Supplementary Notebook

August E. G. Mikkelsen

2025-05-08

Introduction

This notebook provides a walkthrough of the analysis and results described in the paper "Generalized Additive Models for Real Estate Valuation: A Case Study of the Danish Housing Market".

Dataset

1. Description

The analysis is based on a dataset of single family home sales in Denmark in the period 01/01/2014-01/01/2024, which is included as .csv-file in the same repository containing this notebook. The dataset is loaded and inspected below and in the subsequent cells histograms of selected variables are produced.

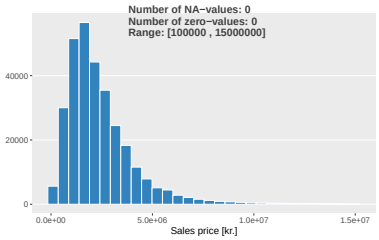
```
dataset <- read.csv2("dataset.csv")
str(dataset)
```

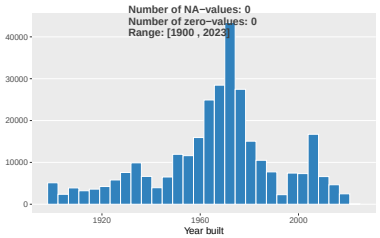
```
## 'data.frame':    307510 obs. of  32 variables:
##  $ id                    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ sales_date            : chr  "2017-05-25" "2015-07-07" "2018-04-15" "2016-06-29" ...
##  $ municipality          : chr  "København" "København" "København" "København" ...
##  $ province              : chr  "ByenKøbenhavn" "ByenKøbenhavn" "ByenKøbenhavn" "ByenKøbenhavn" ...
##  $ region                : chr  "Hovedstaden" "Hovedstaden" "Hovedstaden" "Hovedstaden" ...
##  $ municipality_type     : chr  "Capital" "Capital" "Capital" "Capital" ...
##  $ zone                  : chr  "Urban" "Urban" "Urban" "Urban" ...
##  $ sales_price           : num  4775000 3400000 3200000 2595000 5350000 ...
##  $ house_type            : chr  "Detached" "Detached" "Semi-detached" "Detached" ...
##  $ living_space          : int  120 116 124 82 148 132 124 150 124 124 ...
##  $ number_of_rooms       : int  4 5 5 3 4 5 5 6 5 5 ...
##  $ number_of_floors      : int  1 1 2 1 1 2 2 2 2 2 ...
##  $ number_of_bathrooms   : int  2 2 1 1 2 2 2 1 1 1 ...
##  $ lot_size              : num  363 636 137 412 374 137 184 336 137 137 ...
##  $ year_built            : int  1972 1954 1939 1937 2017 1939 1939 1939 1939 1939 ...
##  $ year_rebuilt          : int  NA 1965 NA NA NA NA NA NA NA NA ...
##  $ roof_type             : chr  "Fiber cement" "Fiber cement" "Roofing felt" "Roofing felt" ...
##  $ heating_type          : chr  "District heating" "District heating" "District heating" "District he
##  $ wall_type             : chr  "Brick" "Brick" "Brick" "Brick" ...
##  $ coordinate_east       : num  726259 725936 719324 717526 720035 ...
##  $ coordinate_north      : num  6171969 6172303 6173274 6179890 6173056 ...
##  $ coast_distance        : int  NA NA NA NA NA NA NA NA NA NA ...
```

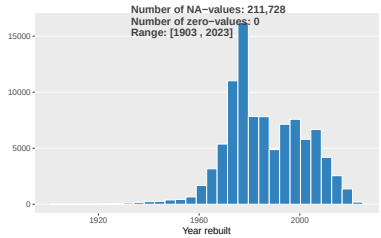
```
##  $ lake_distance         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ stream_distance       : int  NA NA NA 979 NA NA NA NA NA NA ...
##  $ road_distance         : int  181 164 27 97 165 128 36 69 156 36 ...
##  $ highway_distance      : int  1279 1676 327 563 692 452 501 427 487 338 ...
##  $ train_station_distance: int  776 329 915 604 712 799 884 858 769 905 ...
##  $ railway_distance      : int  695 322 173 387 535 294 351 271 330 183 ...
##  $ windmill_distance     : int  NA NA 2623 NA 2064 2688 2791 2699 2707 2629 ...
##  $ ocean_view            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ lake_view             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ forest_size           : num  0 0 0.73 6.88 0 0.66 0.73 0.73 0.59 0.73 ...
```

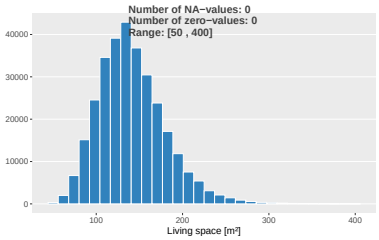
```r
get_label <- function(var, labels_vec) {
  lbl <- tryCatch(labels_vec[[var]], error = function(e) NULL)
  if (is.null(lbl) || is.na(lbl)) var else lbl
}

plot_one <- function(df, var, label) {

  x <- df[[var]]

  if (label == "Sales date") {
    x_date <- as.Date(x)
    x <- decimal_date(x_date)
  }

  na_count <- sum(is.na(x))
  zero_count <- sum(x == 0, na.rm = TRUE)
  na_label <- paste0("Number of NA-values: ", format(na_count, big.mark = ",", scientific = FALSE))
  zero_label <- paste0("Number of zero-values: ", format(zero_count, big.mark = ",", scientific = FALSE)

  if (is.numeric(x)) {

    df2 <- tibble(x = x) %>% filter(is.finite(x) & x != 0)

    rng <- range(df2$x, na.rm = TRUE)

    rng_label <- paste0("Range: [", format(rng[1], big.mark = "", scientific = FALSE, digits = 2, trim =
                                    format(rng[2], big.mark = "", scientific = FALSE, digits = 2, trim =

    x_pos = (0.75*min(x, na.rm = TRUE) + 0.25*max(x, na.rm = TRUE))

    ggplot(df2, aes(x = x)) +
      geom_histogram(bins = 30, fill = "#3182bd", color = "white") +
      labs(x = label, y = NULL) +
      annotate("text", x = x_pos, y = Inf, label = na_label, hjust = 0, vjust = 0, size = 4.5, color =
      annotate("text", x = x_pos, y = Inf, label = zero_label, hjust = 0, vjust = 1.5, size = 4.5, colo
      annotate("text", x = x_pos, y = Inf, label = rng_label, hjust = 0, vjust = 3, size = 4.5, color =
      coord_cartesian(clip = "off") +
      theme(text = element_text(size = 12),
            plot.margin = margin(t = 28, r = 12, b = 5, l = 5),
            panel.grid.minor = element_blank(),
            panel.grid.major.x = element_blank())

  } else {
```

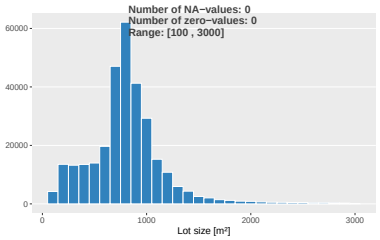
```
    x2 <- fct_lump_n(factor(x), n = 30, other_level = "Other")

    df_counts <- tibble(x = x2) %>%
                 filter(!is.na(x)) %>%
                 count(x, name = "n")

    x_pos = (nlevels(df_counts$x) + 1) / 2

    ggplot(df_counts, aes(x = x, y = n)) +
      geom_col(aes(fill = x), color = "white", show.legend = FALSE) +
      scale_fill_brewer(palette = "Set3") +
      geom_text(aes(label = scales::comma(n)), vjust = -0.2, size = 3.3, color = "grey15") +
      labs(x = NULL, y = NULL) +
      ggtitle(label) +
      scale_y_continuous(expand = expansion(mult = c(0.02, 0.12))) +
      coord_cartesian(clip = "off") +
      theme(plot.margin = margin(t = 20, r = 12, b = 5, l = 5),
            axis.text.x = element_text(angle = 45, hjust = 1),
            panel.grid.minor = element_blank(),
            panel.grid.major.x = element_blank())}
}


variables_continuous <- c(
  sales_price = "Sales price [kr.]",
  year_built = "Year built",
  year_rebuilt = "Year rebuilt",
  living_space = "Living space [m²]",
  lot_size = "Lot size [m²]",
  coast_distance = "Distance to coast [m]",
  lake_distance = "Distance to lake [m]",
  stream_distance = "Distance to stream [m]",
  road_distance = "Distance to road [m]",
  highway_distance = "Distance to highway [m]",
  train_station_distance = "Distance to train station [m]",
  railway_distance = "Distance to railway [m]",
  windmill_distance = "Distance to windmill [m]",
  ocean_view = "Ocean view [m]",
  lake_view = "Lake view [m]",
  forest_size = "Forest size [ha]",
  number_of_bathrooms = "Number of bathrooms",
  number_of_rooms = "Number of rooms",
  number_of_floors = "Number of floors",
  sales_date = "Sales date")

vars_cont <- intersect(names(variables_continuous), names(dataset))

histograms_cont <- map(vars_cont, ~ plot_one(dataset, .x, get_label(.x, variables_continuous)))

saveRDS(histograms_cont, "saved_files/histograms_continuous.rds")
wrap_plots(histograms_cont, ncol = 3)
```

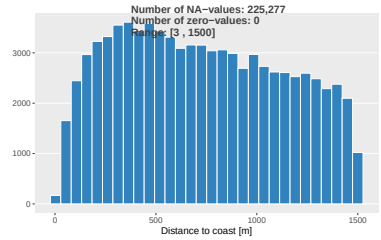

Number of NA−values: 0
Number of zero−values: 0
Range: [100000 , 15000000]
Sales price [kr.]

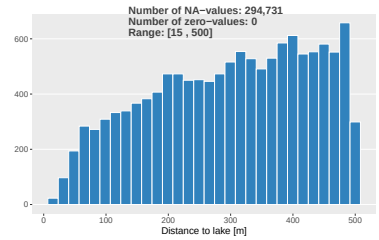

Number of NA−values: 0
Number of zero−values: 0
Range: [1900 , 2023]
Year built

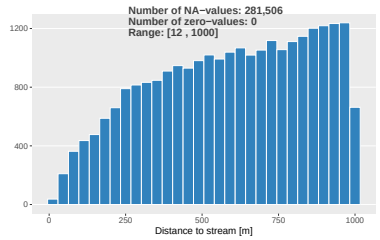

Number of NA−values: 211,728
Number of zero−values: 0
Range: [1903 , 2023]
Year rebuilt

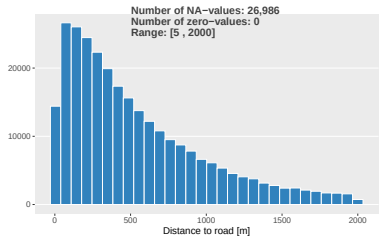

Number of NA−values: 0
Number of zero−values: 0
Range: [50 , 400]
Living space [m²]

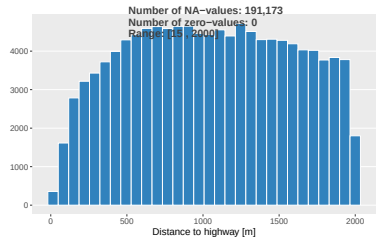

Number of NA−values: 0
Number of zero−values: 0
Range: [100 , 3000]
Lot size [m²]

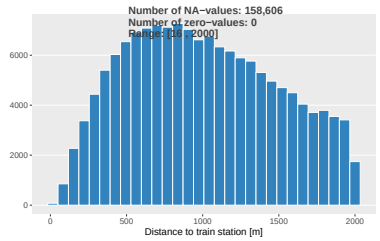

Number of NA−values: 225,277
Number of zero−values: 0
Range: [3 , 1500]
Distance to coast [m]

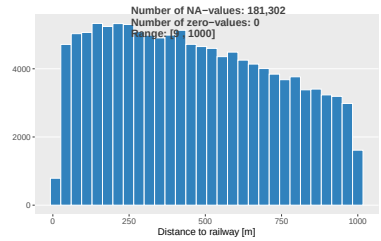

Number of NA−values: 294,731
Number of zero−values: 0
Range: [15 , 500]
Distance to lake [m]

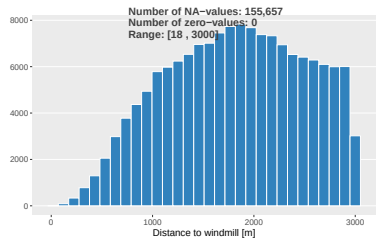

Number of NA−values: 281,506
Number of zero−values: 0
Range: [12 , 1000]
Distance to stream [m]

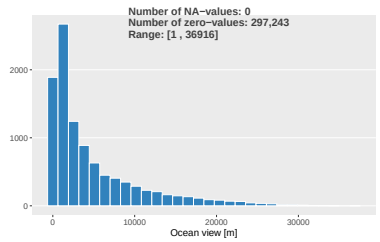

Number of NA−values: 26,986
Number of zero−values: 0
Range: [5 , 2000]
Distance to road [m]

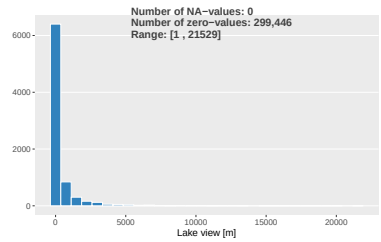

Number of NA−values: 191,173
Number of zero−values: 0
Range: [15 , 2000]
Distance to highway [m]

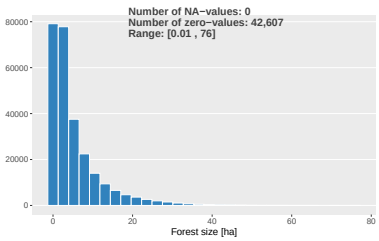

Number of NA−values: 158,606
Number of zero−values: 0
Range: [18 , 2000]
Distance to train station [m]

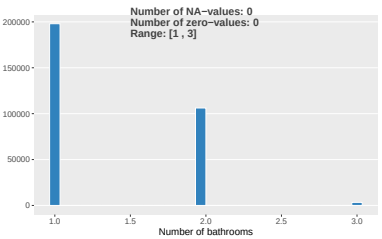

Number of NA−values: 181,302
Number of zero−values: 0
Range: [9 , 1000]
Distance to railway [m]

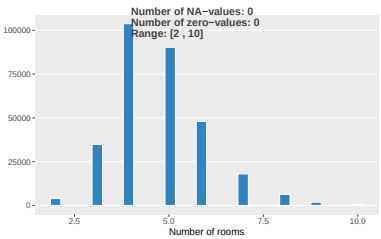

Number of NA−values: 155,657
Number of zero−values: 0
Range: [18 , 3000]
Distance to windmill [m]

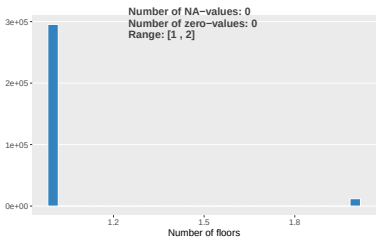

Number of NA−values: 0
Number of zero−values: 297,243
Range: [1 , 36916]
Ocean view [m]

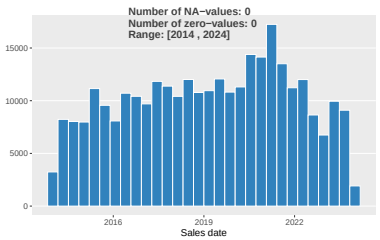

Number of NA−values: 0
Number of zero−values: 299,446
Range: [1 , 21529]
Lake view [m]


Number of NA−values: 0
Number of zero−values: 42,607
Range: [0.01 , 76]
Forest size [ha]

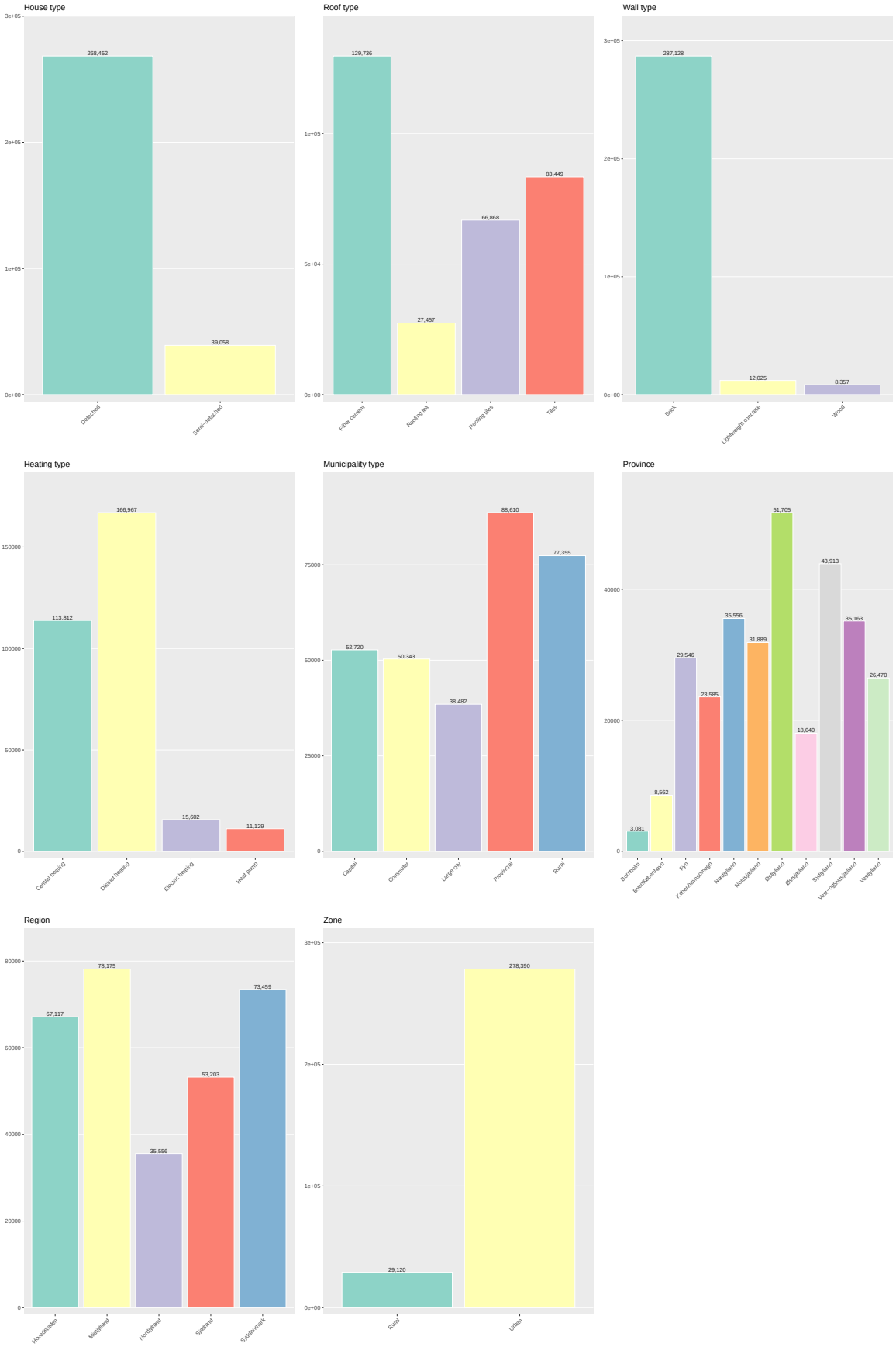

Number of NA−values: 0
Number of zero−values: 0
Range: [1 , 3]
Number of bathrooms


Number of NA−values: 0
Number of zero−values: 0
Range: [2 , 10]
Number of rooms


Number of NA−values: 0
Number of zero−values: 0
Range: [1 , 2]
Number of floors


Number of NA−values: 0
Number of zero−values: 0
Range: [2014 , 2024]
Sales date

```r
variables_categorical <- c(
  house_type = "House type",
  roof_type = "Roof type",
  wall_type = "Wall type",
  heating_type = "Heating type",
  municipality_type = "Municipality type",
  province = "Province",
  region = "Region",
  zone = "Zone"
)

vars_cat <- intersect(names(variables_categorical), names(dataset))

histograms_cat <- map(vars_cat, ~ plot_one(dataset, .x, get_label(.x, variables_categorical)))

saveRDS(histograms_cat, "saved_files/histograms_categorical.rds")

wrap_plots(histograms_cat, ncol = 3)
```


House type
268,452
39,058
Detached
Semi-detached
Roof type
129,736
27,457
66,868
83,449
Fibre cement
Roofing felt
Roofing tiles
Tiles
Wall type
287,128
12,025
8,357
Brick
Lightweight concrete
Wood
Heating type
113,812
166,967
15,602
11,129
Central heating
District heating
Electric heating
Heat pump
Municipality type
52,720
50,343
38,482
88,610
77,355
Capital
Commuter
Large city
Provincial
Rural
Province
3,081
8,562
29,546
23,585
35,556
31,889
51,705
18,040
43,913
35,163
26,470
Bornholm
Byen København
Fyn
Københavns omegn
Nordjylland
Nordsjælland
Østjylland
Østsjælland
Sydjylland
Vest- Sydsjælland
Vestjylland
Region
67,117
78,175
35,556
53,203
73,459
Hovedstaden
Midtjylland
Nordjylland
Sjælland
Syddanmark
Zone
29,120
278,390
Rural
Urban

2. Variable conversions and imputation

The analysis relies on a set of variable conversions performed in order to:

- Handle missing values in the dataset
- Convert categorical variables to factors
- Convert sales dates to numeric values

The two first are straightforward while the treatment of missing values requires more consideration. The analysis presented in XXX is based on the following simply imputation scheme:

- year_rebuilt: For this variable all NA's as well as values less than 1970 are replaced by 1970.
- distance-variables: For these variables all NA's are replaced by the maximum value of the respective variable.

The code for the above transformations is provided below.

```
#convert relevant categorical variables to factors with chosen references
dataset$municipality_type <- relevel(factor(dataset$municipality_type), ref = "Provincial")
dataset$house_type <- relevel(factor(dataset$house_type), ref = "Detached")
dataset$roof_type <- relevel(factor(dataset$roof_type), ref = "Fiber cement")
dataset$wall_type <- relevel(factor(dataset$wall_type), ref = "Brick")
dataset$heating_type <- relevel(factor(dataset$heating_type), ref = "District heating")

#create a numerical variable for sales_date
dataset <- dataset %>% mutate(sales_date = as.Date(sales_date),
                              sales_date_numeric = lubridate::decimal_date(sales_date))

#impute NA-values
dataset <- dataset %>%
  mutate(year_rebuilt = if_else(is.na(year_rebuilt) | year_rebuilt < 1970, 1970, year_rebuilt))

dataset <- dataset %>% mutate(across(c(lake_distance,
                                       stream_distance,
                                       coast_distance,
                                       road_distance,
                                       highway_distance,
                                       train_station_distance,
                                       railway_distance,
                                       windmill_distance),
                                     ~ replace_na(., max(., na.rm = TRUE))))

#check that there are no remaining NAs
which(is.na(dataset), arr.ind = TRUE)
```

3. Cross-Validation and Test Set

A simple 4-fold cross-validation split is implemented in the code below.

```
#shuffle dataset randomly
dataset <- dataset[sample(nrow(dataset)), ]
```

```r
#assign testing points
dataset <- dataset %>% mutate(test_set = runif(n()) < test_fraction)

#assign validation points
dataset <- dataset %>%
  mutate(
    validation_set = ifelse(
      !test_set,
      ceiling(
        rank(runif(sum(!test_set))) * N / sum(!test_set)
      ),
      NA
    )
  )

#check that it looks correct
table(dataset$test_set, useNA = "ifany")
table(dataset$validation_set, useNA = "ifany")
```

GAM Models

1. Model training

The code below fits the GAM model structure described in "Generalized Additive Models for Real Estate Valuation: A Case Study of the Danish Housing Market" based on the cross validation split employed.

```r
fit_fold <- function(i, k_spatial, fraction = 1.0) {

  training_set <- dataset %>% filter(validation_set != i, !test_set)

  training_subset <- training_set %>%
                     group_by(municipality_type) %>%
                     slice_head(prop = fraction) %>%
                     ungroup()

  model <- bam(sales_price ~
               s(coordinate_east, coordinate_north, k = k_spatial, pc = c(697000, 6150000)) +
               s(sales_date_numeric, k = k_other, by = municipality_type, pc = 2024) +
               s(year_built, k = k_other, by = municipality_type, pc = 1970) +
               s(year_rebuilt, k = k_other, by = municipality_type, pc = 1970) +
               s(living_space, k = k_other, by = municipality_type, pc = 140) +
               s(lot_size, k = k_other, by = municipality_type, pc = 800) +
               s(coast_distance, k = k_other, pc = 1500) +
               s(lake_distance, k = k_other, pc = 500) +
               s(stream_distance, k = k_other, pc = 1000) +
               s(road_distance, k = k_other, pc = 2000) +
               s(highway_distance, k = k_other, pc = 2000) +
               s(train_station_distance, k = k_other, pc = 2000) +
               s(railway_distance, k = k_other, pc = 1000) +
               s(windmill_distance, k = k_other, pc = 3000) +
               s(ocean_view, k = k_other, pc = 0) +
```

```
                s(lake_view, k = k_other, pc = 0) +
                s(forest_size, k = k_other, pc = 0) +
                s(number_of_bathrooms, by = municipality_type, k = 2, pc = 1) +
                house_type*municipality_type +
                roof_type * municipality_type +
                wall_type * municipality_type +
                heating_type * municipality_type,
                data = training_subset,
                family = tw(link = log),
                method = "fREML"
              )


  filename <- sprintf("Model%d_%d_%.2f.rds", i, k_spatial, fraction)
  saveRDS(model, file = file.path("checkpoints", filename))

  model

}

if (load) {
  models <- readRDS("saved_files/models.rds")
} else {
  models <- future_lapply(1:N, function(i) fit_fold(i, k_spatial = k_spatial))
  names(models) <- paste0("Model", 1:N)
  saveRDS(models, "saved_files/models.rds")
}
```

2. Partial effects

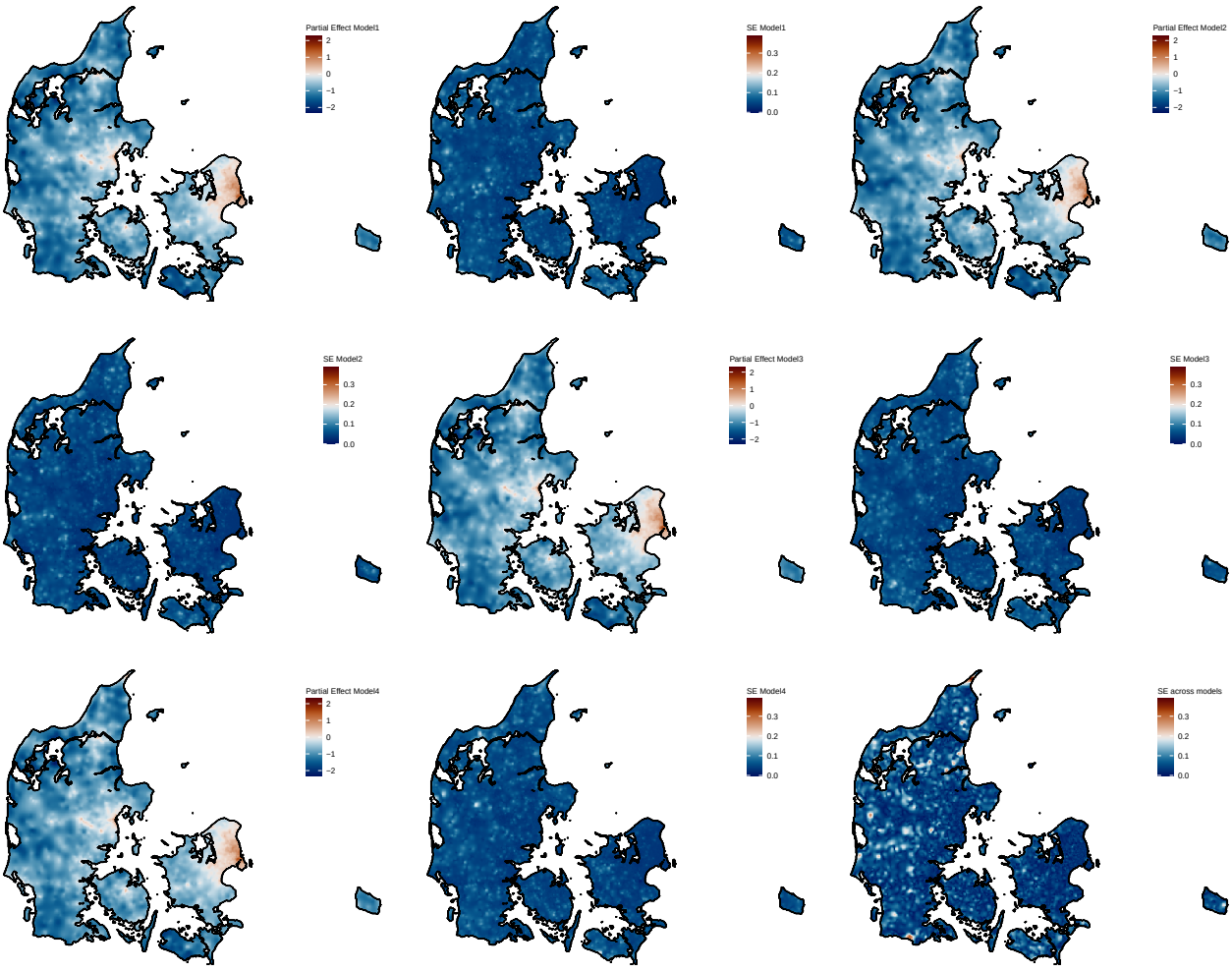
Spatial smooth The code below extracts the spatial smooth for each model and plots it alongside the log-scale standard error. The plotting relies on a premade spatial grid of Denmark, where ocean and land are marked.

```
# 1a) Predict once per model and store results
prediction_results <- future_lapply(seq_along(models), function(i) {
  model <- models[[i]]
  model_data <- model.frame(model)
  grid <- coordinate_grid

  for (var in setdiff(all.vars(formula(model)), names(grid))) {
    x <- model_data[[var]]
    if (is.factor(x)) {
      grid[[var]] <- factor(rep(levels(x)[1], nrow(grid)), levels = levels(x))
    } else {
      grid[[var]] <- 0
    }
  }

  terms <- predict.gam(model, grid, type = "terms", se.fit = TRUE)
  df_predictions <- as.data.frame(terms$fit)
  df_se <- as.data.frame(terms$se.fit)
```

```r
  effect <- df_predictions$`s(coordinate_east,coordinate_north)`
  se <- df_se$`s(coordinate_east,coordinate_north)`
  effect[grid$ocean] <- NA
  se[grid$ocean] <- NA

  grid$effect_coordinates <- effect
  grid$se_coordinates <- se

  list(model_name = names(models)[i],
       grid = grid)
})


# 1b) Compute SD across models (per grid cell) ---
effects_mat <- do.call(cbind, lapply(prediction_results, function(x) x$grid$effect_coordinates))
base_grid   <- prediction_results[[1]]$grid

se_across_models <- apply(effects_mat, 1, sd, na.rm = TRUE)
se_across_models[base_grid$ocean] <- NA

# 2. Global ranges (normalized across everything) ---
# Effects: symmetric across *all model* effects
all_effects <- unlist(lapply(prediction_results, function(x) x$grid$effect_coordinates))
max_abs_effect <- max(abs(all_effects), na.rm = TRUE)
effect_limits  <- c(-max_abs_effect, max_abs_effect)

# SEs: include per-model SEs AND the across-models SD
all_ses_models <- unlist(lapply(prediction_results, function(x) x$grid$se_coordinates))
se_limits <- c(0, max(c(all_ses_models, se_across_models), na.rm = TRUE))  # SEs are >= 0

# 3. Plotting function (adds land/ocean outline) ---
plot_model_terms_from_grid <- function(grid, model_name, effect_range, se_range) {

  plot_effect <- ggplot(data = grid) +
    geom_tile(aes(coordinate_east, coordinate_north, fill = effect_coordinates)) +
    labs(x = "", y = "") +
    scico::scale_fill_scico(
      palette = "vik",
      name = paste("Partial Effect", model_name),
      na.value = "white",
      limits = effect_range
    ) +
    geom_contour(
      aes(x = coordinate_east, y = coordinate_north, z = as.numeric(!ocean)),
      color = "black", size = 0.4
    ) +
    coord_fixed() +
    theme_void() +
    theme(
      text = element_text(size = 12),
      legend.position = c(0.85, 0.75),
      legend.title = element_text(size = 9, margin = margin(b = 5)),
      legend.text = element_text(size = 9),
```

```
      legend.title.align = 0.5
    )

  plot_se <- ggplot(data = grid) +
    geom_tile(aes(coordinate_east, coordinate_north, fill = se_coordinates)) +
    labs(x = "", y = "") +
    scico::scale_fill_scico(
      palette = "vik",
      name = paste("SE", model_name),
      na.value = "white",
      limits = se_range
    ) +
    geom_contour(
      aes(x = coordinate_east, y = coordinate_north, z = as.numeric(!ocean)),
      color = "black", size = 0.4
    ) +
    coord_fixed() +
    theme_void() +
    theme(
      text = element_text(size = 12),
      legend.position = c(0.85, 0.75),
      legend.title = element_text(size = 9, margin = margin(b = 5)),
      legend.text = element_text(size = 9),
      legend.title.align = 0.5
    )

  list(effect = plot_effect, se = plot_se)
}

# 4. Build plots for each model (effect + SE) ---
plot_lists <- lapply(prediction_results, function(result) {
  plot_model_terms_from_grid(
    result$grid, result$model_name,
    effect_range = effect_limits,
    se_range     = se_limits
  )
})

plot_spatial_smooths <- unlist(plot_lists, recursive = FALSE)

# 5. ONE extra plot: SD of effects across models (uses same se_limits) ---
base_grid2 <- base_grid
base_grid2$se_across_models <- se_across_models  # <- bake it into the data

plot_sd_across <- ggplot(data = base_grid2) +
  geom_tile(aes(coordinate_east, coordinate_north, fill = se_across_models)) +
  labs(x = "", y = "") +
  scico::scale_fill_scico(
    palette = "vik",
    name = "SE across models",
    na.value = "white",
    limits = se_limits
  ) +
```

```
  geom_contour(
    aes(x = coordinate_east, y = coordinate_north, z = as.numeric(!ocean)),
    color = "black", size = 0.4
  ) +
  coord_fixed() +
  theme_void() +
  theme(
    text = element_text(size = 12),
    legend.position = c(0.85, 0.75),
    legend.title = element_text(size = 9, margin = margin(b = 5)),
    legend.text = element_text(size = 9),
    legend.title.align = 0.5
  )

plot_spatial_smooths <- c(plot_spatial_smooths, list(plot_sd_across))
saveRDS(plot_spatial_smooths, "saved_files/plot_spatial_smooths.rds")
wrap_plots(plot_spatial_smooths, ncol = 3)
```

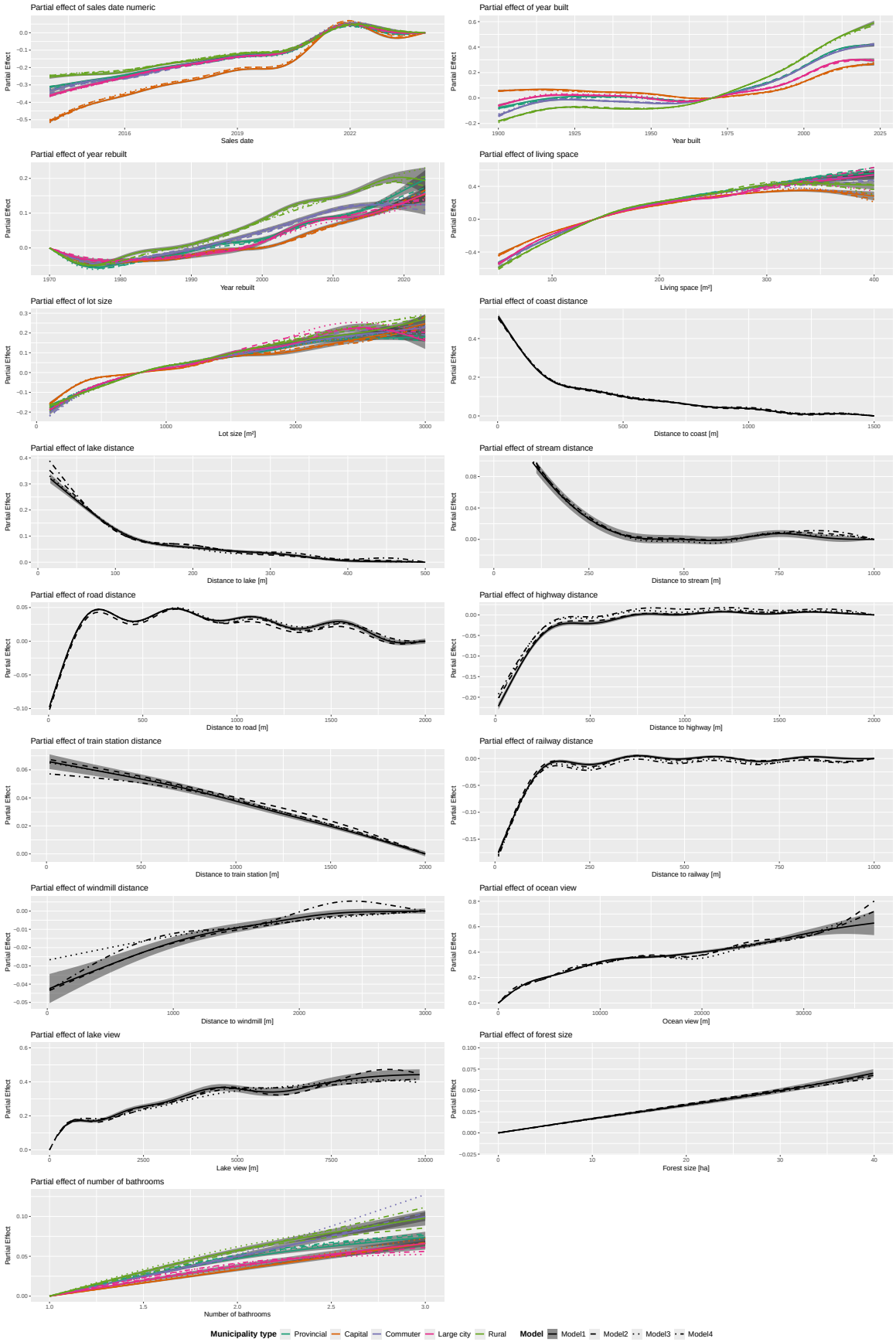


Other smooth effects The code below extracts and plots the remaining smooth effects of the trained GAM-models. A smooth corresponding to each trained model is plotted, but for readability confidence intervals are only displayed for Model1.

```r
smooth_variables <- c(
  sales_date_numeric = "Sales date",
  year_built = "Year built",
  year_rebuilt = "Year rebuilt",
  living_space = "Living space [m²]",
  lot_size = "Lot size [m²]",
  coast_distance = "Distance to coast [m]",
  lake_distance = "Distance to lake [m]",
  stream_distance = "Distance to stream [m]",
  road_distance = "Distance to road [m]",
  highway_distance = "Distance to highway [m]",
  train_station_distance = "Distance to train station [m]",
  railway_distance = "Distance to railway [m]",
  windmill_distance = "Distance to windmill [m]",
  ocean_view = "Ocean view [m]",
  lake_view = "Lake view [m]",
  forest_size = "Forest size [ha]",
  number_of_bathrooms = "Number of bathrooms")

plot_smooths <- list()

for (var in names(smooth_variables)) {

  smooth_dfs <- lapply(names(models), function(model_name) {
    mod <- models[[model_name]]
    df <- tryCatch(
      smooth_estimates(mod, smooth = paste0("s(", var, ")"), partial_match = TRUE),
      error = function(e) {
        message(sprintf("Skipping '%s' in %s: %s", var, model_name, e$message))
        return(NULL)
      })
    if (!is.null(df)) {
      df$Model <- model_name
    }
    df
  })

  smooth_dfs <- Filter(Negate(is.null), smooth_dfs)  # remove NULLs
  if (length(smooth_dfs) == 0) next

  combined_df <- bind_rows(smooth_dfs)

  has_by <- any(!is.na(combined_df$.by))
  plot_title <- paste("Partial effect of", gsub("_", " ", var))

  if (has_by) {

    by_var_name <- unique(na.omit(combined_df$.by))[[1]]

    p <- ggplot(combined_df, aes(x = .data[[var]], y = .estimate,
                                 color = .data[[by_var_name]],
                                 linetype = Model,
                                 group = interaction(Model, .data[[by_var_name]]))) +
```

```
        geom_ribbon(data = subset(combined_df, Model == "Model1"),
                    aes(ymin = .estimate - .se, ymax = .estimate + .se),
                    alpha = 0.5,
                    color = NA) +
        geom_line(linewidth = 1) +
        labs(title = plot_title,
             x = smooth_variables[[var]],
             y = "Partial Effect") +
        scale_color_brewer(palette = "Dark2", name = "Municipality type") +
        scale_linetype_manual(values = c("solid",
                                         "dashed",
                                         "dotted",
                                         "dotdash",
                                         "twodash",
                                         "longdash")) +
        guides(fill = guide_legend(override.aes = list(linetype = 0)))

  } else {

    p <- ggplot(combined_df, aes(x = .data[[var]], y = .estimate, linetype = Model)) +
         geom_ribbon(data = subset(combined_df, Model == "Model1"),
                     aes(ymin = .estimate - .se, ymax = .estimate + .se),
                     alpha = 0.5,
                     color = NA) +
         geom_line(linewidth = 1, color = "black") +
         labs(title = plot_title,
              x = smooth_variables[[var]],
              y = "Partial Effect") +
         scale_linetype_manual(values = c("solid",
                                          "dashed",
                                          "dotted",
                                          "dotdash",
                                          "twodash",
                                          "longdash"))
  }

  p <- p + theme(legend.position = "none")

  plot_smooths[[var]] <- p
}


saveRDS(plot_smooths, "saved_files/plot_smooths.rds")

#manual axis specifications of some ill-behaved smooths to make plots easier to understand
plot_smooths$stream_distance <- plot_smooths$stream_distance + ylim(-0.03,0.1)
plot_smooths$lake_view <- plot_smooths$lake_view + xlim(0.0,10000) + ylim(0, 0.6)
plot_smooths$forest_size <- plot_smooths$forest_size + xlim(0.0,40) + ylim(-0.02, 0.1)

wrap_plots(plot_smooths, ncol = 2) +
           plot_layout(guides = "collect") &
           theme(legend.position = "bottom",
                 legend.text = element_text(size = 12),
```

```
                legend.title = element_text(size = 13, face = "bold"),
                legend.box = "horizontal",
                legend.justification = "center")
```

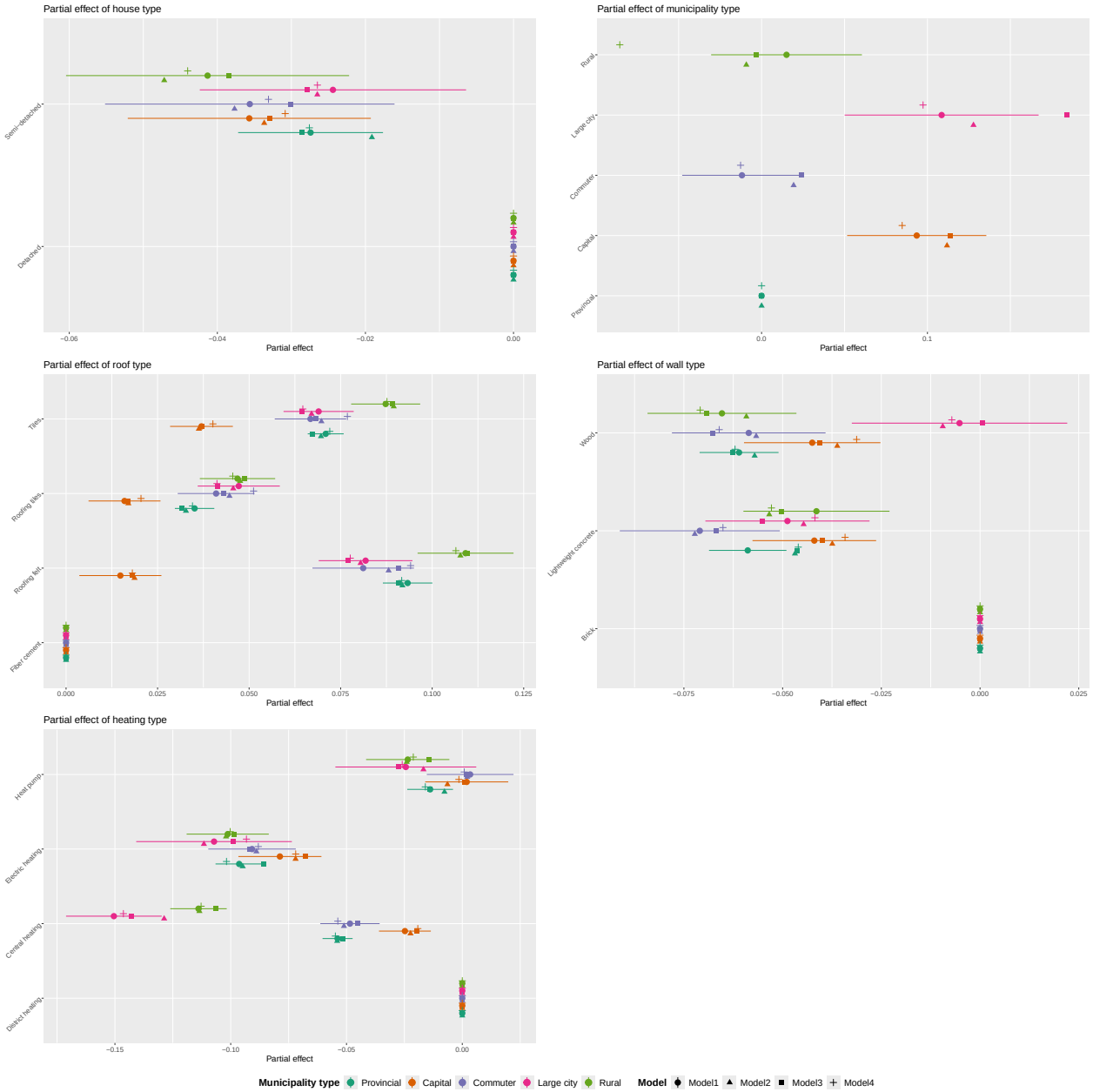

Partial effect of sales date numeric
Partial Effect
Sales date
Partial effect of year built
Year built
Partial effect of year rebuilt
Year rebuilt
Partial effect of living space
Living space [m²]
Partial effect of lot size
Lot size [m²]
Partial effect of coast distance
Distance to coast [m]
Partial effect of lake distance
Distance to lake [m]
Partial effect of stream distance
Distance to stream [m]
Partial effect of road distance
Distance to road [m]
Partial effect of highway distance
Distance to highway [m]
Partial effect of train station distance
Distance to train station [m]
Partial effect of railway distance
Distance to railway [m]
Partial effect of windmill distance
Distance to windmill [m]
Partial effect of ocean view
Ocean view [m]
Partial effect of lake view
Lake view [m]
Partial effect of forest size
Forest size [ha]
Partial effect of number of bathrooms
Number of bathrooms
Municipality type
Provincial
Capital
Commuter
Large city
Rural
Model
Model1
Model2
Model3
Model4

Factor effects The code below extracts and plots the factor effects of trained GAM-models. A point is shown for the effect of each model but error bars are shown only for Model1.

```
factor_variables <- c(
  house_type = "House type",
  municipality_type = "Municipality type",
  roof_type = "Roof type",
  wall_type = "Wall type",
  heating_type = "Heating type")

make_termwise_df <- function(model, model_data, variable, interaction_by = "municipality_type") {

  v_vals <- unique(model_data[[variable]])
  if (is.null(interaction_by)) {
    df <- data.frame(setNames(list(v_vals), variable), stringsAsFactors = FALSE)
  } else {
    m_vals <- unique(model_data[[interaction_by]])
    grid_list <- setNames(list(v_vals, m_vals), c(variable, interaction_by))
    df <- expand.grid(grid_list, stringsAsFactors = FALSE)
    df[[interaction_by]] <- factor(df[[interaction_by]],
                                   levels = levels(model_data[[interaction_by]]))
  }
  for (var in setdiff(all.vars(formula(model)), names(df))) {
    x <- model_data[[var]]
    if (is.factor(x)) {
      df[[var]] <- factor(rep(levels(x)[1], nrow(df)), levels = levels(x))
    } else {
      df[[var]] <- 0
    }
  }
  return(df)
}

combine_all_term_effects <- function(df, term, interaction_by = "municipality_type") {
  pattern <- paste0("^", term, "$|^", term, ":" ,"|:", term, "$")
  cols_to_sum <- grep(pattern, names(df), value = TRUE)
  df[[paste0("effect_", term)]] <- rowSums(as.matrix(df[cols_to_sum]))
  return(df)
}

combine_all_term_se <- function(df, term, interaction_by = "municipality_type") {
  pattern <- paste0("^", term, "$|^", term, ":" ,"|:", term, "$")
  cols_to_sum <- grep(pattern, names(df), value = TRUE)
  df[[paste0("se_", term)]] <- sqrt(rowSums(as.matrix(df[cols_to_sum])^2))
  return(df)
}

plot_term_effect <- function(df, term, interaction_by = "municipality_type") {
  effect_col <- paste0("effect_", term)
  lower_col <- paste0("lower_", effect_col)
  upper_col <- paste0("upper_", effect_col)
  df_sub <- df %>% filter(term == !!term)
  plot_title <- paste("Partial effect of", gsub("_", " ", term))
```

```
  p <- ggplot(df_sub, aes(x = .data[[term]],
                          y = .data[[effect_col]],
                          shape = Model,
                          group = interaction(Model, if (identical(term, interaction_by) ||
                                                         !(interaction_by %in% names(df_sub))) NULL
                                              else .data[[interaction_by]]))) +
      geom_pointrange(data = subset(df_sub, Model == "Model1"),
                      aes(ymin = .data[[lower_col]], ymax = .data[[upper_col]]),
                      position = position_dodge(width = 0.5),
                      size = 0.8) +
      geom_point(data = subset(df_sub, Model != "Model1"),
                 position = position_dodge(width = 0.5),
                 size = 2.5) +
      coord_flip() +
      labs(x = NULL, y = "Partial effect", title = plot_title, shape = "Model") +
      theme(axis.text.y = element_text(angle = 45, hjust = 1))

  if (interaction_by %in% names(df_sub)) {
      color_var <- if (term == interaction_by) term else interaction_by
      p <- p + aes(color = .data[[color_var]]) +
      scale_color_brewer(palette = "Dark2", name = "Municipality type")
  }

  return(p)
}

factor_dfs <- list()

for (model_name in names(models)) {

  model <- models[[model_name]]
  model_data <- model.frame(model)
  relevant_factors <- factor_variables[names(factor_variables) %in% all.vars(formula(model))]

  for (var in names(relevant_factors)) {

    interaction <- if (var == "municipality_type") NULL else "municipality_type"
    df <- make_termwise_df(model, model_data, var, interaction_by = interaction)
    df$term <- var

    terms <- predict.gam(model, df, type = "terms", se.fit = TRUE)
    df_predictions <- as.data.frame(terms$fit)
    df_se <- as.data.frame(terms$se.fit)

    combined_predictions <- combine_all_term_effects(df_predictions,
                                                     var,
                                                     interaction_by = interaction) %>%
                            select(starts_with("effect_"))

    combined_se <- combine_all_term_se(df_se,
                                       var,
                                       interaction_by = interaction) %>%
                   select(starts_with("se_"))
```

```
    df_final <- df %>% select(all_of(c(var, interaction)), term) %>%
                       bind_cols(combined_predictions) %>%
                       bind_cols(combined_se) %>%
                       mutate(across(starts_with("effect_"),
                                     ~ . - 1.96 * get(str_replace(cur_column(),
                                     "effect_",
                                     "se_")),
                                     .names = "lower_{.col}")) %>%
                       mutate(across(starts_with("effect_"),
                                     ~ . + 1.96 * get(str_replace(cur_column(), "effect_", "se_")),
                                     .names = "upper_{.col}"))

    df_final$Model <- model_name
    factor_dfs[[length(factor_dfs) + 1]] <- df_final
  }
}

df_all_factors <- bind_rows(factor_dfs)
plot_factors <- lapply(names(factor_variables),
                       function(term) {plot_term_effect(df_all_factors, term)})
names(plot_factors) <- names(factor_variables)

saveRDS(plot_factors, "saved_files/plot_factors.rds")

wrap_plots(plot_factors, ncol = 2) +
           plot_layout(guides = "collect") &
           theme(legend.position = "bottom",
                 legend.text = element_text(size = 12),
                 legend.title = element_text(size = 13, face = "bold"),
                 legend.box = "horizontal",
                 legend.justification = "center")
```



3. Model performance

The code below extracts various residual types and calculates different performance measures for the training and validation set. Note that using N-fold cross-validation means that there are N-1 estimates for each training point, and the training error is simply taken as the average over these.

```
get_residuals <- function(models, dataset) {

  residual_list <- future_lapply(seq_along(models), function(i) {

    model_name <- names(models)[i]
    model <- models[[i]]
    fold <- as.integer(sub("^Model(\\d+).*$", "\\1", model_name))
```

```
    sales_price <- dataset$sales_price
    predicted_sales_price <- predict(model, newdata = dataset, type = "response")
    residual_sales_price <- predicted_sales_price - sales_price
    residual_sales_price_percent <- residual_sales_price / sales_price * 100
    residual_sales_price_log <- log(predicted_sales_price) - log(sales_price)
    validation <- dataset$validation_set == fold
    test <- dataset$test_set

    df <- as.data.frame(list("sales_price" = sales_price,
                             "predicted_sales_price" = predicted_sales_price,
                             "residual_sales_price" = residual_sales_price,
                             "residual_sales_price_percent" = residual_sales_price_percent,
                             "residual_sales_price_log" = residual_sales_price_log,
                             "ratio" = predicted_sales_price / sales_price,
                             "validation" = validation,
                             "test" = test,
                             "id" = dataset$id,
                             "municipality_type" = dataset$municipality_type,
                             "Model" = model_name))

  })

  bind_rows(residual_list)

}


residual_df <- get_residuals(models, dataset)

performance_countrywide <- residual_df %>%
                           mutate(dataset_type = case_when(test ~ "Test",
                                                           validation ~ "Validation",
                                                           TRUE ~ "Training")) %>%
                                 group_by(dataset_type) %>%
                                 summarise(municipality_type = "Countrywide",
                                           MALE = mean(abs(residual_sales_price_log)),
                                           MAPE = mean(abs(residual_sales_price_percent)),
                                           RMSE = sqrt(mean(residual_sales_price^2)),
                                           PM20 = mean(abs(residual_sales_price_percent) <= 20) * 100,
                                           COD = mean(abs(ratio - median(ratio))) / median(ratio) * 10
                                           PRD = mean(ratio) / (sum(predicted_sales_price)/sum(sales_pr
                                           .groups = "drop")


performance_municipality_type <- residual_df %>%
                                 mutate(dataset_type = case_when(test ~ "Test",
                                                                 validation ~ "Validation",
                                                                 TRUE ~ "Training")) %>%
                                 group_by(dataset_type, municipality_type) %>%
                                 summarise(MALE = mean(abs(residual_sales_price_log)),
                                           MAPE = mean(abs(residual_sales_price_percent)),
                                           RMSE = sqrt(mean(residual_sales_price^2)),
                                           PM20 = mean(abs(residual_sales_price_percent) <= 20) * 100,
```

```
                                        COD = mean(abs(ratio - median(ratio))) / median(ratio) * 100
                                        .groups = "drop")


performance <- bind_rows(performance_countrywide, performance_municipality_type)

municipality_levels <- c("Countrywide", sort(setdiff(unique(performance$municipality_type), "Countrywide

performance_sorted <- performance %>%
                      mutate(dataset_type = factor(dataset_type, levels = c("Training", "Validation", "
                             municipality_type = factor(municipality_type, levels = municipality_levels
                             RMSE = round(RMSE,0),
                             PRD = ifelse(is.na(PRD), "", round(PRD, 2))) %>%
                      arrange(municipality_type, dataset_type)  %>%
                      rename(Type = dataset_type,
                             `Municipality type` = municipality_type,
                             `MAPE [%]` = MAPE,
                             `RMSE [kr.]` = RMSE,
                             `PM20 [%]` = PM20,
                             `COD [%]` = COD)

#standard table
#knitr::kable(performance_sorted, digits = 2, caption = "Model Performance by Municipality Type")

#nicer table
kable(performance_sorted %>%
select(-`Municipality type`),
       digits = 2,
       caption = "Model Performance by Municipality Type",
       booktabs = TRUE) %>%
kable_styling(full_width = FALSE, position = "center") %>%
group_rows("Countrywide", 1, 3) %>%
group_rows("Capital", 4, 6) %>%
group_rows("Commuter", 7, 9) %>%
group_rows("Large city", 10, 12) %>%
group_rows("Provincial", 13, 15) %>%
group_rows("Rural", 16, 18)
```

Table 1: Model Performance by Municipality Type

Type	MALE	MAPE [%]	RMSE [kr.]	PM20 [%]	COD [%]	PRD
Countrywide						
Training	0.17	18.84	520085	71.36	18.72	1.06
Validation	0.17	19.17	529261	70.78	19.05	1.07
Test	0.17	19.23	534130	71.00	19.05	1.07
Capital						
Training	0.11	11.43	729896	86.02	11.40	
Validation	0.11	11.55	740084	85.79	11.52	
Test	0.11	11.41	761328	85.19	11.36	
Commuter						
Training	0.17	18.73	423918	71.34	18.64	
Validation	0.17	19.13	433749	70.49	19.03	

Test	0.17	19.33	404461	71.18	19.21
Large city					
Training	0.14	14.64	564377	78.35	14.58
Validation	0.14	14.86	574716	77.87	14.80
Test	0.14	15.13	599554	77.55	14.95
Provincial					
Training	0.16	17.87	499929	71.93	17.76
Validation	0.16	18.15	508628	71.39	18.03
Test	0.16	18.25	509348	71.61	18.04
Rural					
Training	0.23	27.17	387730	57.24	26.79
Validation	0.24	27.70	396238	56.52	27.31
Test	0.23	27.58	397941	57.35	27.16

```
tbl_latex <- kable(performance_sorted %>% select(-`Municipality type`),
                   format = "latex", booktabs = TRUE, digits = 2,
                   caption = "Model performance by municipality type.",
                   label = "perf_by_muni") %>%
             kable_styling(full_width = FALSE,
                           position = "center",
                           latex_options = c("hold_position")) %>%
             group_rows("Countrywide", 1, 3) %>%
             group_rows("Capital",      4, 6) %>%
             group_rows("Commuter",     7, 9) %>%
             group_rows("Large city",  10,12) %>%
             group_rows("Provincial",  13,15) %>%
             group_rows("Rural",       16,18)

writeLines(tbl_latex, "saved_files/performance_by_muni.tex")

residual_df_test <- residual_df %>%
                    filter(test, Model == "Model1") %>%
                    mutate(pm20 = as.integer(abs(residual_sales_price_percent) <= 20))


plot_test_residuals_log <- ggplot(residual_df_test,
                                  aes(x = sales_price / 1e6,
                                      y = residual_sales_price_log,
                                      color = municipality_type)) +
                           geom_point(alpha = 0.6, size = 0.25) +
                           geom_smooth(aes(x = sales_price / 1e6,
                                           y = abs(residual_sales_price_log),
                                           linetype = "MALE"),
                                       se = FALSE,
                                       color = "black",
                                       size = 0.8,
                                       inherit.aes = FALSE) +
                           scale_linetype_manual(name = NULL,
                                                 values = c(MALE = "solid"),
                           guide = guide_legend(override.aes = list(color = "black", size = 0.8),
                                                nrow = 1,
                                                byrow = TRUE)) +
```

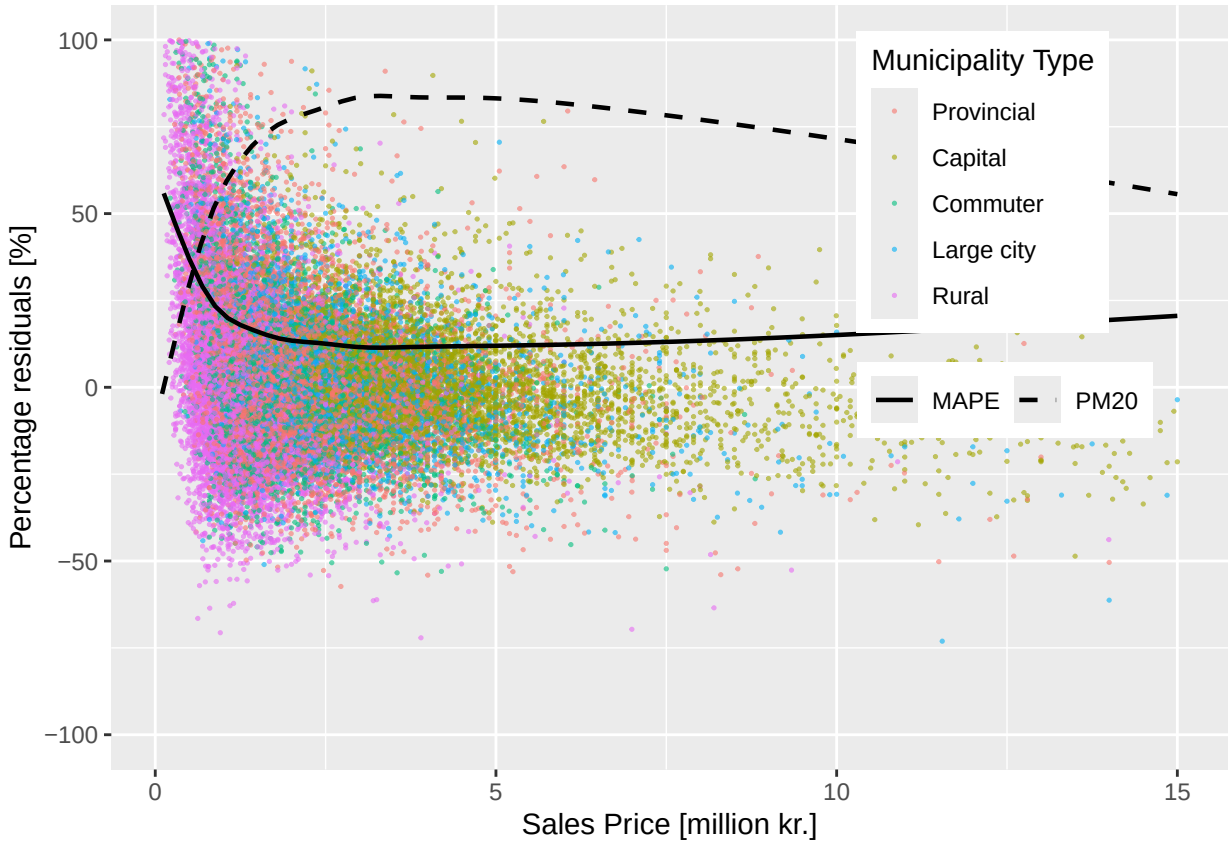
```
                          ylim(-1, 1) +
                          labs(x = "Sales Price [million kr.]", y = "Log Residuals", color = "Municipal
                          theme(legend.position = c(0.8, 0.7))

saveRDS(plot_test_residuals_log, "saved_files/plot_test_residuals_log.rds")
print(plot_test_residuals_log)
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```


```
plot_test_residuals_percent <- ggplot(residual_df_test,
                                      aes(x = sales_price / 1e6,
                                          y = residual_sales_price_percent,
                                          color = municipality_type)) +
                               geom_point(alpha = 0.6, size = 0.25) +

                               # Absolute residuals line (MAPE) - inherit.aes = FALSE to decouple from
                               geom_smooth(aes(x = sales_price / 1e6,
                                               y = abs(residual_sales_price_percent),
                                               linetype = "MAPE"),
                                           se = FALSE,
                                           color = "black",
                                           size = 0.8,
                                           inherit.aes = FALSE) +

                                # pm20 probability line
```

```
                              geom_smooth(aes(x = sales_price / 1e6,
                                              y = pm20 * 100,
                                              linetype = "PM20"),
                                          se = FALSE,
                                          color = "black",
                                          size = 0.8,
                                          inherit.aes = FALSE) +
                              scale_linetype_manual(name = NULL,
                                                    values = c(MAPE = "solid", PM20 = "dashed"),
                                                    guide = guide_legend(override.aes = list(color =
                                                                         size = 0.8),
                                                    nrow = 1, byrow = TRUE)) +
                              ylim(-100, 100) +
                              labs(x = "Sales Price [million kr.]",
                                   y = "Percentage residuals [%]",
                                   color = "Municipality Type") +
                              theme(legend.position = c(0.8, 0.7))


saveRDS(plot_test_residuals_percent, "saved_files/plot_test_residuals_percent.rds")
print(plot_test_residuals_percent)
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



```r
plot_test_residuals <- ggplot(residual_df_test,
                              aes(x = sales_price / 1e6,
                                  y = residual_sales_price / 1e6,
                                  color = municipality_type)) +
                       geom_point(alpha = 0.6, size = 0.25) +

                       # RMSE line, decoupled from color mapping
                       geom_smooth(aes(x = sales_price / 1e6,
                                       y = abs(residual_sales_price) / 1e6,
                                       linetype = "RMSE"),
                                   se = FALSE,
                                   color = "black",
                                   size = 0.8,
                                   inherit.aes = FALSE) +

                       scale_linetype_manual(name = NULL,
                                             values = c(RMSE = "solid"),
                                             guide = guide_legend(override.aes = list(color = "black", s
                                                                  nrow = 1,
                                                                  byrow = TRUE)) +
                       ylim(-3, 3) +
                       labs(x = "Sales Price [million kr.]", y = "Residuals [million kr.]", color = "Mu
                       theme(legend.position = c(0.8, 0.7)) # inside top-right



saveRDS(plot_test_residuals, "saved_files/plot_test_residuals.rds")
print(plot_test_residuals)
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

4. Model interpretation

```
model <- models[[1]]
model_data <- model.frame(model)

#generate some fictitious properties. One in reference point, one close to coast near Esbjerg, one clos
grid <- as.data.frame(list(coordinate_east = c(697000, 462037, 469142, 725599),
                           coordinate_north = c(6150000, 6150225, 6322298, 6185920)))

point_constraints <- list()
for (sm in model$smooth) {
  if (!is.null(sm$point.con)) {
    point_constraints <- c(point_constraints, sm$point.con)
  }
}

# Get all variables used in model
vars_in_model <- all.vars(formula(model))

# Fill missing variables in grid using point constraints (or defaults)
for (var in setdiff(vars_in_model, names(grid))) {
  if (is.factor(model_data[[var]])) {
    # Use reference level for factor
    grid[[var]] <- factor(rep(levels(model_data[[var]])[1], nrow(grid)),
                          levels = levels(model_data[[var]]))
```

```
  } else {
    # Use point constraint if available, else use mean
    value <- if (var %in% names(point_constraints)) {
      point_constraints[[var]]
    } else {
      mean(model_data[[var]], na.rm = TRUE)
    }
    grid[[var]] <- rep(value, nrow(grid))
  }
}

#augment properties
grid$coast_distance[2:4] <- 100 #close to coast
grid$year_built[2:4] <- 2020 #new house
grid$municipality_type[2:4] <- factor(c("Provincial", "Rural", "Capital"))
grid$type <- c("Reference", "Esbjerg", "Nordjylland", "Strandvejen")

terms <- as.data.frame(predict.gam(model, grid, type = "terms"))


adjustments <- terms %>% mutate(

                          # Intercept
                          standard_price = exp(as.vector(coef(model))[1]),

                          # Non-varying smooth
                          adjustment_coordinates = exp(`s(coordinate_east,coordinate_north)`),

                          # Smooths with municipality_type interactions -
                          #might as well sum over all is it only non-zero for the relevant smooth

                          adjustment_sales_date = exp(
                             `s(sales_date_numeric):municipality_typeCapital` +
                               `s(sales_date_numeric):municipality_typeRural` +
                               `s(sales_date_numeric):municipality_typeCommuter` +
                               `s(sales_date_numeric):municipality_typeProvincial` +
                               `s(sales_date_numeric):municipality_typeLarge city`),

                          adjustment_year_built = exp(
                               `s(year_built):municipality_typeCapital` +
                                 `s(year_built):municipality_typeRural` +
                                 `s(year_built):municipality_typeCommuter` +
                                 `s(year_built):municipality_typeProvincial` +
                                 `s(year_built):municipality_typeLarge city`),

                          adjustment_year_rebuilt = exp(
                               `s(year_rebuilt):municipality_typeCapital` +
                                 `s(year_rebuilt):municipality_typeRural` +
                                 `s(year_rebuilt):municipality_typeCommuter` +
                                 `s(year_rebuilt):municipality_typeProvincial` +
                                 `s(year_rebuilt):municipality_typeLarge city`),

                          adjustment_living_space = exp(
```

```
      `s(living_space):municipality_typeCapital` +
        `s(living_space):municipality_typeRural` +
        `s(living_space):municipality_typeCommuter` +
        `s(living_space):municipality_typeProvincial` +
        `s(living_space):municipality_typeLarge city`),

    adjustment_lot_size = exp(
      `s(lot_size):municipality_typeCapital` +
        `s(lot_size):municipality_typeRural` +
        `s(lot_size):municipality_typeCommuter` +
        `s(lot_size):municipality_typeProvincial` +
        `s(lot_size):municipality_typeLarge city`),

    adjustment_coast_distance = exp(`s(coast_distance)`),

    adjustment_lake_distance = exp(`s(lake_distance)`),

    adjustment_stream_distance = exp(`s(stream_distance)`),

    adjustment_road_distance = exp(`s(road_distance)`),

    adjustment_highway_distance = exp(`s(highway_distance)`),

    adjustment_train_station_distance = exp(`s(train_station_distance)`),

    adjustment_railway_distance = exp(`s(railway_distance)`),

    adjustment_windmill_distance = exp(`s(windmill_distance)`),

    adjustment_ocean_view = exp(`s(ocean_view)`),

    adjustment_lake_view = exp(`s(lake_view)`),

    adjustment_forest_size = exp(`s(forest_size)`),

    adjustment_number_of_bathrooms = exp(
       `s(number_of_bathrooms):municipality_typeCapital` +
        `s(number_of_bathrooms):municipality_typeRural` +
        `s(number_of_bathrooms):municipality_typeCommuter` +
        `s(number_of_bathrooms):municipality_typeProvincial` +
        `s(number_of_bathrooms):municipality_typeLarge city`
    ),


    # Factor terms with interactions
    adjustment_house_type = exp(house_type),
    adjustment_municipality_type = exp(municipality_type),
    adjustment_roof_type = exp(roof_type + `municipality_type:roof_type`),
    adjustment_wall_type = exp(wall_type + `municipality_type:wall_type`),
    adjustment_heating_type = exp(heating_type + `municipality_type:heating_type`)

    adjustment_location = adjustment_coordinates *
                          adjustment_sales_date *
```

```
                                adjustment_lot_size *
                                adjustment_coast_distance *
                                adjustment_lake_distance *
                                adjustment_stream_distance *
                                adjustment_road_distance *
                                adjustment_highway_distance *
                                adjustment_train_station_distance *
                                adjustment_railway_distance *
                                adjustment_windmill_distance *
                                adjustment_ocean_view *
                                adjustment_lake_view *
                                adjustment_municipality_type,

         adjustment_property = adjustment_year_built *
                               adjustment_year_rebuilt *
                               adjustment_living_space *
                               adjustment_roof_type *
                               adjustment_wall_type *
                               adjustment_heating_type,

         adjustment_total = adjustment_location * adjustment_property,

        predicted_price = standard_price * adjustment_total) %>%

        select(standard_price,
               adjustment_municipality_type,
               adjustment_coordinates,
               adjustment_sales_date,
               adjustment_year_built,
               adjustment_year_rebuilt,
               adjustment_living_space,
               adjustment_lot_size,
               adjustment_coast_distance,
               adjustment_lake_distance,
               adjustment_stream_distance,
               adjustment_road_distance,
               adjustment_highway_distance,
               adjustment_train_station_distance,
               adjustment_railway_distance,
               adjustment_windmill_distance,
               adjustment_ocean_view,
               adjustment_lake_view,
               adjustment_forest_size,
               adjustment_number_of_bathrooms,
               adjustment_house_type,
               adjustment_roof_type,
               adjustment_wall_type,
               adjustment_heating_type,
               adjustment_location,
               adjustment_property,
               adjustment_total,
               predicted_price)
```

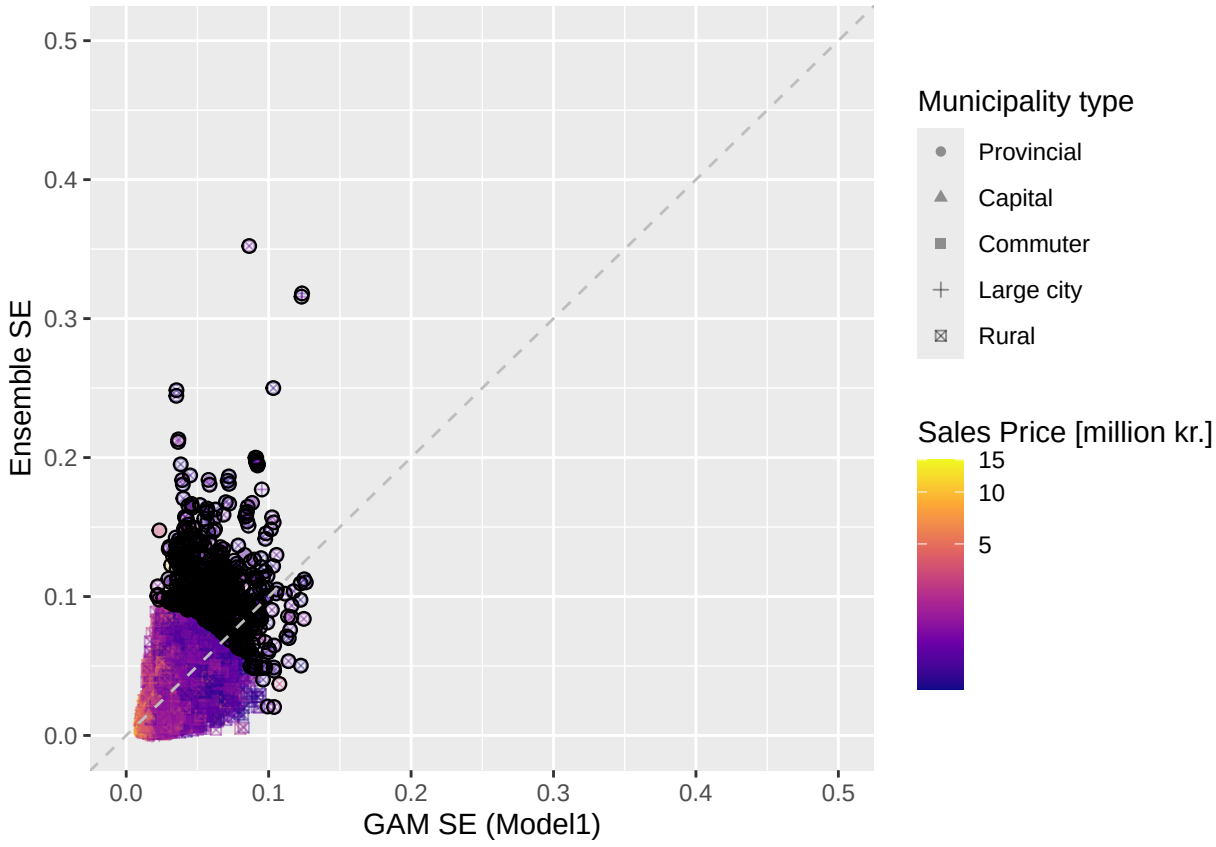
```
grid <- grid %>% select(coast_distance, year_built, municipality_type, type)

adjustments <- adjustments %>%
               select(standard_price,
                      adjustment_municipality_type,
                      adjustment_coordinates,
                      adjustment_year_built,
                      adjustment_coast_distance,
                      adjustment_location,
                      adjustment_property,
                      adjustment_total,
                      predicted_price)

results <- bind_cols(grid, adjustments)
```

4. Uncertainty detection

The code below flags outliers based on both the GAM-based uncertainty estimates as well as the ensemble-based robustness measure.

```
test_set <- dataset %>% filter(test_set)


predictions <- imap(models, function(model, name) {
  preds <- predict.gam(model, test_set, type = "link", se.fit = TRUE)
  tibble(!!paste0("pred_", name) := as.vector(preds$fit),
         !!paste0("se_", name) := as.vector(preds$se.fit))
})

pred_matrix <- bind_cols(predictions)

# Add to test_set
test_set <- bind_cols(test_set, pred_matrix)

pred_cols <- grep("^pred_", names(test_set), value = TRUE)

se_limit <- 0.1

test_set <- test_set %>%
            rowwise() %>%
            mutate(se_Models = sd(c_across(all_of(pred_cols))),
                   se_combined = sqrt(se_Model1^2 + se_Models^2),
                   above_limit = se_combined > se_limit) %>%
            ungroup()

plot_uncertainties <- ggplot(test_set, aes(x = se_Model1,
                                           y = se_Models,
                                           color = sales_price / 1e6,
                                           shape = municipality_type)) +
                      geom_point(alpha = 0.4) +
                      geom_point(data = subset(test_set, above_limit),
                                 aes(x = se_Model1, y = se_Models),
```

```
                              color = "black", shape = 1, size = 2, stroke = 0.6) +
                    geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray") +
                    scale_color_viridis_c(option = "plasma", trans = "log1p", name = "Sales Price [mi
                    scale_shape_discrete(name = "Municipality type") +
                    xlim(0, 0.5) +
                    ylim(0, 0.5) +
                    labs(x = "GAM SE (Model1)",
                         y = "Ensemble SE",
                         color = "Sales Price [million kr.]")

saveRDS(plot_uncertainties, "saved_files/plot_uncertainties.rds")
print(plot_uncertainties)
```

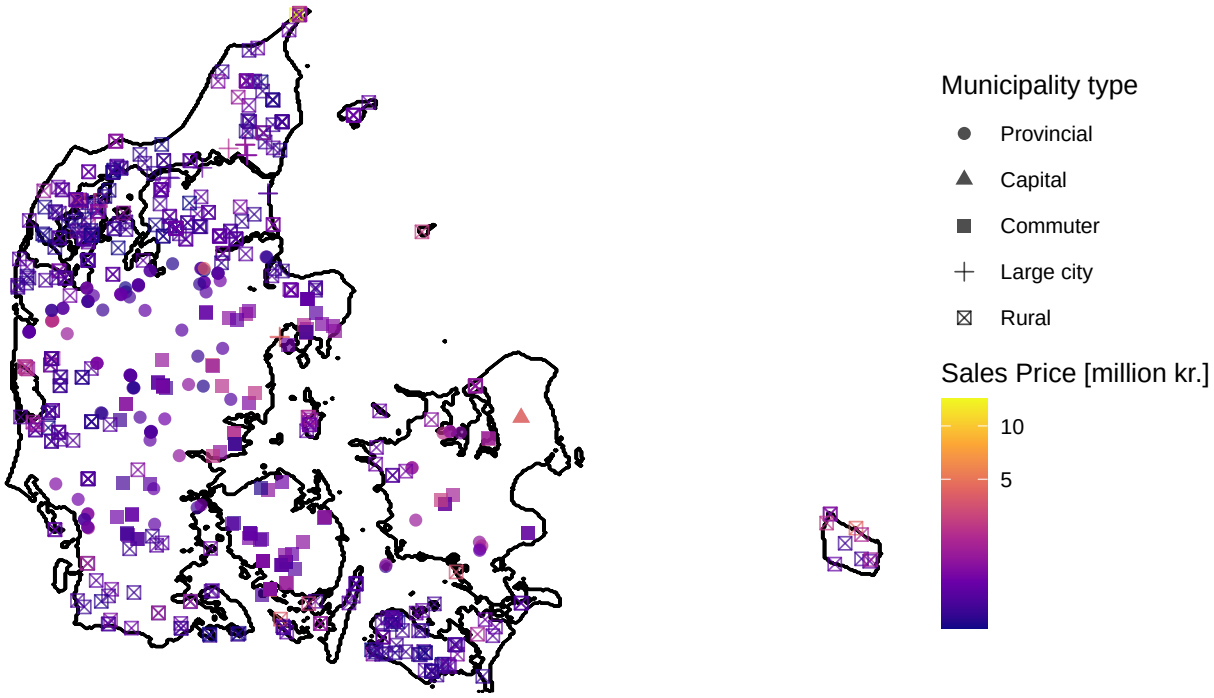



```
#Plot outliers geographically
outliers <- test_set %>% filter(above_limit)

plot_outliers_geographically <- ggplot() +
                                # Add land/ocean contour
                                geom_contour(data = coordinate_grid,
                                             aes(x = coordinate_east,
                                                 y = coordinate_north,
                                                 z = as.numeric(!ocean)),
                                             color = "black", size = 0.4) +
                                # Outliers
                                geom_point(data = outliers,
                                           aes(x = coordinate_east,
```

```
                                        y = coordinate_north,
                                        color = sales_price / 1e6,
                                        shape = municipality_type),
                                    size = 2.0, alpha = 0.7) +
                        scale_color_viridis_c(option = "plasma", trans = "log1p") +
                        coord_fixed() +
                        theme_void(base_size = 10) +
                        scale_shape_discrete(name = "Municipality type") +
                        scale_color_viridis_c(option = "plasma",
                                              trans = "log1p",
                                              name = "Sales Price [million kr.]") +
                        theme(plot.background = element_rect(fill = "white", color = NA),
                              panel.background = element_rect(fill = "white", color = NA))
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```

```
saveRDS(plot_outliers_geographically, "saved_files/plot_outliers_geographically.rds")
print(plot_outliers_geographically)
```



5. Model checking

Standard Model Checks

The code below plots various standard diagnostics for Model1 using the appraise() function from the gratia library as described in https://gavinsimpson.github.io/gratia/reference/appraise.html.

```r
k.check(models$Model1)
```

```
##                                                  k'         edf   k-index
## s(coordinate_east,coordinate_north)            2999 2604.026668 0.9750053
## s(sales_date_numeric):municipality_typeProvincial     9    8.847276 0.9984213
## s(sales_date_numeric):municipality_typeCapital        9    8.956035 0.9984213
## s(sales_date_numeric):municipality_typeCommuter       9    8.705528 0.9984213
## s(sales_date_numeric):municipality_typeLarge city     9    8.746595 0.9984213
## s(sales_date_numeric):municipality_typeRural          9    8.757616 0.9984213
## s(year_built):municipality_typeProvincial             9    8.772137 0.9789178
## s(year_built):municipality_typeCapital                9    8.581911 0.9789178
## s(year_built):municipality_typeCommuter               9    8.451849 0.9789178
## s(year_built):municipality_typeLarge city             9    8.594896 0.9789178
## s(year_built):municipality_typeRural                  9    8.376008 0.9789178
## s(year_rebuilt):municipality_typeProvincial           9    7.521685 0.9990118
## s(year_rebuilt):municipality_typeCapital              9    7.143056 0.9990118
## s(year_rebuilt):municipality_typeCommuter             9    6.115138 0.9990118
## s(year_rebuilt):municipality_typeLarge city           9    6.693677 0.9990118
## s(year_rebuilt):municipality_typeRural                9    7.547821 0.9990118
## s(living_space):municipality_typeProvincial           9    7.463673 0.9890568
## s(living_space):municipality_typeCapital              9    7.463260 0.9890568
## s(living_space):municipality_typeCommuter             9    6.778289 0.9890568
## s(living_space):municipality_typeLarge city           9    7.890333 0.9890568
## s(living_space):municipality_typeRural                9    7.589384 0.9890568
## s(lot_size):municipality_typeProvincial               9    6.859765 1.0195635
## s(lot_size):municipality_typeCapital                  9    7.762616 1.0195635
## s(lot_size):municipality_typeCommuter                 9    6.665045 1.0195635
## s(lot_size):municipality_typeLarge city               9    7.204634 1.0195635
## s(lot_size):municipality_typeRural                    9    6.415238 1.0195635
## s(coast_distance)                                     9    8.881031 1.0006721
## s(lake_distance)                                      9    6.942578 1.0011482
## s(stream_distance)                                    9    6.182903 0.9963925
## s(road_distance)                                      9    8.941469 1.0205525
## s(highway_distance)                                   9    8.835869 1.0120398
## s(train_station_distance)                             9    2.539631 1.0158914
## s(railway_distance)                                   9    8.868763 1.0316832
## s(windmill_distance)                                  9    3.126808 1.0121149
## s(ocean_view)                                         9    7.579546 0.9990492
## s(lake_view)                                          9    8.629777 1.0079746
## s(forest_size)                                        9    2.643188 0.9683385
## s(number_of_bathrooms):municipality_typeProvincial    2    1.889946 1.0249361
## s(number_of_bathrooms):municipality_typeCapital       2    1.001868 1.0249361
## s(number_of_bathrooms):municipality_typeCommuter      2    1.001871 1.0249361
## s(number_of_bathrooms):municipality_typeLarge city    2    1.571666 1.0249361
## s(number_of_bathrooms):municipality_typeRural         2    1.761222 1.0249361
##                                                 p-value
## s(coordinate_east,coordinate_north)              0.0200
## s(sales_date_numeric):municipality_typeProvincial  0.5325
## s(sales_date_numeric):municipality_typeCapital     0.4750
## s(sales_date_numeric):municipality_typeCommuter    0.4950
## s(sales_date_numeric):municipality_typeLarge city  0.4850
## s(sales_date_numeric):municipality_typeRural       0.4850
## s(year_built):municipality_typeProvincial          0.0625
```

```
## s(year_built):municipality_typeCapital              0.0775
## s(year_built):municipality_typeCommuter             0.0775
## s(year_built):municipality_typeLarge city           0.1000
## s(year_built):municipality_typeRural                0.0725
## s(year_rebuilt):municipality_typeProvincial         0.5150
## s(year_rebuilt):municipality_typeCapital            0.5200
## s(year_rebuilt):municipality_typeCommuter           0.4950
## s(year_rebuilt):municipality_typeLarge city         0.5000
## s(year_rebuilt):municipality_typeRural              0.5375
## s(living_space):municipality_typeProvincial         0.2550
## s(living_space):municipality_typeCapital            0.2725
## s(living_space):municipality_typeCommuter           0.2575
## s(living_space):municipality_typeLarge city         0.2350
## s(living_space):municipality_typeRural              0.2375
## s(lot_size):municipality_typeProvincial             0.9350
## s(lot_size):municipality_typeCapital                0.9225
## s(lot_size):municipality_typeCommuter               0.9400
## s(lot_size):municipality_typeLarge city             0.9200
## s(lot_size):municipality_typeRural                  0.9250
## s(coast_distance)                                   0.5875
## s(lake_distance)                                    0.5350
## s(stream_distance)                                  0.4200
## s(road_distance)                                    0.9425
## s(highway_distance)                                 0.7925
## s(train_station_distance)                           0.8425
## s(railway_distance)                                 0.9925
## s(windmill_distance)                                0.8225
## s(ocean_view)                                       0.4825
## s(lake_view)                                        0.7675
## s(forest_size)                                      0.0175
## s(number_of_bathrooms):municipality_typeProvincial  0.9675
## s(number_of_bathrooms):municipality_typeCapital     0.9700
## s(number_of_bathrooms):municipality_typeCommuter    0.9625
## s(number_of_bathrooms):municipality_typeLarge city  0.9625
## s(number_of_bathrooms):municipality_typeRural       0.9700
```

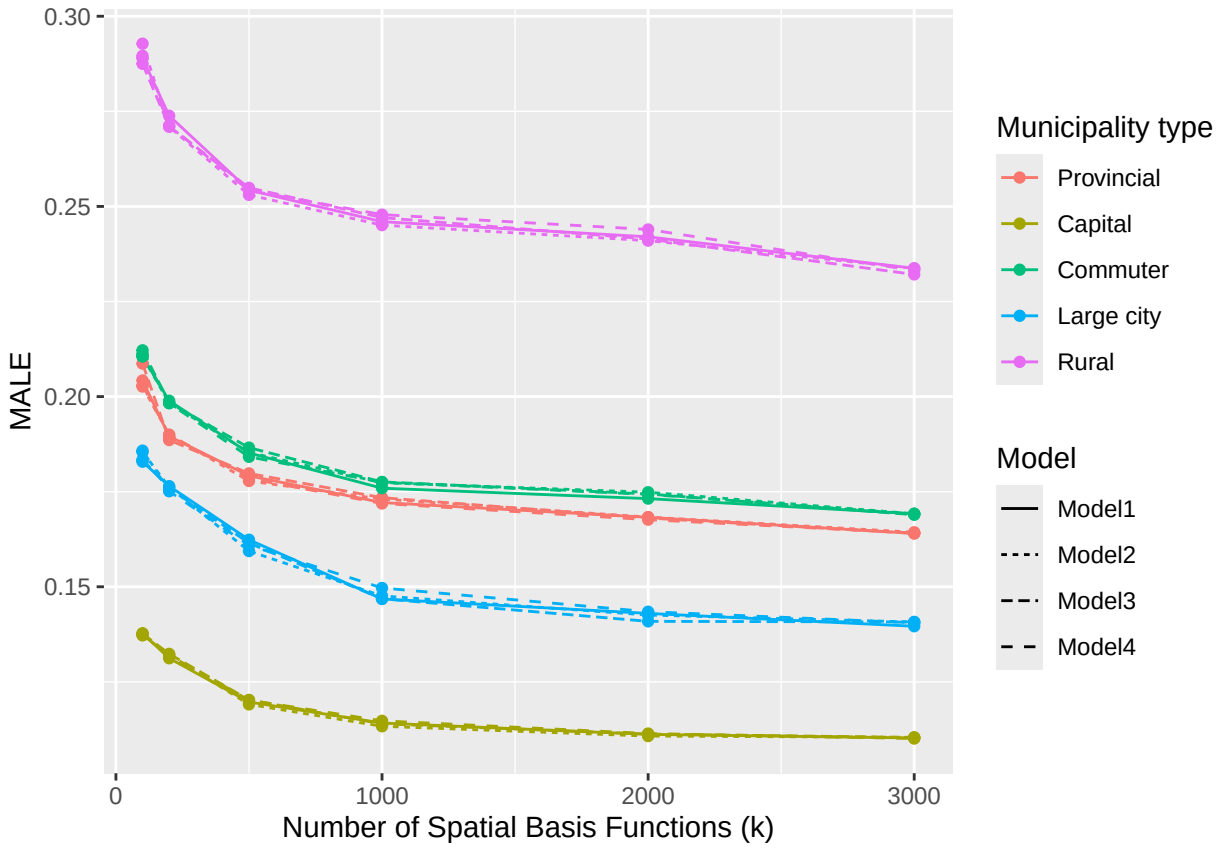
```
plot_appraise <- appraise(models$Model1)
saveRDS(plot_appraise, "saved_files/plot_appraise.rds")
```

Convergence of the spatial smooth

The code below fits the model ensemble for different basis set sizes of the spatial smooth to examine the convergence of the test error.

```
if (load) {
  models_basis <- readRDS("saved_files/models_basis.rds")
} else {

  folds <- 1:N
  k_values <- c(100, 200, 500, 1000, 2000, 3000, 4000)
  k_values <- k_values[k_values <= k_spatial]

  model_grid <- expand.grid(fold = folds, k_spatial = k_values)
```

```
  models_basis <- future_lapply(1:nrow(model_grid), function(idx) {
    row <- model_grid[idx, ]
    fit_fold(row$fold, row$k_spatial)})

  names(models_basis) <- paste0("Model", model_grid$fold, "_", model_grid$k_spatial)
  saveRDS(models_basis, "saved_files/models_basis.rds")
}




residual_df <- get_residuals(models_basis, dataset)

residual_df$k <- as.numeric(sub(".*_(\\d+)$", "\\1", residual_df$Model))
residual_df$Model <- sub("(_.*)$", "", residual_df$Model)


performance <- residual_df %>%
               mutate(dataset_type = case_when(test ~ "Test",
                                               validation ~ "Validation",
                                               TRUE ~ "Training")) %>%
                      group_by(dataset_type, municipality_type, k, Model) %>%
               summarise(MALE = mean(abs(residual_sales_price_log)),
                         .groups = "drop")

performance_test <- performance %>% filter(dataset_type == "Test")

# Plot
plot_basis <- ggplot(performance_test, aes(x = k, y = MALE, color = municipality_type, lty = Model)) +
              geom_line() +
              geom_point() +
              labs(x = "Number of Spatial Basis Functions (k)",
                   y = "MALE",
                   color = "Municipality type")

saveRDS(plot_basis, "saved_files/plot_basis.rds")
print(plot_basis)
```

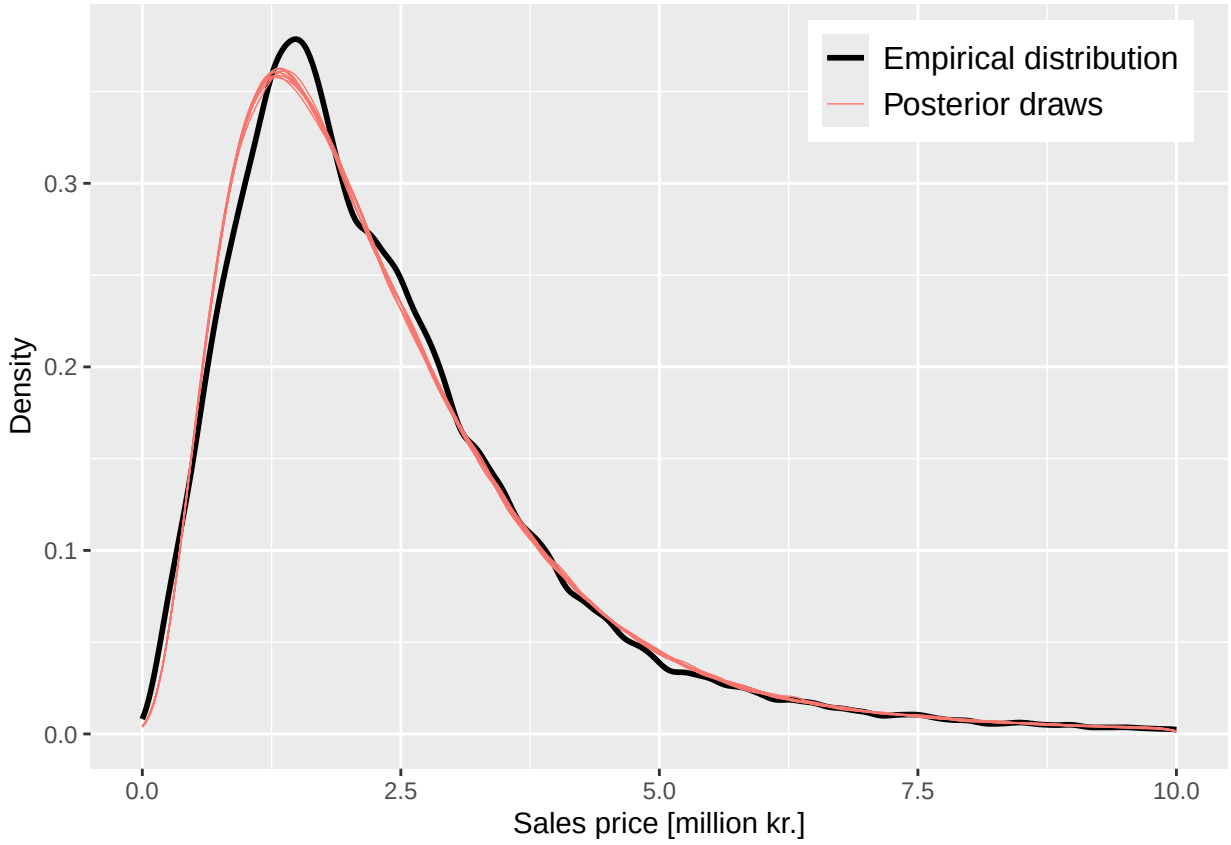


The Distributional Assumption

The code simulates a series of posterior distributions of the data based on Model1 and compares it to the actual distribution of sales prices (see https://gavinsimpson.github.io/gratia/articles/posterior-simulation.html for details on posterior simulation with gratia).

```
n_draws <- 10
samples <-  posterior_samples(models$Model1, n = n_draws) %>%
            select(.draw, .response) %>%
            mutate(draw = paste("Draw", .draw),
                   type = "Posterior draws",
                   sales_price = .response) %>%
            select(draw, type, sales_price)

samples <- samples %>%
           group_by(draw) %>%
           ungroup()

data <- as.data.frame(list(type = "Empirical distribution",
                           draw = "Empirical distribution",
                           sales_price = models$Model1$y))


plot_distribution <- ggplot(data = data,
                            aes(sales_price / 1e6, group = draw, color = type)) +
                     geom_density(linewidth = 1.0, key_glyph = "path") +
```

```
                    geom_density(data = samples,
                                 aes(sales_price / 1e6, color = type),
                                     linewidth = 0.25,
                                     key_glyph = "path") +
                    xlim(0, 10) +
                    xlab('Sales price [million kr.]') +
                    ylab('Density') +
                    theme(legend.position = c(0.8, 0.9),
                          legend.text = element_text(size = 12)) +
                    scale_color_manual(name = NULL,
                                       values = c("Empirical distribution" = "black",
                                                  "Posterior draws" = "#F8766D"))

saveRDS(plot_distribution, "saved_files/plot_distribution.rds")
print(plot_distribution)
```



Concurvity analysis

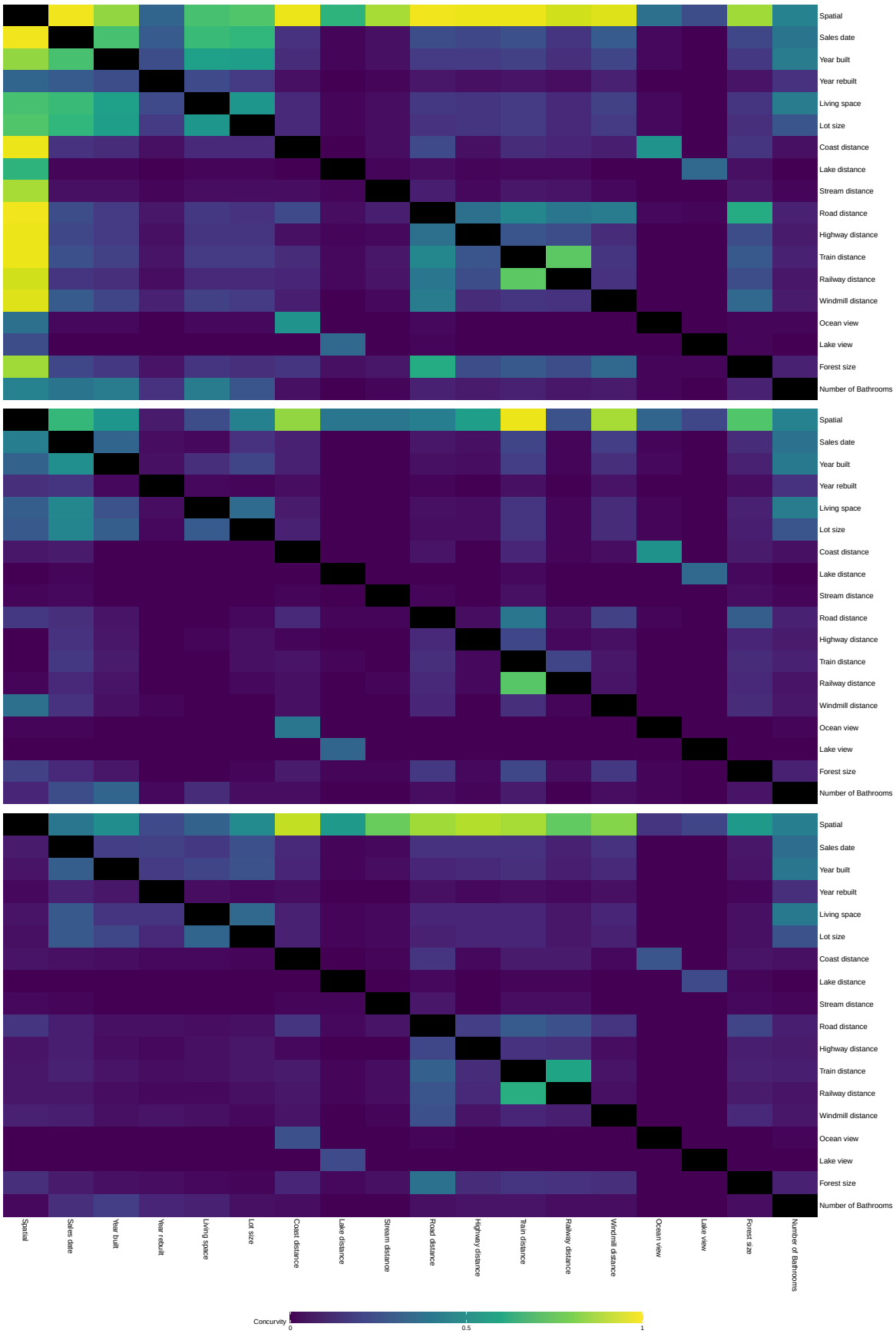
The code investigates concurvity between the various smooths for Model1 based on https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/concurvity.html.

```
cc <- concurvity(models$Model1, full = FALSE)

collapse_concurvity <- function(conc_mat, pretty_names) {
```

```r
  # remove self-concurvity first
  diag(conc_mat) <- NA

  # base smooth names (strip :municipality_typeXXXX)
  base_names <- sub(":municipality_type.*$", "", colnames(conc_mat))
  uniq_terms <- unique(base_names)

  # prepare new collapsed matrix
  conc_grouped <- matrix(
    NA_real_,
    nrow = length(uniq_terms),
    ncol = length(uniq_terms),
    dimnames = list(uniq_terms, uniq_terms)
  )

  # fill blocks
  for (i in seq_along(uniq_terms)) {
    for (j in seq_along(uniq_terms)) {
      rows <- base_names == uniq_terms[i]
      cols <- base_names == uniq_terms[j]
      block <- conc_mat[rows, cols, drop = FALSE]
      conc_grouped[i, j] <- max(block)
    }
  }

  # drop parametric part
  conc_grouped <- conc_grouped[
    rownames(conc_grouped) != "para",
    colnames(conc_grouped) != "para",
    drop = FALSE
  ]

  # apply nice labels
  rownames(conc_grouped) <- pretty_names[rownames(conc_grouped)]
  colnames(conc_grouped) <- pretty_names[colnames(conc_grouped)]

  as.data.frame(conc_grouped)
}

# 0) Your pretty names
pretty_names <- c("s(coordinate_east,coordinate_north)" = "Spatial",
                  "s(sales_date_numeric)" = "Sales date",
                  "s(living_space)" = "Living space",
                  "s(lot_size)" = "Lot size",
                  "s(number_of_bathrooms)" = "Number of Bathrooms",
                  "s(year_built)" = "Year built",
                  "s(year_rebuilt)" = "Year rebuilt",
                  "s(coast_distance)" = "Coast distance",
                  "s(lake_distance)" = "Lake distance",
                  "s(stream_distance)" = "Stream distance",
                  "s(train_station_distance)" = "Train distance",
                  "s(railway_distance)" = "Railway distance",
                  "s(road_distance)" = "Road distance",
```

```
                  "s(highway_distance)" = "Highway distance",
                  "s(windmill_distance)" = "Windmill distance",
                  "s(forest_size)" = "Forest size",
                  "s(ocean_view)" = "Ocean view",
                  "s(lake_view)" = "Lake view")

# 1) collapse concurvity
conc_grouped_worst <- collapse_concurvity(cc$worst, pretty_names)
conc_grouped_observed <- collapse_concurvity(cc$observed, pretty_names)
conc_grouped_estimate <- collapse_concurvity(cc$estimate, pretty_names)

# 2) pheatmaps WITHOUT legends
concurvity_heatmap_worst <- pheatmap(conc_grouped_worst,
                                     cluster_rows = FALSE,
                                     cluster_cols = FALSE,
                                     na_col       = "black",
                                     fontsize_row = 12,
                                     fontsize_col = 12,
                                     labels_col   = rep("", ncol(conc_grouped_worst)),
                                     color        = viridis(100),
                                     legend       = FALSE,
                                     breaks       = seq(0, 1, length.out = 100),
                                     border_color = NA,
                                     silent       = TRUE)

concurvity_heatmap_observed <- pheatmap(conc_grouped_observed,
                                        cluster_rows = FALSE,
                                        cluster_cols = FALSE,
                                        na_col       = "black",
                                        fontsize_row = 12,
                                        fontsize_col = 12,
                                        labels_col   = rep("", ncol(conc_grouped_observed)),
                                        color        = viridis(100),
                                        legend       = FALSE,
                                        breaks       = seq(0, 1, length.out = 100),
                                        border_color = NA,
                                        silent       = TRUE)

concurvity_heatmap_estimate <- pheatmap(conc_grouped_estimate,
                                        cluster_rows = FALSE,
                                        cluster_cols = FALSE,
                                        na_col       = "black",
                                        fontsize_row = 12,
                                        fontsize_col = 12,
                                        color        = viridis(100),
                                        legend       = FALSE,
                                        breaks       = seq(0, 1, length.out = 100),
                                        border_color = NA,
                                        silent       = TRUE)

# 3) stack the three heatmaps

concurvity_heatmaps <- plot_grid(concurvity_heatmap_worst$gtable,
```

```
                                concurvity_heatmap_observed$gtable,
                                concurvity_heatmap_estimate$gtable,
                                ncol = 1,
                                align = "v",
                                rel_heights = c(1, 1, 1.2))


legend_df <- data.frame(x = 1, y = 1, z = 0.5)

legend_plot <- ggplot(legend_df, aes(x = x, y = y, fill = z)) +
               geom_tile() +
               scale_fill_viridis_c(limits = c(0, 1),
                                    breaks = c(0, 0.5, 1),
                                    labels = c("0", "0.5", "1"),
                                    name   = "Concurvity") +
               theme_void() +
               theme(legend.position   = "bottom",
                     legend.title      = element_text(size = 11),
                     legend.text       = element_text(size = 10),
                     legend.key.width  = unit(4.0, "cm"),
                     legend.key.height = unit(0.6, "cm"),
                     legend.margin = margin(t = -20, b = -20))

g <- ggplotGrob(legend_plot)
guide_ix  <- which(sapply(g$grobs, function(x) x$name) == "guide-box")
legend_grob <- g$grobs[[guide_ix]]

concurvity_heatmaps <- plot_grid(concurvity_heatmaps,
                                legend_grob,
                                ncol = 1,
                                rel_heights = c(1, 0.04))


saveRDS(concurvity_heatmaps, "saved_files/concurvity_heatmaps.rds")
print(concurvity_heatmaps)
```


Spatial
Sales date
Year built
Year rebuilt
Living space
Lot size
Coast distance
Lake distance
Stream distance
Road distance
Highway distance
Train distance
Railway distance
Windmill distance
Ocean view
Lake view
Forest size
Number of Bathrooms
Concurvity
0
0.5
1

Limitations of the framework

1. Market noise

The code below estimates the market noise based on properties sold twice and compares this to the error obtained using our GAM model as well as an XGboost model.

```
### Fit time evolution and index prices

if (load){

  time_model <- readRDS("saved_files/time_model.rds")

} else{

  time_model <- gam(sales_price ~
                      municipality_type +
                      s(sales_date_numeric, k = k_other, by = municipality_type, pc = 2024),
                                  data = dataset,
                                  family = tw(link=log),
                                  method = "REML")

  saveRDS(time_model, "saved_files/time_model.rds")

}


time_model_terms <- as.data.frame(predict.gam(time_model, dataset, type = "terms"))
time_adjustment <- as.vector(rowSums(time_model_terms[, grepl("sales_date_numeric",
                                     colnames(time_model_terms)),
                                     drop = FALSE]))

dataset <- dataset %>% mutate(log_sales_price = log(sales_price),
                              log_sales_price_indexed = log_sales_price + time_adjustment,
                              sales_price_indexed = exp(log_sales_price_indexed))



### Identify double sales and discard those with changes to interior between sales

dataset <- dataset %>%
           group_by(id) %>%
           mutate(n = n()) %>%
           ungroup() %>%
           arrange(id)

other_sales <- dataset %>% filter(n != 2)
double_sales <- dataset %>% filter(n == 2)

double_sales <- double_sales %>%
                group_by(id) %>%
                filter(year_rebuilt < 2014,
                       n_distinct(living_space) == 1,
                       n_distinct(lot_size) == 1,
```

```
                        n_distinct(year_built) == 1,
                        n_distinct(house_type) == 1,
                        n_distinct(number_of_floors) == 1,
                        n_distinct(number_of_rooms) == 1,
                        n_distinct(number_of_bathrooms) == 1,
                        n_distinct(roof_type) == 1,
                        n_distinct(heating_type) == 1,
                        n_distinct(wall_type) == 1) %>%
                ungroup()



### Fit GAM-model and evaluate performance on double sales

if (load){

  gam_model <- readRDS("saved_files/gam_model.rds")

} else{

  gam_model <- mgcv::bam(sales_price_indexed ~
                         s(coordinate_east, coordinate_north, k = k_spatial, pc = c(697000, 6150000)) +
                         s(year_built, k = k_other, by = municipality_type, pc = 1970) +
                         s(year_rebuilt, k = k_other, by = municipality_type, pc = 1970) +
                         s(living_space, k = k_other, by = municipality_type, pc = 140) +
                         s(lot_size, k = k_other, by = municipality_type, pc = 800) +
                         s(coast_distance, k = k_other, pc = 1500) +
                         s(lake_distance, k = k_other, pc = 500) +
                         s(stream_distance, k = k_other, pc = 1000) +
                         s(road_distance, k = k_other, pc = 2000) +
                         s(highway_distance, k = k_other, pc = 2000) +
                         s(train_station_distance, k = k_other, pc = 2000) +
                         s(railway_distance, k = k_other, pc = 1000) +
                         s(windmill_distance, k = k_other, pc = 3000) +
                         s(ocean_view, k = k_other, pc = 0) +
                         s(lake_view, k = k_other, pc = 0) +
                         s(forest_size, k = k_other, pc = 0) +
                         s(number_of_bathrooms, k = 2, by = municipality_type, pc = 1) +
                         house_type*municipality_type +
                         roof_type*municipality_type +
                         wall_type*municipality_type +
                         heating_type*municipality_type,
                         data = other_sales,
                         family = tw(link=log),
                         method = "fREML")


  saveRDS(gam_model, "saved_files/gam_model.rds")
}

double_sales <- double_sales %>%
                mutate(residuals_gam =
                        predict(gam_model, double_sales, type = "link") - log_sales_price_indexed)
```

```r
### Fit XGboost Model and evaluate performance

dummy_model <- caret::dummyVars(~ roof_type +
                                  wall_type +
                                  heating_type +
                                  house_type +
                                  municipality_type,
                                data = other_sales)

categorical_encoded <- predict(dummy_model, newdata = other_sales)

X <- other_sales %>%
     select(coordinate_east,
            coordinate_north,
            number_of_bathrooms,
            year_built,
            year_rebuilt,
            living_space,
            lot_size,
            coast_distance,
            lake_distance,
            stream_distance,
            road_distance,
            highway_distance,
            train_station_distance,
            railway_distance,
            windmill_distance,
            forest_size,
            ocean_view,
            lake_view) %>%
      as.matrix()

X_full <- cbind(X, categorical_encoded)

if (load){

  xgb_model <- readRDS("saved_files/xgb_model.rds")

} else{

  dtrain <- xgb.DMatrix(data = X_full, label = other_sales$log_sales_price_indexed)

  xgb_model <- xgboost(
    data = dtrain,
    objective = "reg:squarederror",
    nrounds = 1000,
    max_depth = 12,
    eta = 0.1,
    subsample = 0.8,
    colsample_bytree = 0.8,
```

```
    verbose = 1
  )

  saveRDS(xgb_model, "saved_files/xgb_model.rds")

}


categorical_encoded <- predict(dummy_model, newdata = double_sales)

X <- double_sales %>%
     select(coordinate_east,
            coordinate_north,
            number_of_bathrooms,
            year_built,
            year_rebuilt,
            living_space,
            lot_size,
            coast_distance,
            lake_distance,
            stream_distance,
            road_distance,
            highway_distance,
            train_station_distance,
            railway_distance,
            windmill_distance,
            forest_size,
            ocean_view,
            lake_view) %>%
     as.matrix()

X_full <- cbind(X, categorical_encoded)

double_sales <- double_sales %>%
                mutate(residuals_xgb = predict(xgb_model, X_full) - log_sales_price_indexed)



double_sales_lagged <- double_sales %>%
                       arrange(id, sales_date) %>%
                       group_by(id) %>%
                       mutate(sales_price_diff_indexed_log = log_sales_price_indexed - lag(log_sales_pri
                              avg_price = (sales_price_indexed + lag(sales_price_indexed)) / 2,
                              sales_date_diff = sales_date_numeric - lag(sales_date_numeric)) %>%
                       ungroup()


### Noise model

if (load){

  noise_model <- readRDS("saved_files/noise_model.rds")
```

```
} else{

  noise_model <- gam(sales_price_diff_indexed_log ~ s(avg_price),
                     data = double_sales_lagged,
                     family = gaussian())


  saveRDS(noise_model, "saved_files/noise_model.rds")

}



double_sales_lagged$residuals_noise <- NA
double_sales_lagged$residuals_noise[which(!is.na(double_sales_lagged$sales_date_diff))] <- residuals(noi



### Make scatter plot with errors
smooth_data <- bind_rows(double_sales_lagged %>%
                           transmute(x = avg_price / 1e6, y = abs(residuals_noise), model = "Market noise
                           double_sales %>%
                           transmute(x = sales_price / 1e6, y = abs(residuals_gam), model = "GAM"),
                           double_sales %>%
                           transmute(x = sales_price / 1e6, y = abs(residuals_xgb), model = "XGBoost"))

plot_market_noise <- ggplot(data = double_sales_lagged,
                     aes(x = avg_price / 1e6, y = residuals_noise, color = municipality_type)) +
                     geom_point(size = 0.3) +
                     geom_smooth(data = smooth_data,
                                 aes(x = x, y = y, linetype = model),
                                 method = "gam", formula = y ~ s(x),
                                 se = TRUE,
                                 color = "black",
                                 linewidth = 1) +
                     labs(x = "Average Price [million kr.]",
                          y = "Log residuals",
                          color = "Municipality type",
                          linetype = NULL) +
                     ylim(-1, 1) +
                     xlim(0, 10) +
                     scale_linetype_manual(values = c("Market noise" = "solid",
                                                      "GAM" = "dashed",
                                                      "XGBoost" = "dotted")) +
                     theme_minimal() +
                     theme(text = element_text(size = 12),
                           legend.position = c(0.85, 0.65),
                           legend.title = element_text(size = 12),
                           legend.text = element_text(size = 10))

saveRDS(plot_market_noise, "saved_files/plot_market_noise.rds")
print(plot_market_noise)
```

2. Data scarcity

The code below fits a list of models to increasing fractions of the training set and assesses their validation MALE. Learning curves for each municipality type are then plotted.

```
fractions <- c(0.1, 0.2, 0.5, 1.0)
folds <- 1:N
model_grid <- expand.grid(fold = folds, fraction = fractions)

if (load) {
  models_fraction <- readRDS("saved_files/models_fraction.rds")
} else {

  models_fraction <- future_lapply(1:nrow(model_grid), function(idx) {
    row <- model_grid[idx, ]
    fit_fold(i = row$fold, k_spatial = k_spatial, fraction = row$fraction)
  })

  names(models_fraction) <- paste0("Model", model_grid$fold, "_", k_spatial, "_", model_grid$fraction)
  saveRDS(models_fraction, "saved_files/models_fraction.rds")
}

residual_df <- get_residuals(models_fraction, dataset)

residual_df$frac <- as.numeric(sub(".*_", "", residual_df$Model))
residual_df$Model <- sub("(_.*)$", "", residual_df$Model)




performance <- residual_df %>%
               mutate(dataset_type = case_when(test ~ "Test",
                      validation ~ "Validation",
                      TRUE ~ "Training")) %>%
               group_by(dataset_type, municipality_type, Model, frac) %>%
               summarise(MALE = mean(abs(residual_sales_price_log)),
                         .groups = "drop")


performance_test <- performance %>% filter(dataset_type == "Test")

# Plot
plot_learning_curves <- ggplot(performance_test, aes(x = frac, y = MALE, color = municipality_type, lty
                        geom_line(linewidth = 1) +
                        geom_point() +
                        labs(color = "Municipality type",
                             x = "Fraction of training set",
                             y = "MALE") +
                        theme_minimal() +
                        theme(text = element_text(size = 10),
                              legend.position = c(0.85, 0.65),
                              legend.title = element_text(size = 12),
                              legend.text = element_text(size = 10))
```

```r
saveRDS(plot_learning_curves, "saved_files/plot_learning_curves.rds")
print(plot_learning_curves)
```

3. Interplay between market noise and data scarcity

The code below examines the interplay between and noise and data scarcity by examining the validation MALE in the 2D space of KNN-averaged sales densities and sales prices.

```r
# Parameters
K <- 100 #number of neighbors to average over
N <- 3 #number of clusters in each direction
log_scale <- TRUE #perform partitioning on log scale

# Split data
test_set <- dataset %>% filter(test_set)
training_set <- dataset %>% filter(validation_set != 1, !test_set)

# Predict log price
test_set$sales_price_predicted_log <- as.vector(
  predict.gam(models$Model1, test_set, type = "link")
)

# Find spatial neighbors
neighbors <- knn(
  data = training_set[, c("coordinate_east", "coordinate_north")],
  query = test_set[, c("coordinate_east", "coordinate_north")],
  k = K,
  radius = 0
)

distance_matrix <- neighbors$nn.dists
id_matrix <- neighbors$nn.idx

# Compute local sales densities
test_set$sales_density <- apply(distance_matrix, 1, function(dists) {
  K / (pi * (0.001 * max(dists))^2)
})

# Compute average sales price of neighbors
test_set$avg_sales_price <- vapply(
  1:nrow(test_set),
  function(j) mean(training_set$sales_price[id_matrix[j, ]]),
  numeric(1)
)

# Compute residuals in log space
test_set$resid_sales_price_log <- test_set$sales_price_predicted_log - log(test_set$sales_price)

#Find clusters
price_vec <- if (log_scale) log1p(test_set$avg_sales_price) else test_set$avg_sales_price
density_vec <- if (log_scale) log1p(test_set$sales_density) else test_set$sales_density
price_clusters <- kmeans(price_vec, centers = N)$cluster
```

```r
density_clusters <- kmeans(density_vec, centers = N)$cluster
price_order <- rank(tapply(test_set$avg_sales_price, price_clusters, mean, na.rm = TRUE))
density_order <- rank(tapply(test_set$sales_density, density_clusters, mean, na.rm = TRUE))

test_set <- test_set %>%
            mutate(price_cluster = price_clusters,
                   density_cluster = density_clusters,
                   price_group = factor(price_order[price_cluster],
                   labels = c("Low Price", "Medium Price", "High Price")),
                   density_group = factor(density_order[density_cluster],
                   labels = c("Low Density", "Medium Density", "High Density")),
                   price_density_label = paste(price_group, density_group, sep = ", "))

plot_clusters <- ggplot(test_set, aes(x = sales_density,
                                      y = avg_sales_price,
                                      color = price_density_label,
                                      shape = municipality_type)) +
                 geom_point(alpha = 0.6, size = 1.5) +
                 scale_color_brewer(palette = "Set1") +
                 scale_x_continuous(trans = "log1p") +
                 scale_y_continuous(trans = "log1p") +
                 scale_shape_discrete(name = "Municipality type") +
                 labs(x = expression("Sales Density [sales per km"^2*"]"),
                      y = "Average Sales Price [kr.]",
                      color = "Cluster") +
                 theme_minimal()

saveRDS(plot_clusters, "saved_files/plot_clusters.rds")
print(plot_clusters)


#Plot clusters geographically
plot_clusters_geographically <- ggplot() +
                                # Add land/ocean contour
                                geom_contour(data = coordinate_grid,
                                             aes(x = coordinate_east,
                                                 y = coordinate_north,
                                                 z = as.numeric(!ocean)),
                                             color = "black", size = 0.4) +
                                # Add price-density group points
                                geom_point(data = test_set,
                                           aes(x = coordinate_east,
                                               y = coordinate_north,
                                               color = price_density_label),
                                           size = 0.5, alpha = 0.7) +
                                scale_color_brewer(palette = "Set1") +
                                coord_fixed() +
                                theme_void(base_size = 10) +
                                theme(legend.position = "none",
                                      plot.background = element_rect(fill = "white", color = NA),
                                      panel.background = element_rect(fill = "white", color = NA))

saveRDS(plot_clusters_geographically, "saved_files/plot_clusters_geographically.rds")
```

```r
print(plot_clusters_geographically)



#Examine interplay between model performance and clusters
binned_data <- test_set %>%
               filter(!is.na(price_group), !is.na(density_group)) %>%
               group_by(price_group, density_group) %>%
               summarise(count = n(),
                         mean_abs_resid = mean(abs(resid_sales_price_log), na.rm = TRUE),
                         .groups = "drop")

plot_interplay <- ggplot(binned_data, aes(x = density_group,
                                          y = price_group,
                                          fill = mean_abs_resid)) +
                  geom_tile(color = "white") +
                  geom_text(aes(label = count), color = "black", size = 3) +
                  scale_fill_viridis_c(name = "MALE") +
                  labs(x = NULL,
                       y = NULL) +
                  theme_minimal()

saveRDS(plot_interplay, "saved_files/plot_interplay.rds")
print(plot_interplay)
```