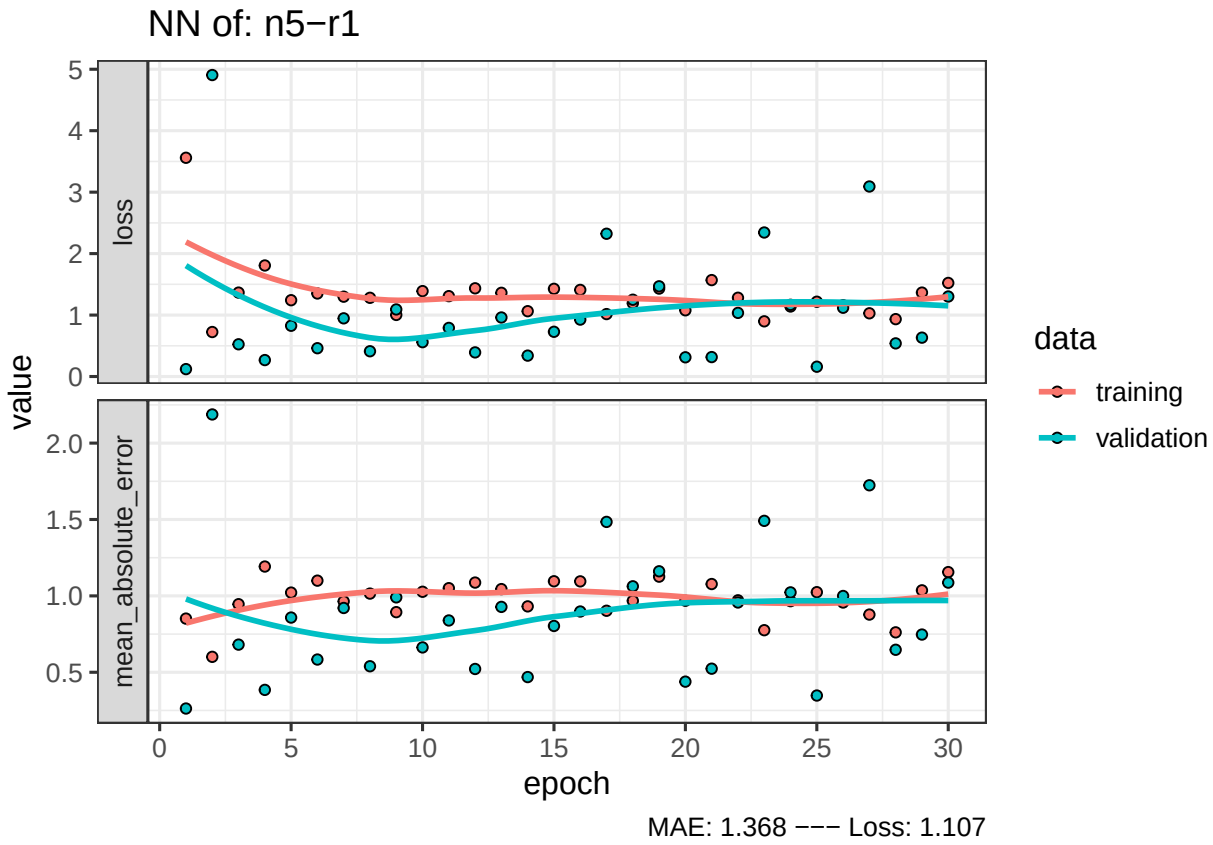



```
## MAE:  0.3195457
## Loss:  0.4957582
##
## [1] "========== n5-r1 =========="
```

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

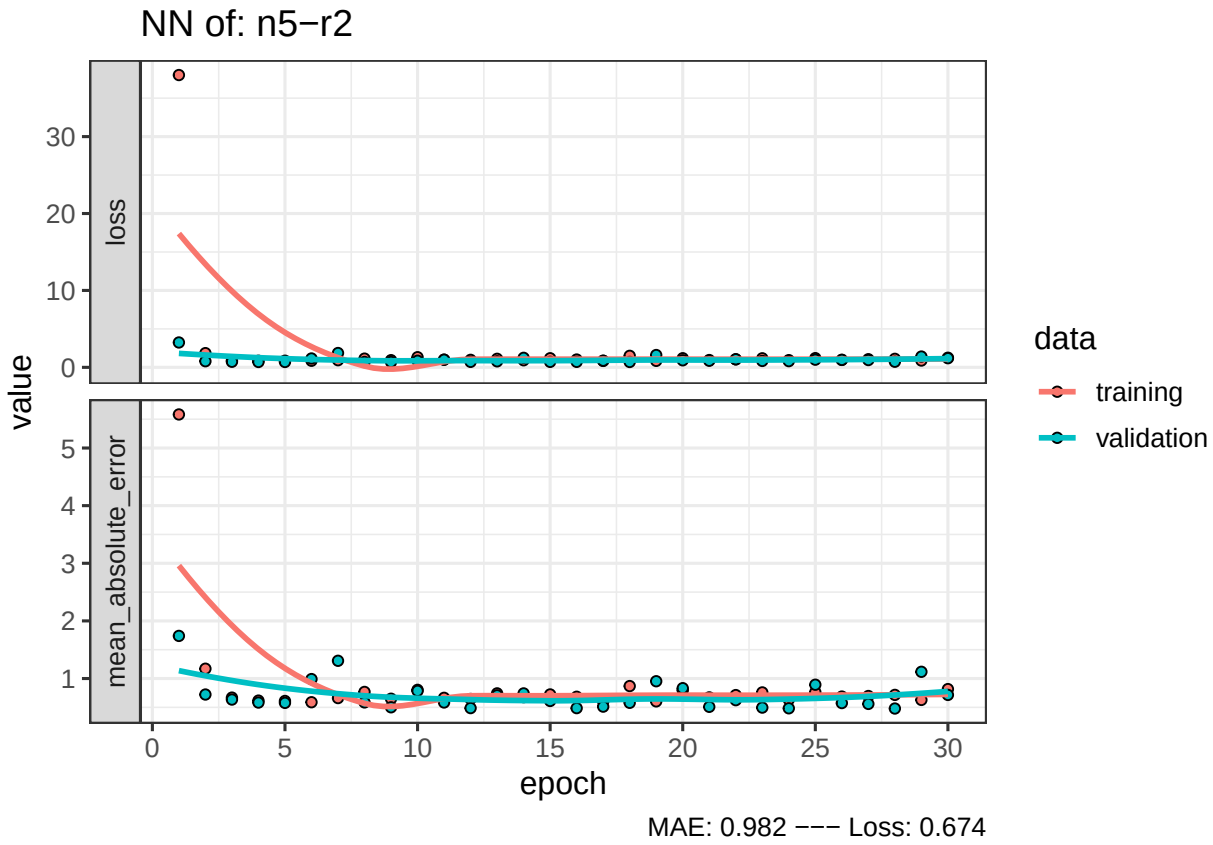



```
## `geom_smooth()` using formula 'y ~ x'
```

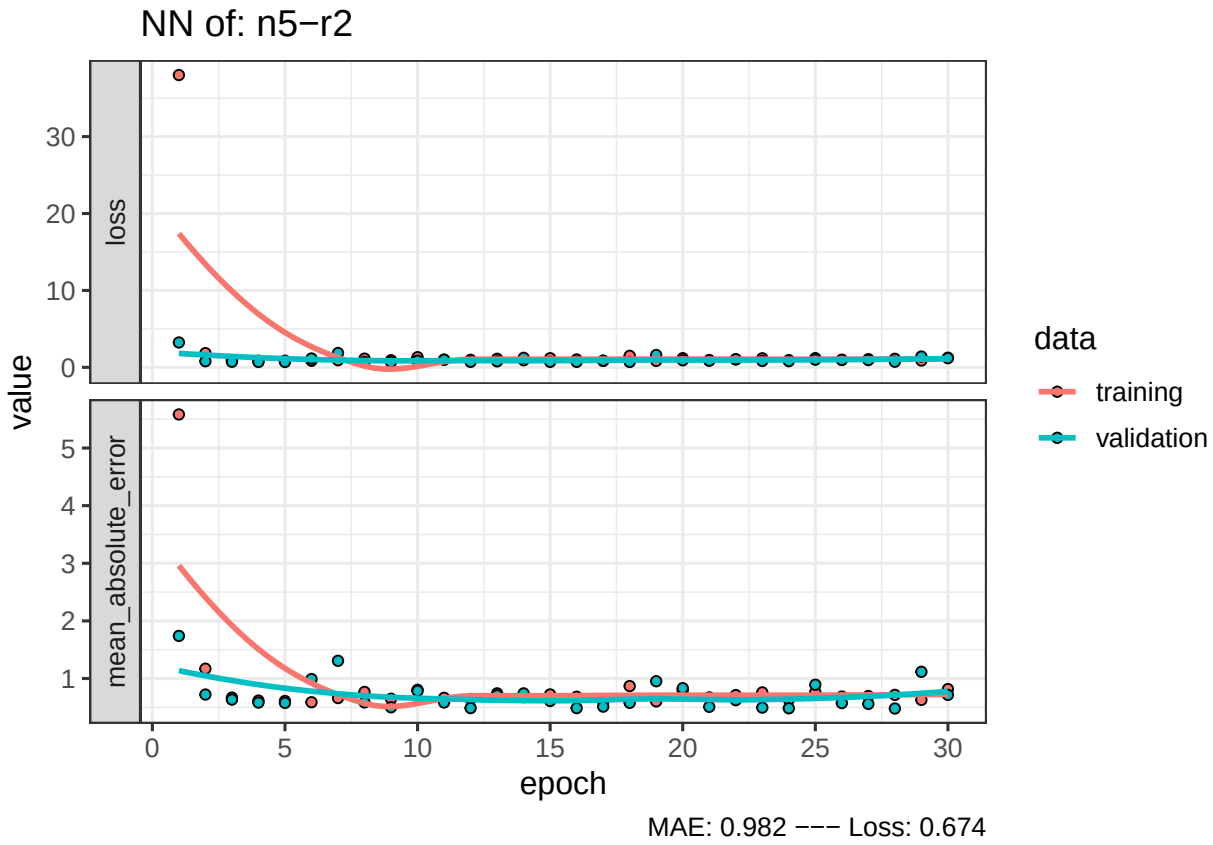



```
## MAE:  1.367847
## Loss:  1.106661
##
## [1] "========== n5-r2 =========="
```

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

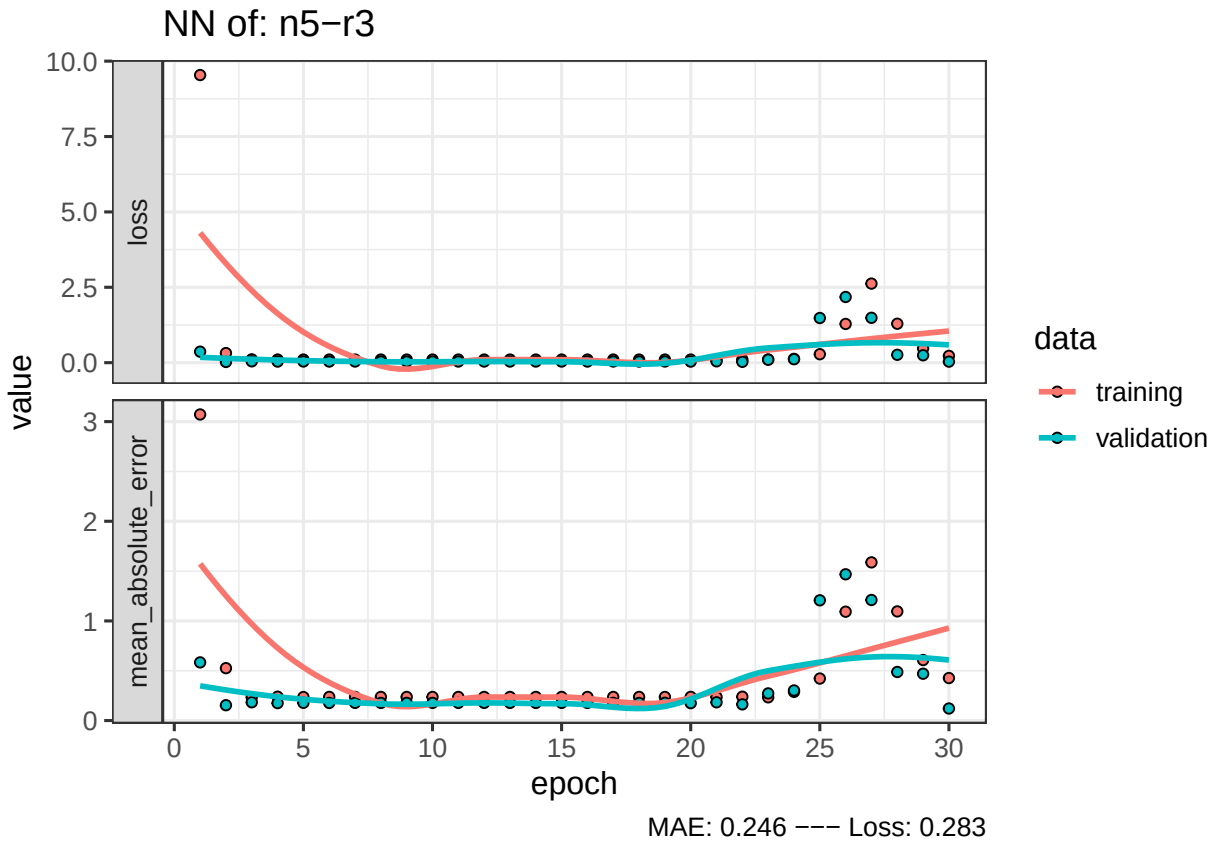



```
## `geom_smooth()` using formula 'y ~ x'
```

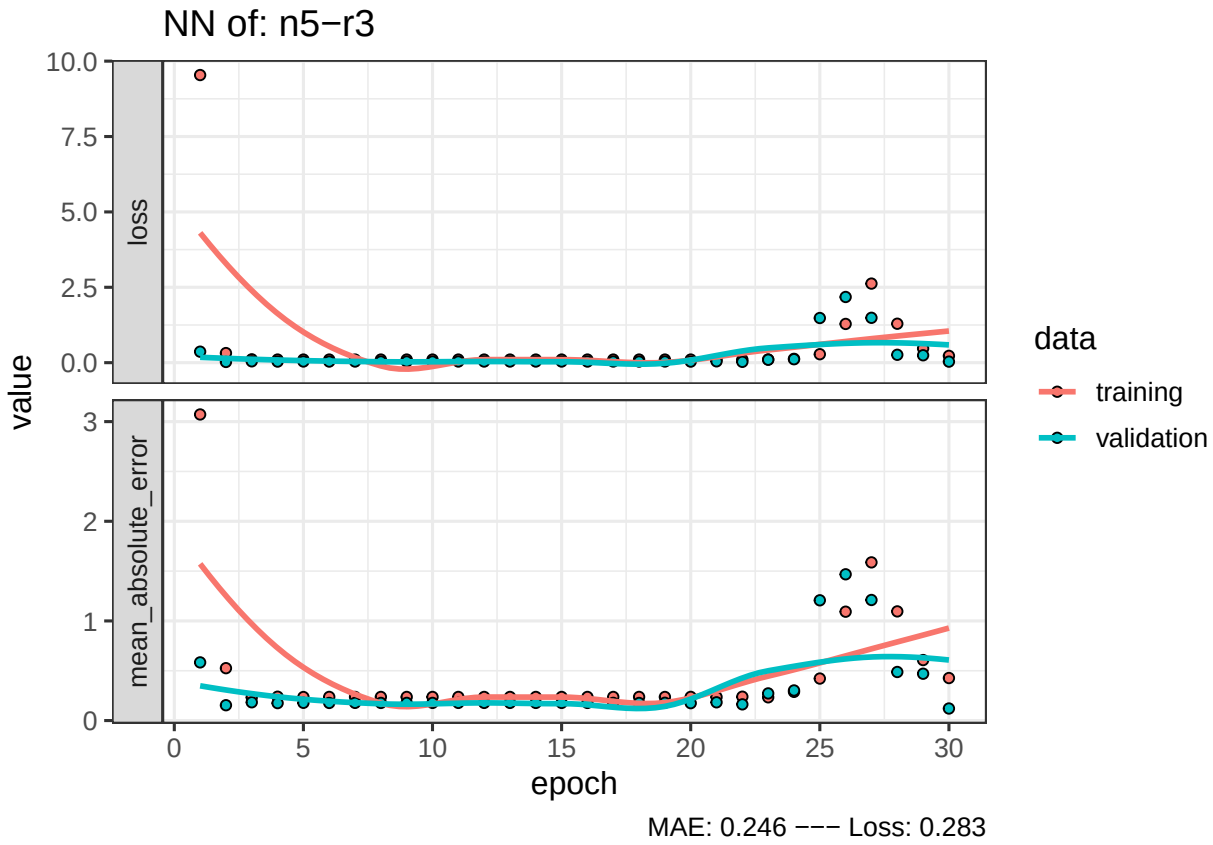



```
## MAE:  0.9816905
## Loss:  0.6742522
##
## [1] "========== n5-r3 =========="
```

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

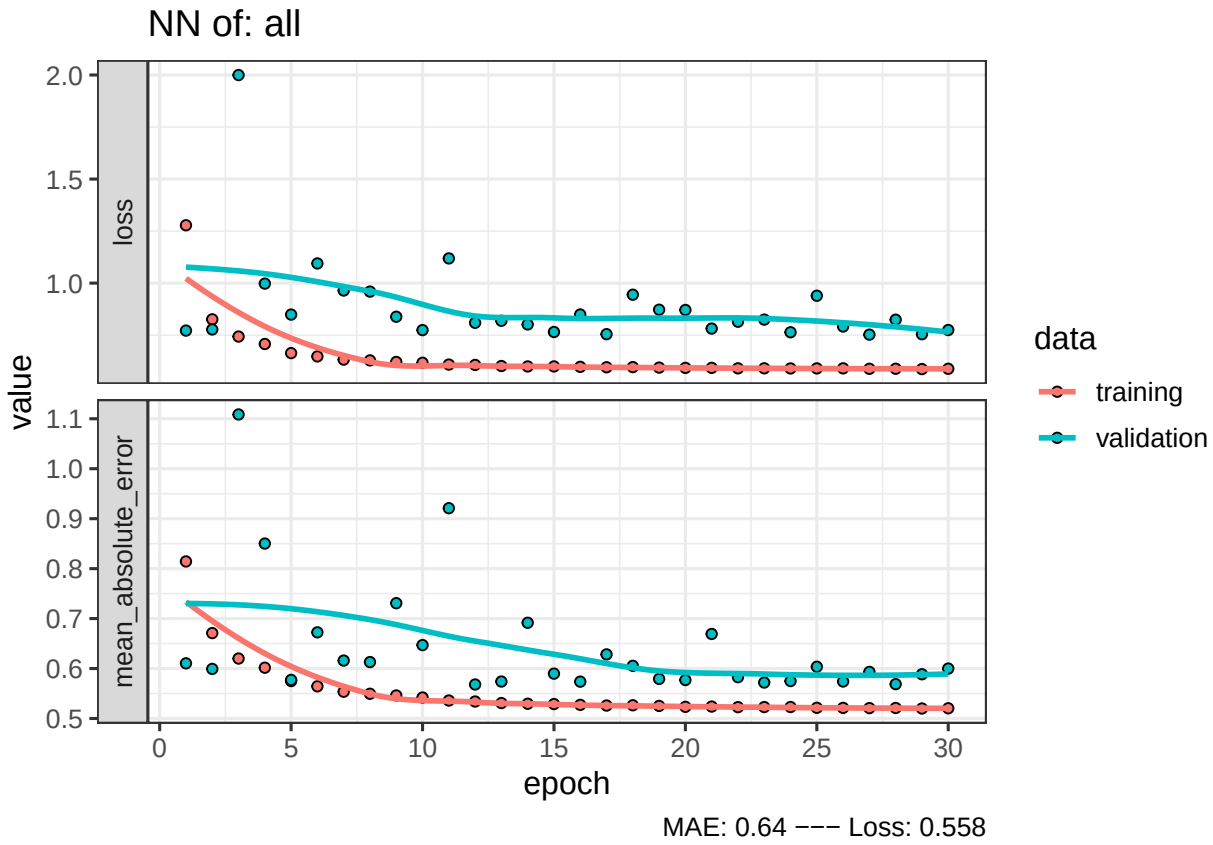



```
## `geom_smooth()` using formula 'y ~ x'
```

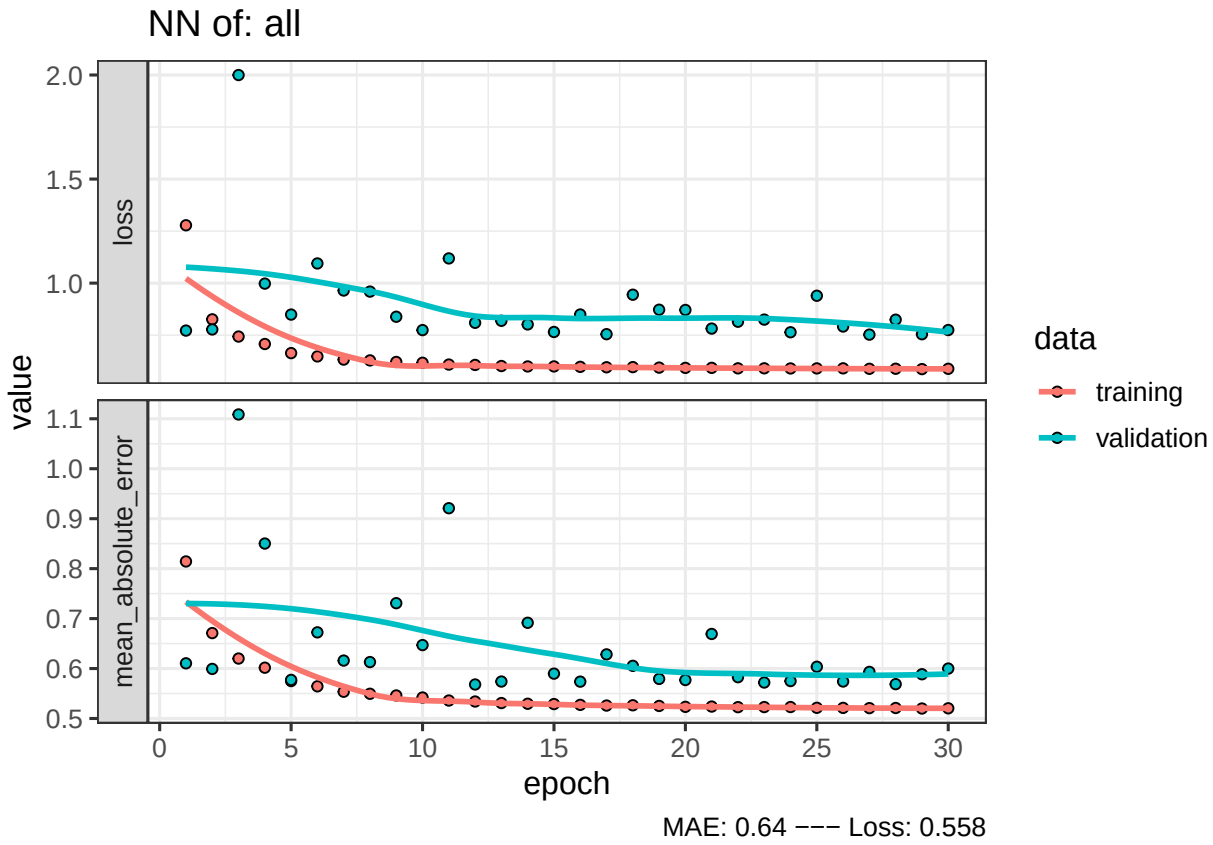



```
## MAE:  0.2455914
## Loss:  0.282955
##
## [1] "========== all =========="
```

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




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




```
## MAE:  0.6396843
## Loss:  0.5579407
```

Comparison between the models

```
concat = c()

for (n in unique(ds$neighbourhood_group))
{
  concat = c(concat,n)
}
for (n in unique(ds$neighbourhood_group))
{
  for(r in unique(ds$room_type))
  {
    concat = c(concat, paste0(n,"/",r))

  }
}
concat = c(concat,"All")

n = names(nn)
```

```
cols = vector("list")
cols[["Subset"]] = concat
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)
#knitr::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 Random.Forest
## 1                      Brooklyn         0.4747922     0.4788720    0.45401237
## 2                     Manhattan         0.7776761     0.7851166    0.73559511
## 3                        Queens         0.4202809     0.4228161    0.39627682
## 4                Staten Island         0.3467308     0.3471879    0.33675009
## 5                         Bronx         0.2741482     0.2744778    0.27152894
## 6          Brooklyn/Private room         0.1858138     0.1806116    0.19321403
## 7       Brooklyn/Entire home/apt         0.7533368     0.7483236    0.79295154
## 8           Brooklyn/Shared room         0.2639579     0.2505339    0.24216019
## 9         Manhattan/Private room         0.4445611     0.3753113    0.38059182
## 10     Manhattan/Entire home/apt         0.9361941     0.9585893    0.97707056
## 11         Manhattan/Shared room         0.4904792     0.4992158    0.55878634
## 12           Queens/Private room         0.1680487     0.1600965    0.16300480
## 13        Queens/Entire home/apt         0.6992986     0.6995247    0.71164555
## 14            Queens/Shared room         0.6218959     0.6218172    0.63174807
## 15    Staten Island/Private room         0.2640100     0.2339327    0.22699579
## 16 Staten Island/Entire home/apt         0.9903964     1.1892994    1.04889417
## 17     Staten Island/Shared room         0.1199132     0.2241880    0.07268596
## 18            Bronx/Private room         0.1475827     0.1611146    0.13436599
## 19         Bronx/Entire home/apt         0.5627967     0.7622148    0.66205900
## 20            Bronx/Shared room         0.2071295     0.1939584    0.17492181
## 21                         All         0.6167807     0.6149423    0.58299244
##    Ranger.Random.Forest Neural.Networks
## 1             0.4517082       0.5105357
## 2             0.7401571       0.8534025
## 3             0.3953393       0.4212552
## 4             0.3476114       0.3655377
## 5             0.2725558       0.6189409
```

```
## 6             0.1937730       0.2255452
## 7             0.7956161       0.8153250
## 8             0.2385574       0.2973050
## 9             0.3776716       0.5021502
## 10            0.9769060       1.0470899
## 11            0.5606141       0.5272432
## 12            0.1619174       0.7964161
## 13            0.7118833       0.7023412
## 14            0.6315979       0.6350197
## 15            0.2294381       0.5431977
## 16            1.0661382       1.1773806
## 17            0.2287501       0.3195457
## 18            0.1341032       1.3678471
## 19            0.6617712       0.9816905
## 20            0.1738229       0.2455915
## 21            0.5552083       0.6396844
```

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 =
```

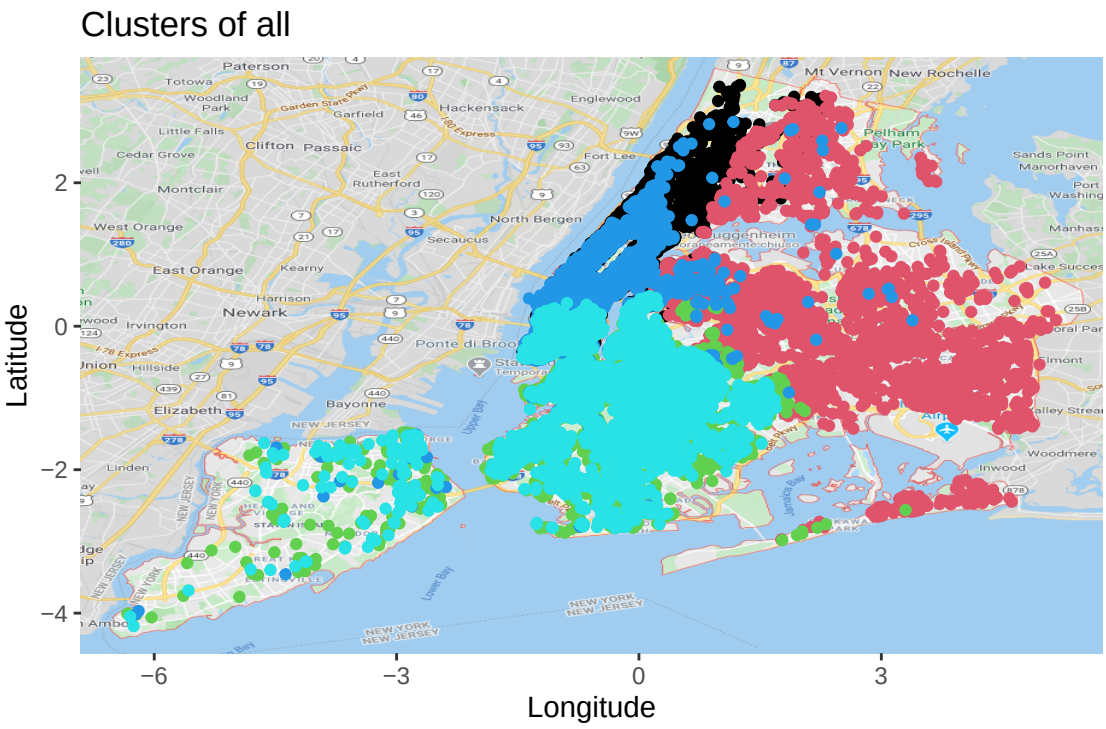
```
  count = 1
  l$clust_plot = vector("list")
  for(el in clust)
  {
    myplot = myplot+ geom_point(data = el, aes(y = latitude, x = longitude),color= count)

    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)
    #print(paste0("=== clust: ",count,"==="))
    #print(summary(el))
    #print("==========")

    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
```

```r
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
```

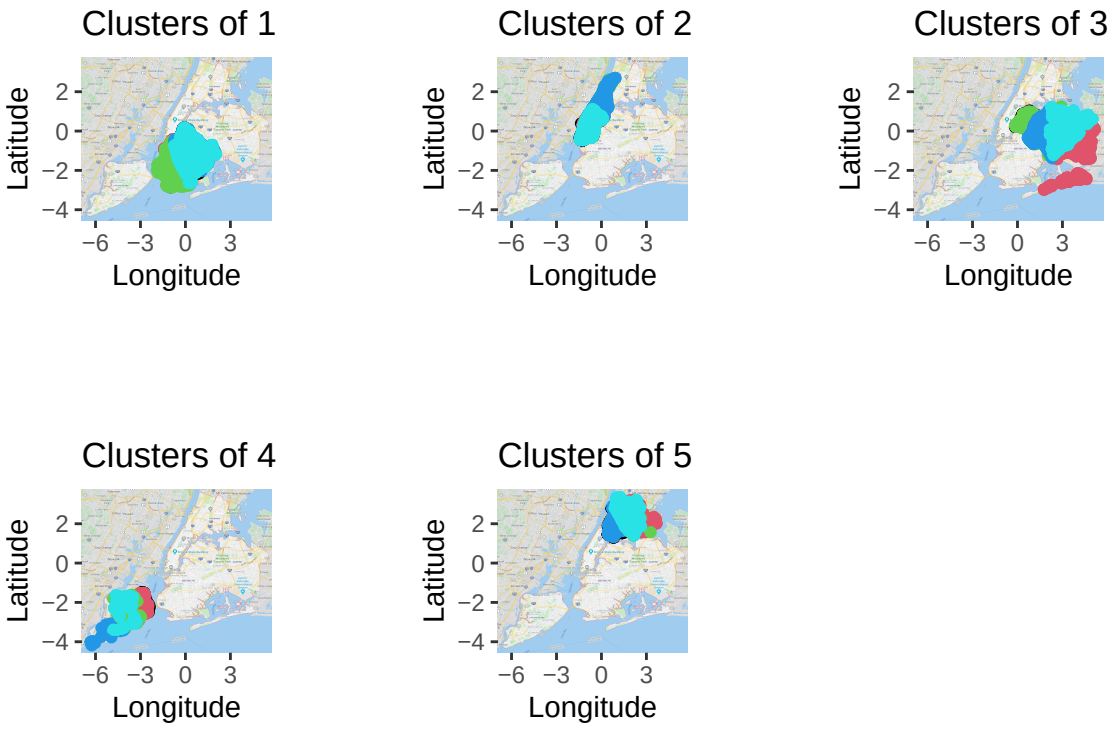
```
## $`1`
##     neighbourhood_group    latitude          longitude
##  Brooklyn      :    0    Min.   :-0.4160   Min.   :-1.44113
##  Manhattan     :10955    1st Qu.: 0.6177   1st Qu.:-0.55298
##  Queens        :  701    Median : 1.1118   Median :-0.03818
##  Staten Island:    0    Mean   : 1.1188   Mean   :-0.10827
##  Bronx         :  588    3rd Qu.: 1.5750   3rd Qu.: 0.21187
##                          Max.   : 3.3632   Max.   : 2.52697
##            room_type         price
##  Private room   :5664   Min.   :-1.32482
##  Entire home/apt:6131   1st Qu.:-0.64323
##  Shared room    : 449   Median :-0.35924
##                         Mean   :-0.27143
##                         3rd Qu.: 0.08379
##                         Max.   : 2.01494
##
```

```
## $`2`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     :  56     Min.   :-2.9781   Min.   :0.1866
##  Manhattan    :   0     1st Qu.:-0.4289   1st Qu.:0.9451
##  Queens       :4611     Median : 0.2705   Median :1.6892
##  Staten Island:   0     Mean   : 0.1644   Mean   :1.9488
##  Bronx        : 480     3rd Qu.: 0.6139   3rd Qu.:2.7813
##                         Max.   : 3.2131   Max.   :5.1626
##            room_type        price
##  Private room   :3691   Min.   :-1.3248
##  Entire home/apt:1246   1st Qu.:-0.9386
##  Shared room    : 210   Median :-0.7568
##                         Mean   :-0.6038
##                         3rd Qu.:-0.4728
##                         Max.   : 4.0143
##
## $`3`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     :8786     Min.   :-4.0600   Min.   :-6.33242
##  Manhattan    :  34     1st Qu.:-1.0655   1st Qu.:-0.40671
##  Queens       :  86     Median :-0.7977   Median :-0.07171
##  Staten Island: 156     Mean   :-0.8380   Mean   :-0.12296
##  Bronx        :   0     3rd Qu.:-0.4476   3rd Qu.: 0.23544
##                         Max.   : 0.2964   Max.   : 3.30112
##            room_type        price
##  Private room   :  93   Min.   :-1.24530
##  Entire home/apt:8939   1st Qu.:-0.35924
##  Shared room    :  30   Median : 0.03835
##                         Mean   : 0.14476
##                         3rd Qu.: 0.51546
##                         Max.   : 3.61666
##
## $`4`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     : 733     Min.   :-4.0186   Min.   :-6.2338
##  Manhattan    :8099     1st Qu.:-0.1210   1st Qu.:-1.0029
##  Queens       : 163     Median : 0.1694   Median :-0.7745
##  Staten Island:  20     Mean   : 0.2083   Mean   :-0.7093
##  Bronx        :  14     3rd Qu.: 0.5639   3rd Qu.:-0.5387
##                         Max.   : 2.8482   Max.   : 3.3857
##            room_type        price
##  Private room   : 806   Min.   :-0.1661
##  Entire home/apt:8186   1st Qu.: 0.7654
##  Shared room    :  37   Median : 1.3334
##                         Mean   : 1.5055
##                         3rd Qu.: 1.9127
##                         Max.   : 4.1847
##
## $`5`
##     neighbourhood_group    latitude         longitude
##  Brooklyn     :10283    Min.   :-4.1808   Min.   :-6.29846
##  Manhattan    : 1789    1st Qu.:-0.9712   1st Qu.:-0.53179
##  Queens       :   71    Median :-0.6490   Median :-0.01785
##  Staten Island:  190    Mean   :-0.7161   Mean   :-0.09622
```

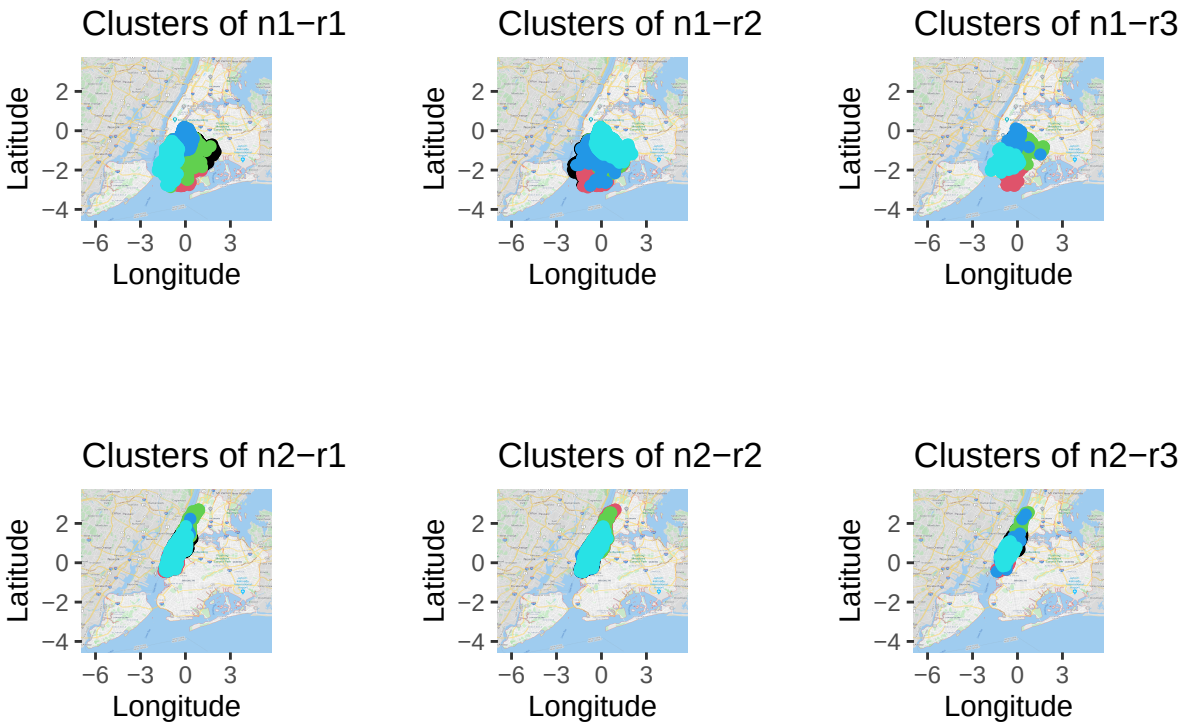
```
##  Bronx         :    0    3rd Qu.:-0.3060   3rd Qu.: 0.36652
##                          Max.   : 0.4209   Max.   : 1.69290
##            room_type         price
##  Private room   :11913   Min.   :-1.3248
##  Entire home/apt:    1   1st Qu.:-0.9272
##  Shared room    :  419   Median :-0.7568
##                          Mean   :-0.6871
##                          3rd Qu.:-0.5183
##                          Max.   : 0.7767
```

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

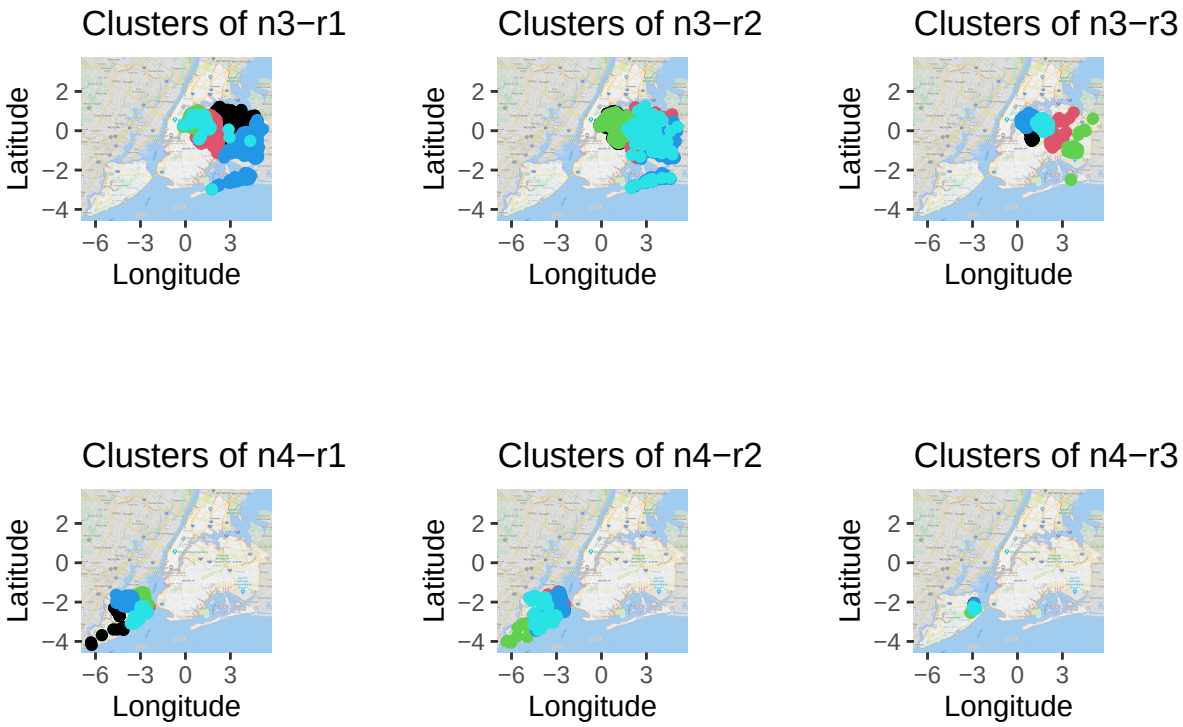
```
## Loading required package: grid
```



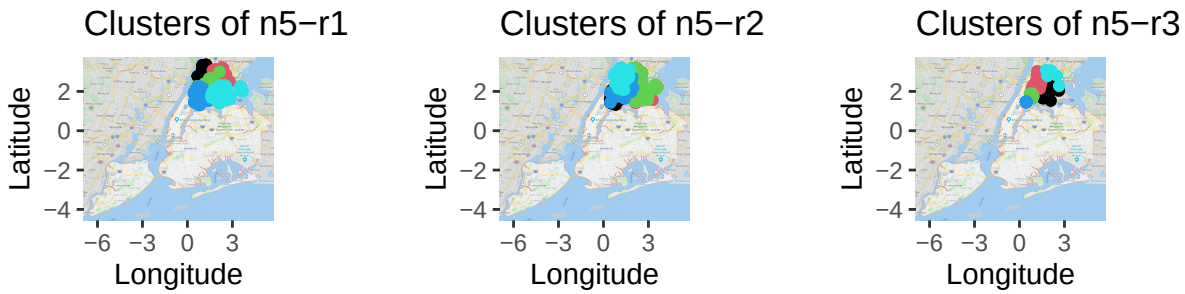

```
multiplots(all_cluster$all_plot[6:11], cols=3)
```




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




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




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

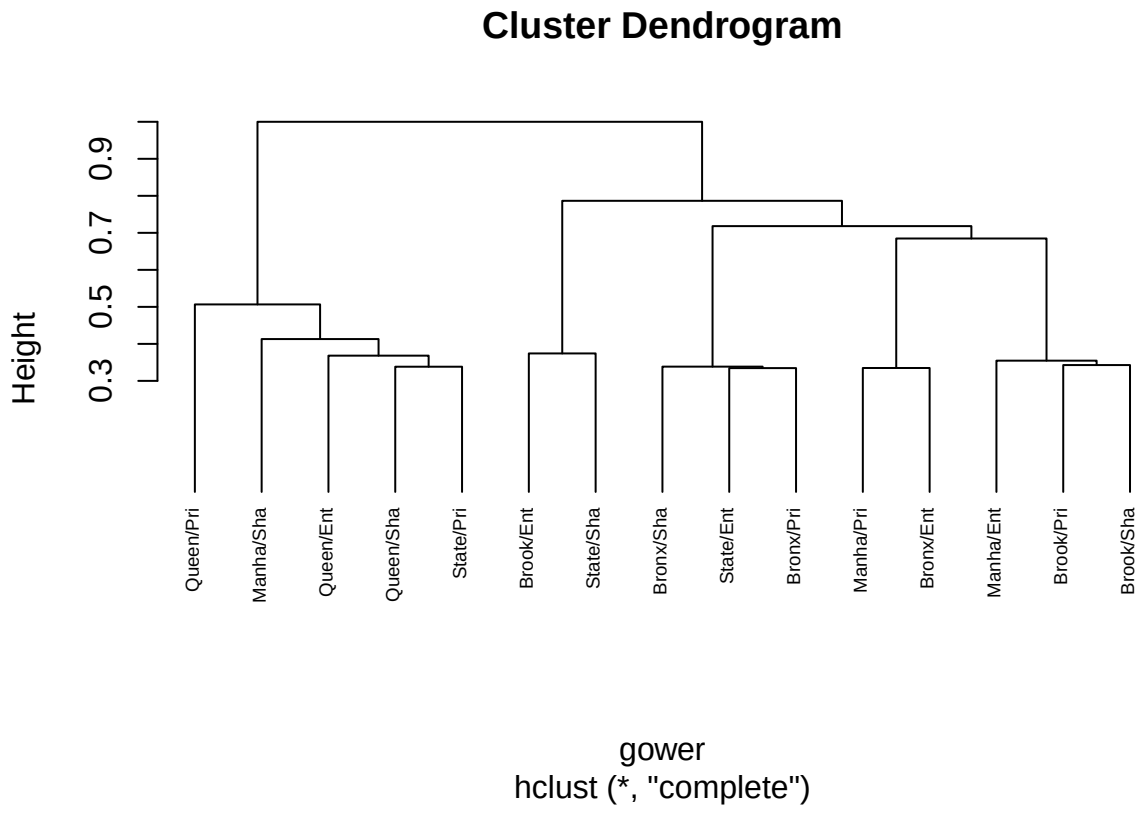
Hierarchical Cluster Analysis

```
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
```

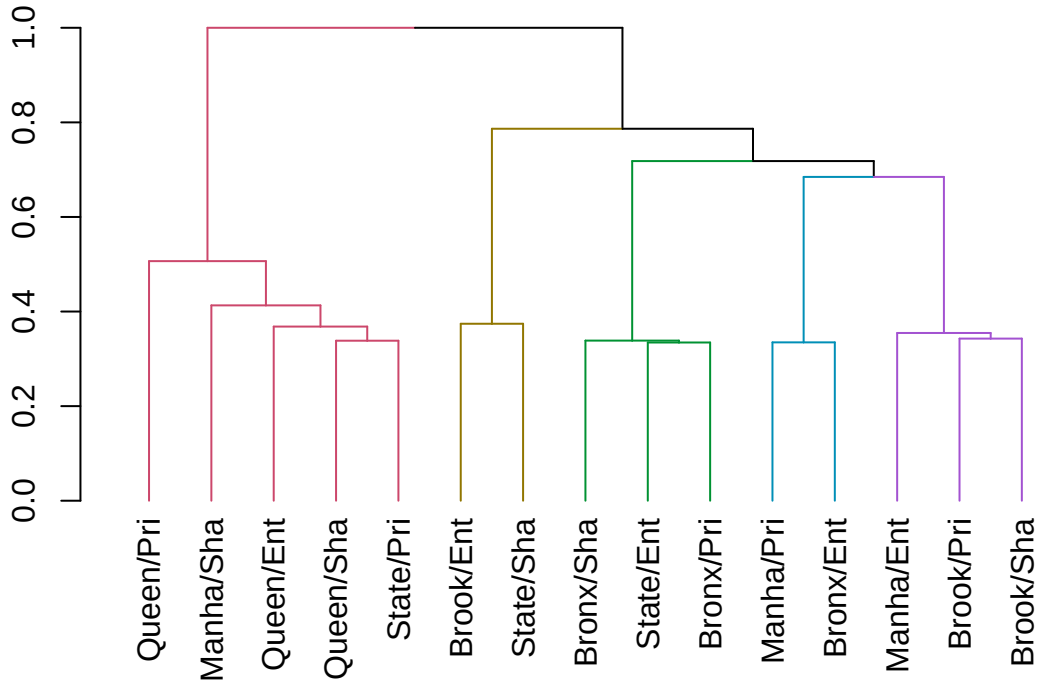
```
##           neighbourhood_group       room_type       price
## Brook/Pri            Brooklyn    Private room -0.68331638
## Brook/Ent           Manhattan    Private room -0.31844499
## Brook/Sha              Queens    Private room -0.73368292
## Manha/Pri       Staten Island    Private room -0.78758723
```

```
## 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
```

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

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



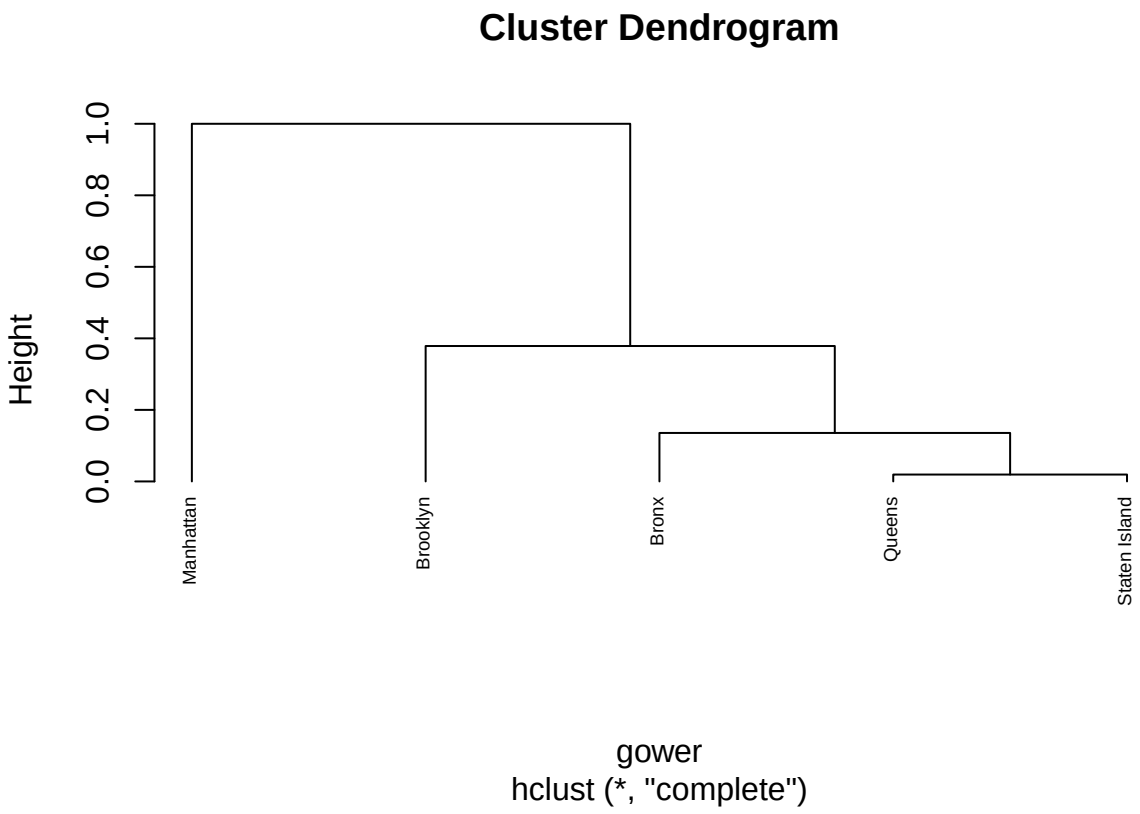

```
avg_dend_obj <- as.dendrogram(hc1)
avg_col_dend <- color_branches(avg_dend_obj, h = 0.6)
plot(avg_col_dend)
```




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

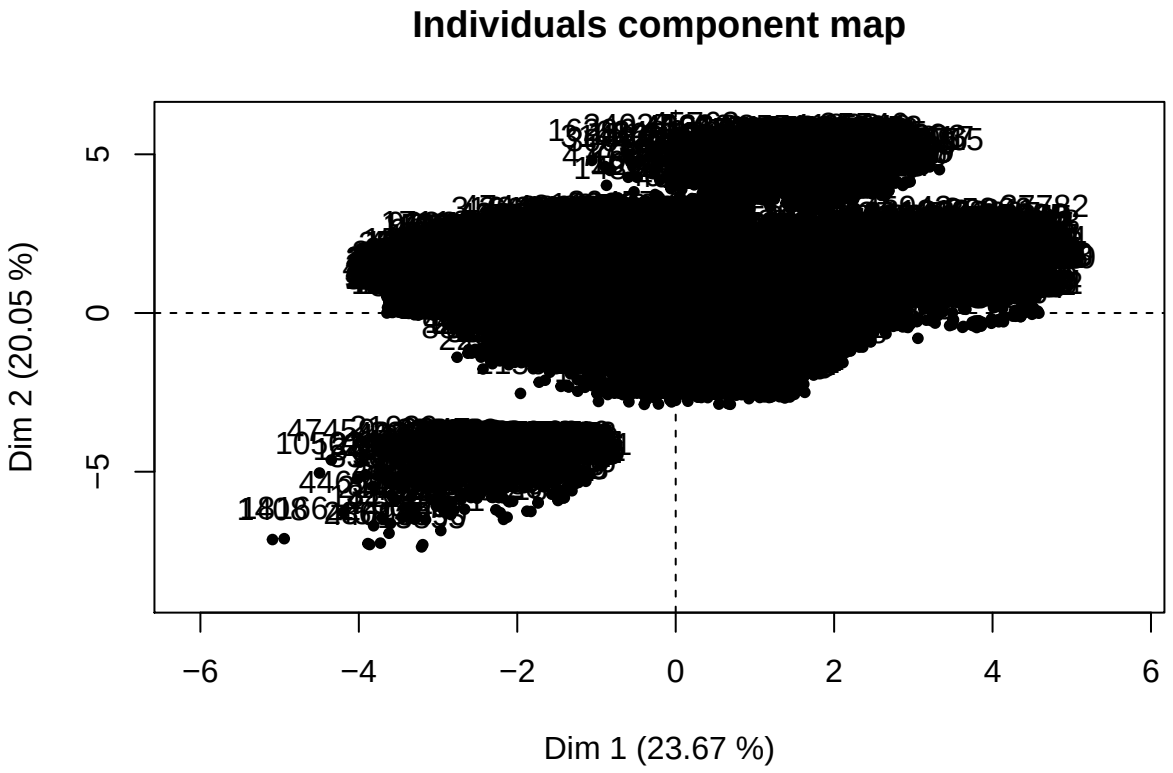
```
##   neighbourhood_group      price
## 1            Brooklyn -0.2147398
## 2           Manhattan  0.3601591
## 3              Queens -0.4396243
## 4       Staten Island -0.4574125
## 5               Bronx -0.5650283
```

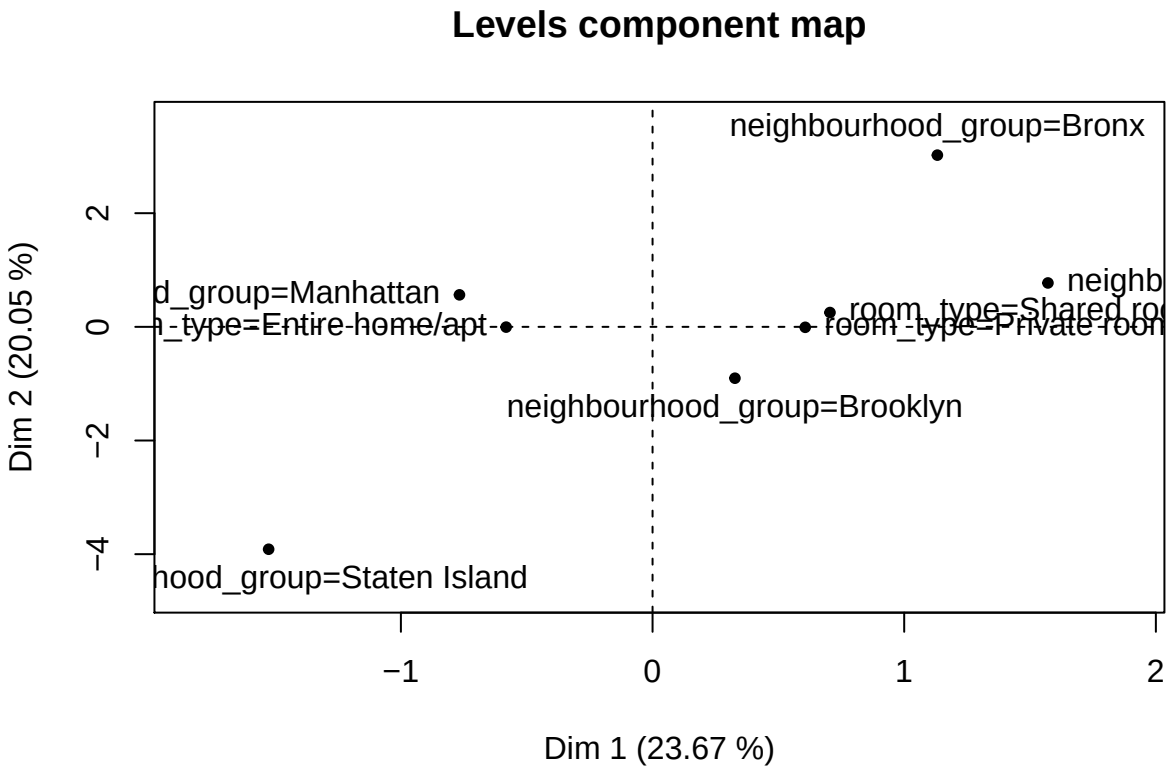
```
rownames(agg) = c("Brooklyn","Manhattan",
                  "Queens","Staten Island", "Bronx")
agg$neighbourhood_group = NULL
gower <- daisy(agg, metric = "gower")
hc1 <- hclust(gower, method = "complete" )

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

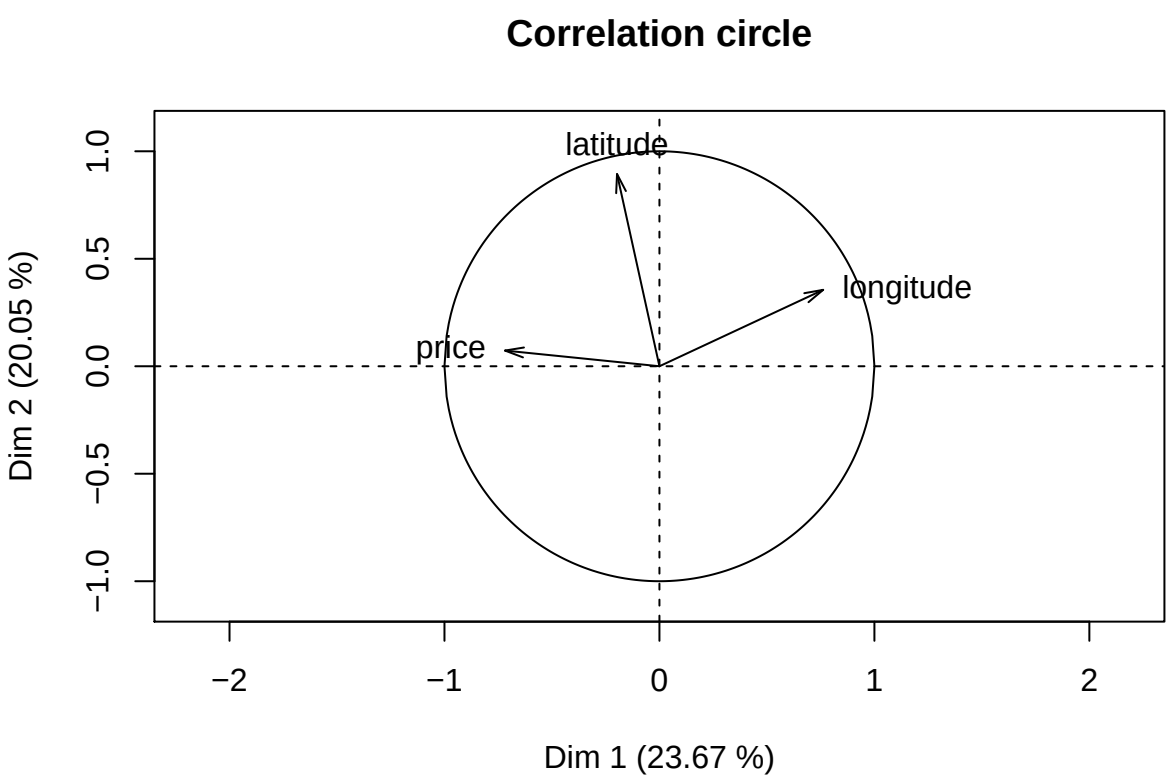


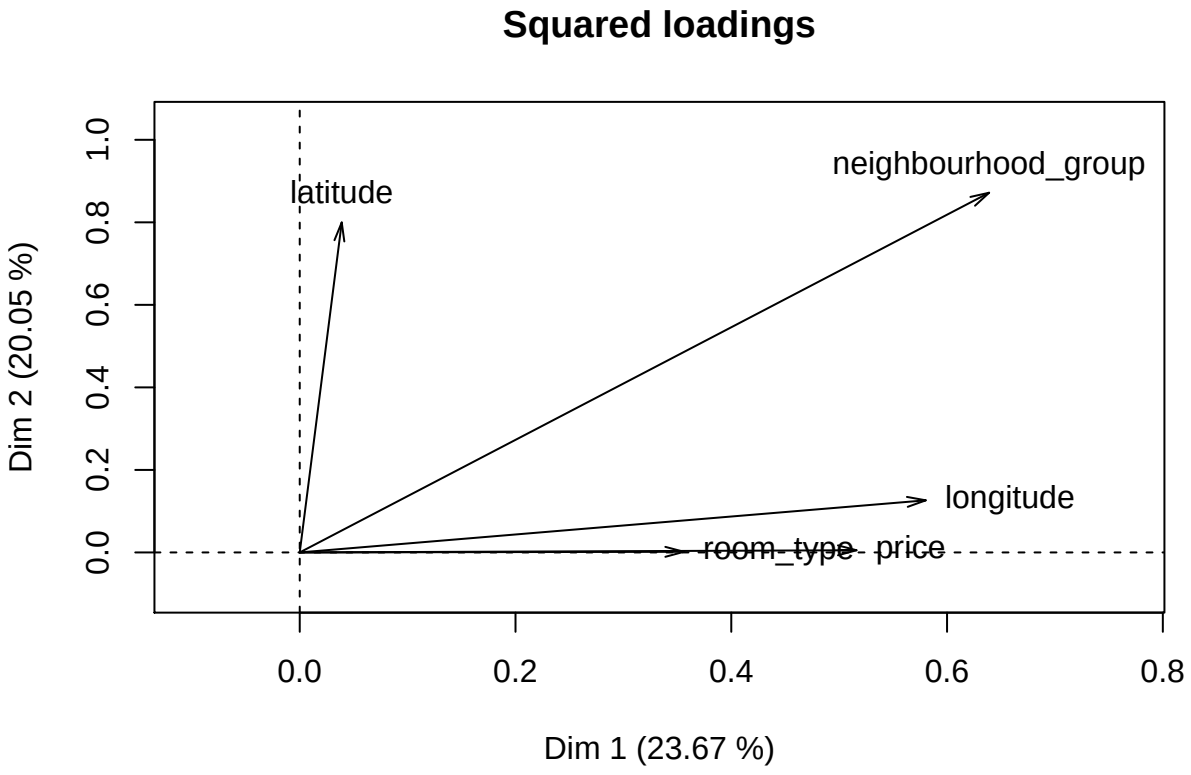
PCAmixdata

```
## Split mixed dataset into quantitative and qualitative variables
## For now excluding the target variable "Churn", which will be added later as a supplementary variable
#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 %)
5
0
−5
−6
−4
−2
0
2
4
6
Dim 1 (23.67 %)


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


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




```
res.pcamix
```

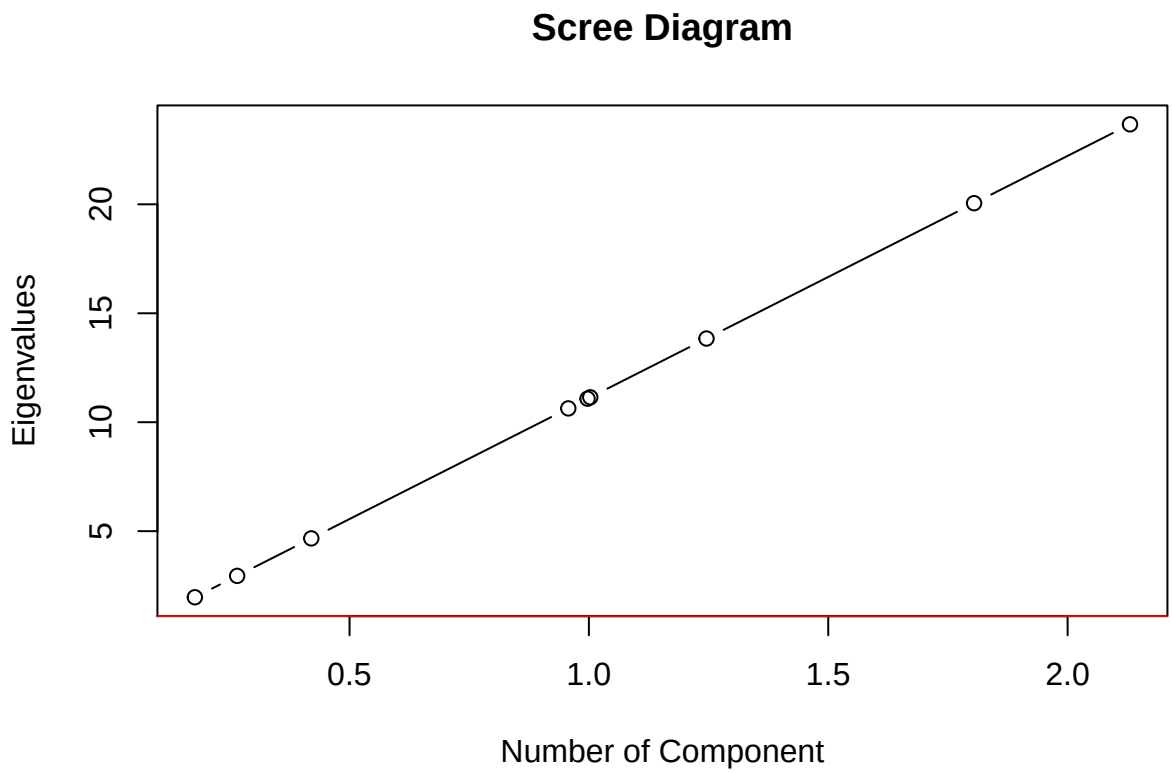
```
##
## Call:
## PCAmix(X.quanti = split$X.quanti, X.quali = split$X.quali, rename.level = TRUE)
##
## Method = Principal Component of mixed data (PCAmix)
##
##
## "name" "description"
## "$eig" "eigenvalues of the principal components (PC) "
## "$ind" "results for the individuals (coord,contrib,cos2)"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

```
# Use Scree Diagram to select the components:

plot(res.pcamix$eig, type="b", main="Scree Diagram", xlab="Number of Component", ylab="Eigenvalues")
abline(h=1, lwd=3, col="red")
```



- Factor Analysis of Mixed Data (FAMD)

http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/115-famd-factor-analysis-of-mixed-data-in-r-essentials/ https://nextjournal.com/pc-methods/calculate-pc-mixed-data https://cran.r-project.org/web/packages/FactoMineR/index.html https://stats.stackexchange.com/questions/5774/can-principal-component-analysis-be-applied-to-datasets-containing-a-mix-of-cont

```
library("FactoMineR")
library("factoextra")
```

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.

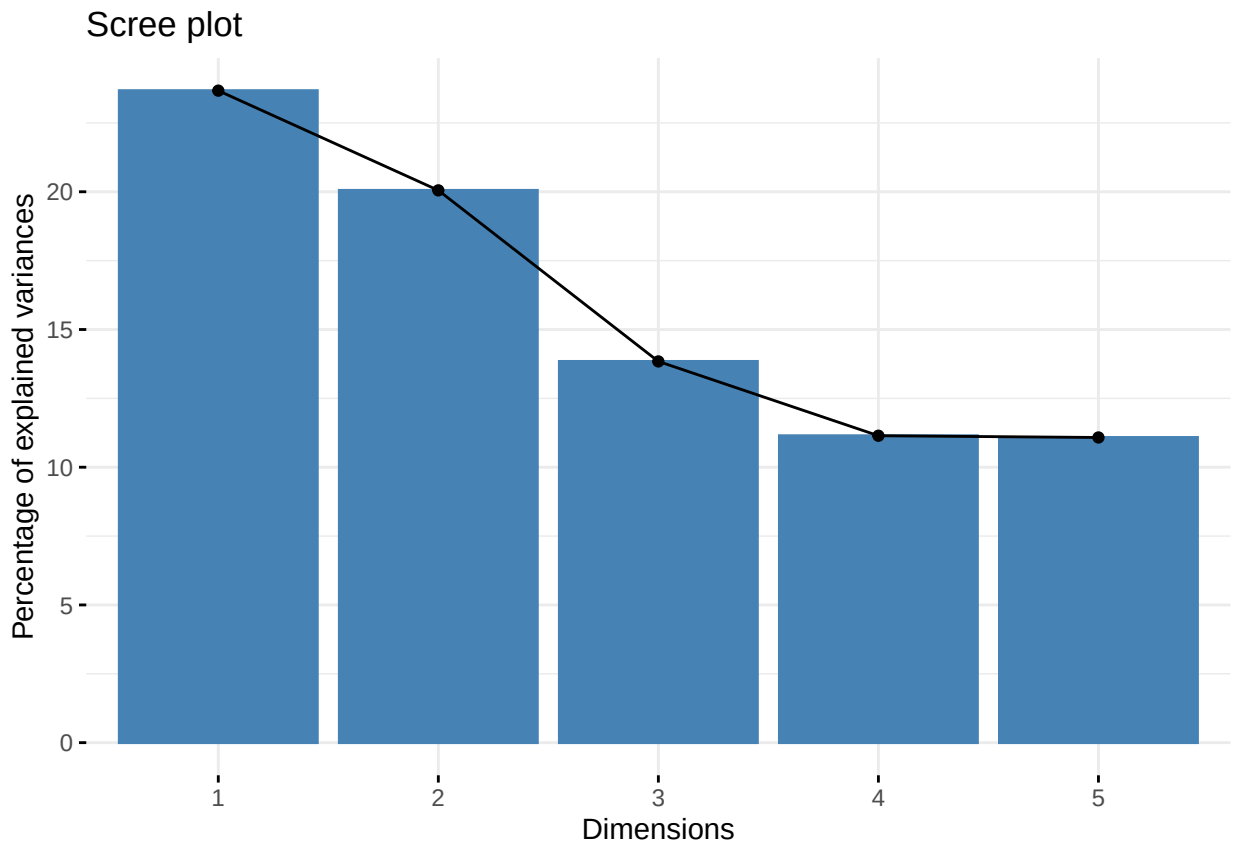
```
res.famd <- FAMD(clust_data$all, graph = F, ncp = 5)
print(res.famd)
```

```
## *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)
```

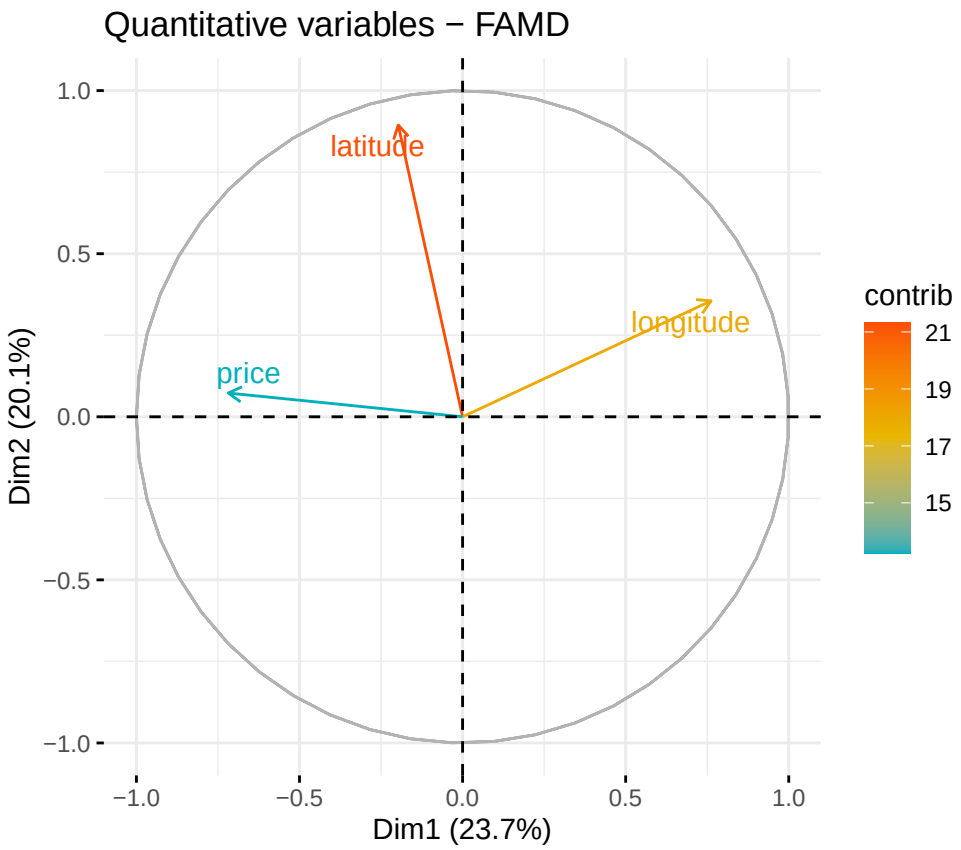



```
quanti.var <- get_famd_var(res.famd, "quanti.var")
quanti.var
```

```
## FAMD results for quantitative variables
##  ===================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
```

```
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

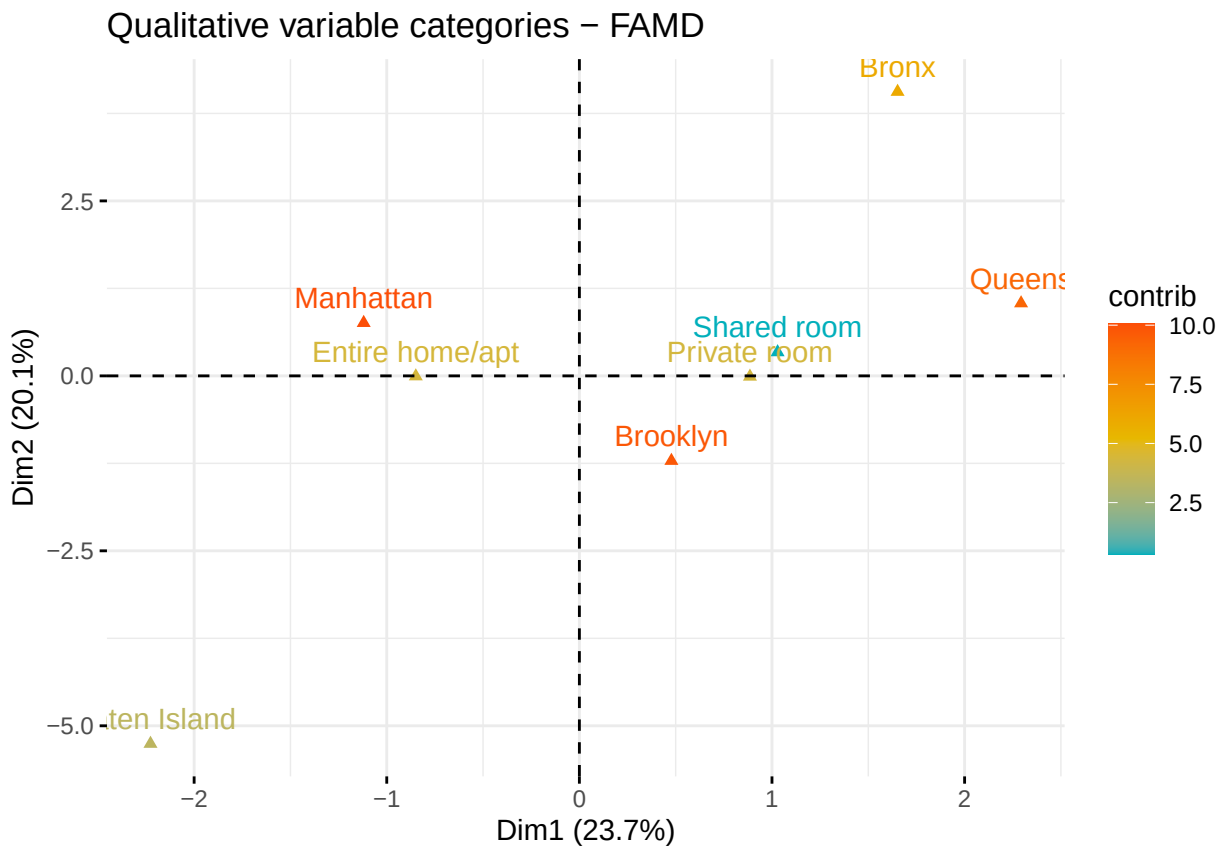
```
fviz_famd_var(res.famd, "quanti.var", col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
              repel = TRUE)
```




```
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)
```

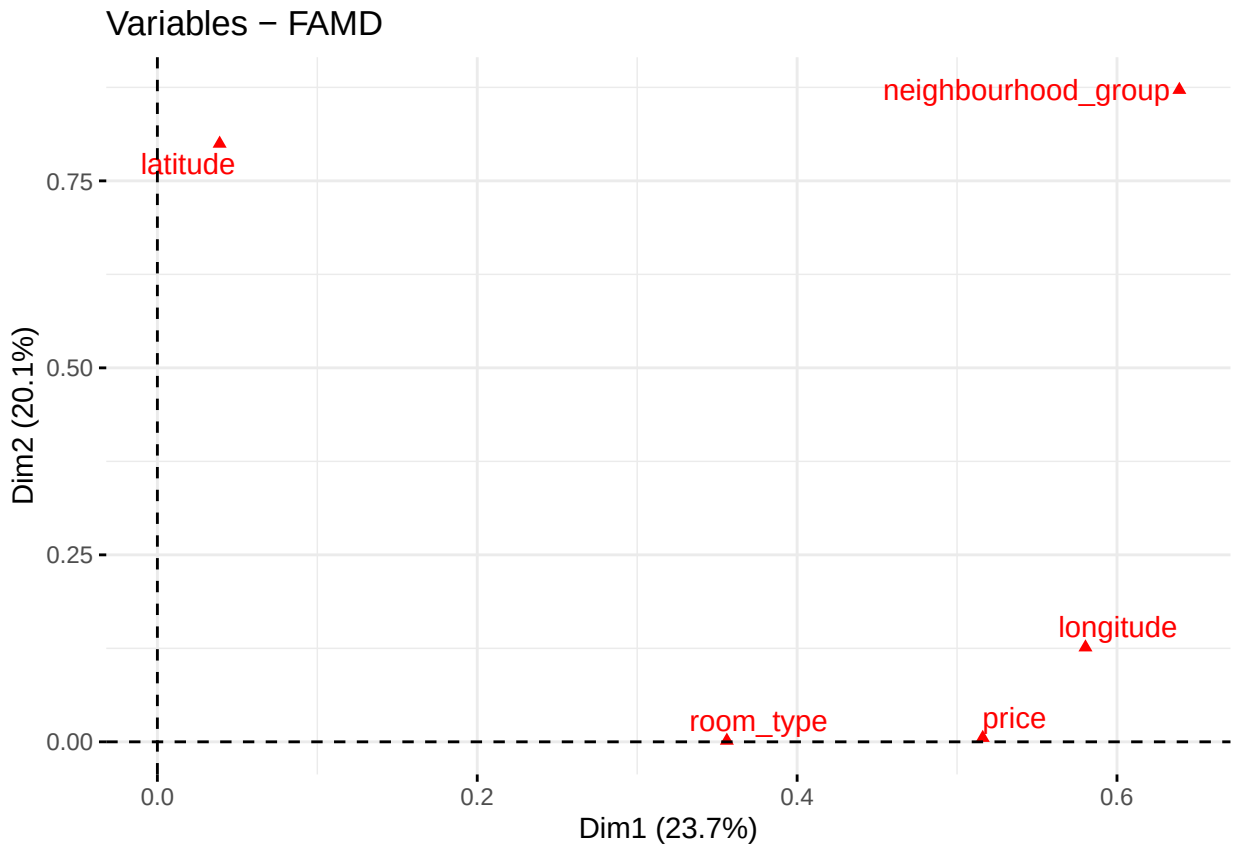
```
##                          Dim.1        Dim.2       Dim.3        Dim.4
## latitude            0.00151495 6.395904e-01 0.002404821 2.826116e-06
## longitude           0.33675663 1.595010e-02 0.027281805 1.448739e-08
## price               0.26633043 2.822981e-05 0.055567086 2.579807e-07
## neighbourhood_group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

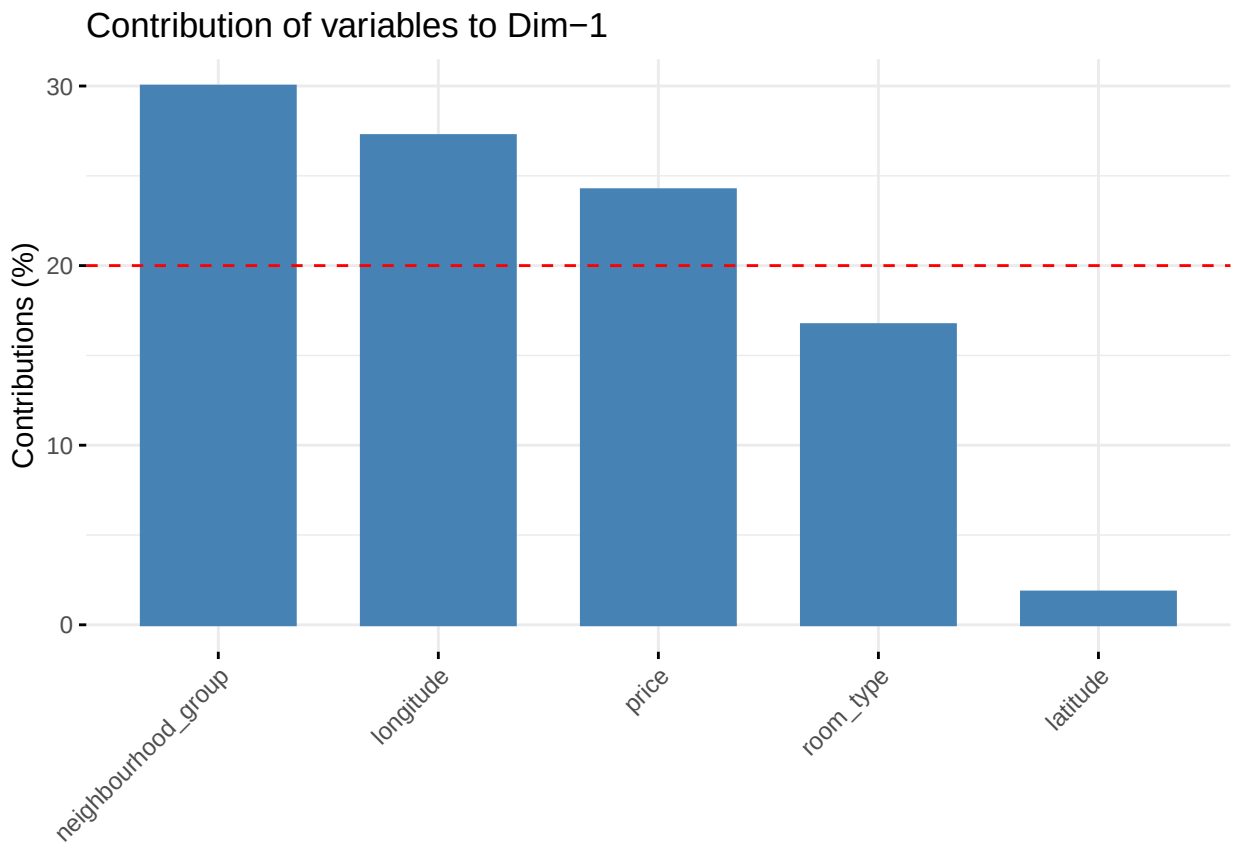
```
# Contributions to the  dimensions
head(var$contrib)
```

```
##                          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
```

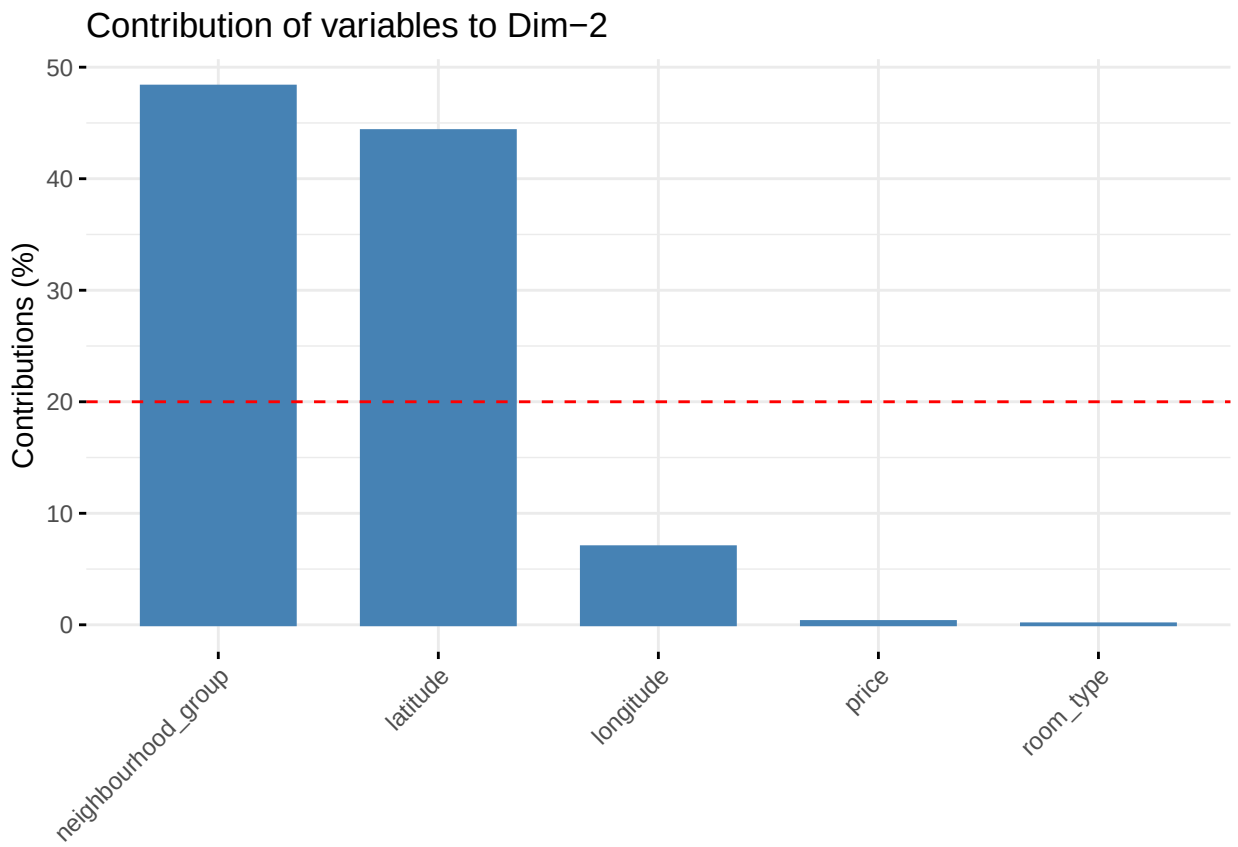
```
# Plot of variables

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



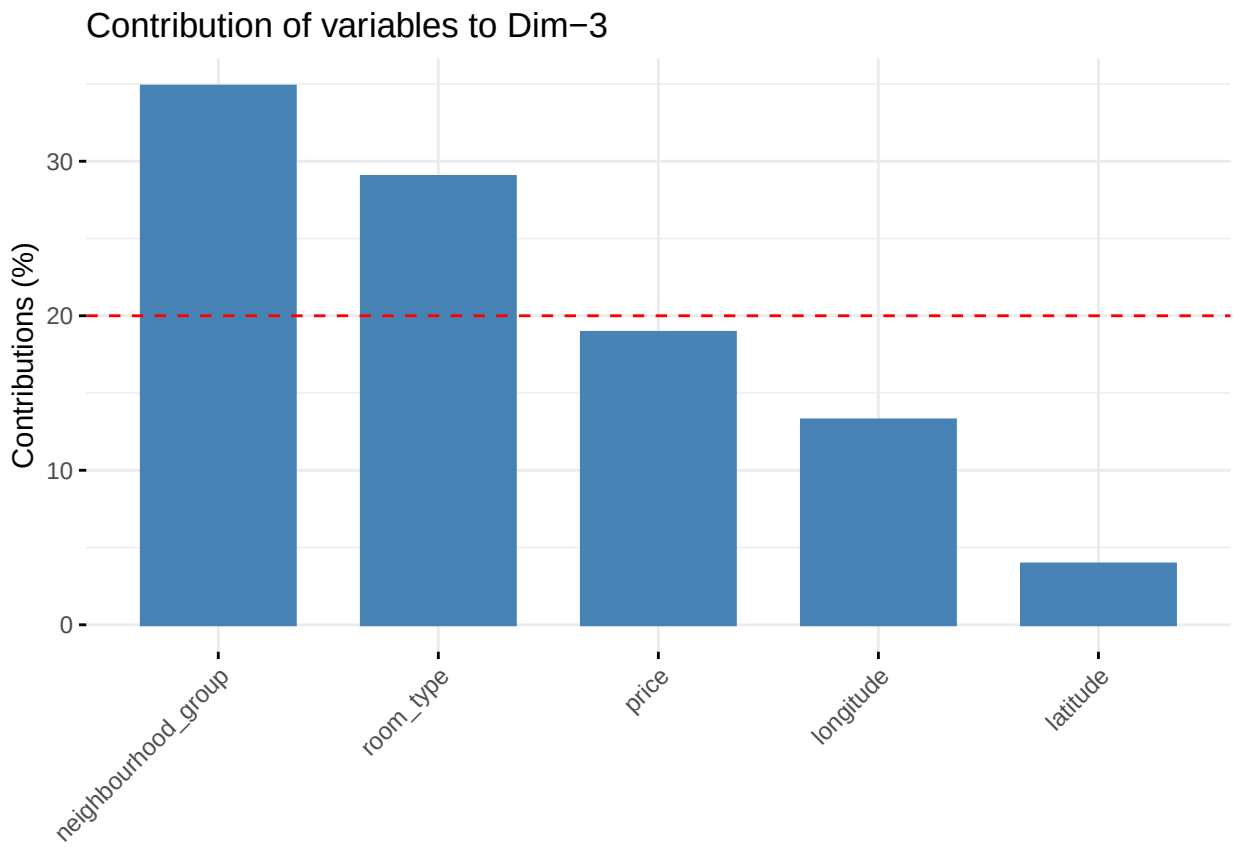

```
# Contribution to the first dimension
fviz_contrib(res.famd, "var", axes = 1)
```



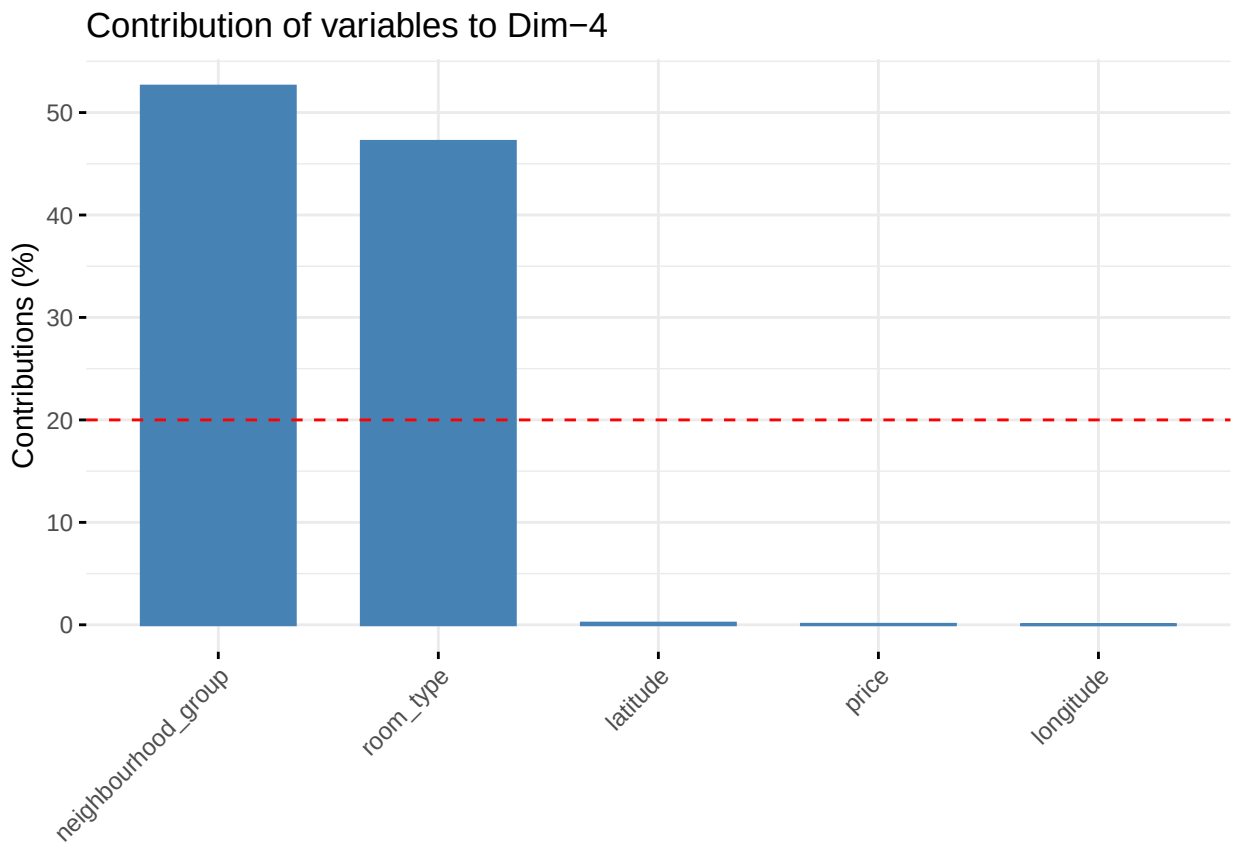

```
# Contribution to the second dimension
fviz_contrib(res.famd, "var", axes = 2)
```



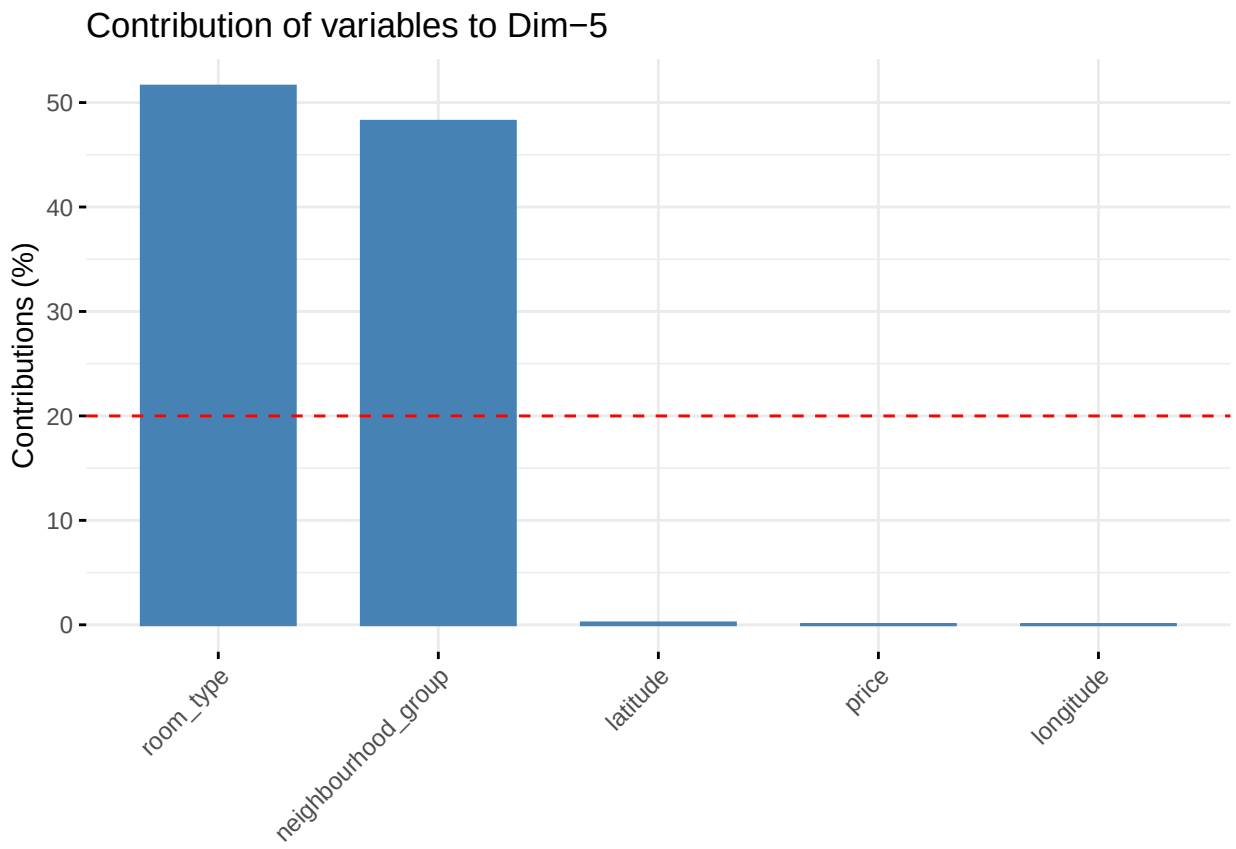

```
# Contribution to the third dimension
fviz_contrib(res.famd, "var", axes = 3)
```




```
# 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