Stable FH Estimators over T time periods: The Multivariate Fay Herriot Modelling Approach

This section describes procedures that yield stable small area estimators for each of D areas over T subsequent time instants. Area populations, the samples and the data might change between time periods. Accordingly, we denote U_t the overall population at time t, which is partitioned into D areas $U_{1t}, ..., U_{Dt}$, of respective population sizes N_{1t}

An overview of the MFH model estimation process is as follows:

- Step 1: Compute the selected direct area estimators for each target area $d = 1, ..., D$ for each time $t = 1, ..., T$ and estimators of their corresponding sampling variances and covariances.
- Step 2: Prepare variables (from household survey, administrative data or other sources) at the level of the target area for each time instant in the MFH model. We present a simple approach which performs model selection in a pooled linear regression model without time effects.
- Step 3: Fit the MFH models to test for homoskedastic area-time effects $(\mu_{d_1}, ..., \mu_{d_T})$ are homoskedastic or not. If we reject the homoskedasticity of variances, implement the MFH3 model. Otherwise, we proceed with the MFH2 model.
- Step 4: Check the selected model assumptions, including linearity, normality of predicted area effects and standardized model residuals, and the presence of the outlying areas.
- Step 5: In case of systematic model departures such as isolated departures because of outlying areas, some adjustments might need to be implemented before returning to Step 2 to recompute the MFH model.
- Step 6: If model assumptions hold, using the above direct estimates and estimated sampling variances and covariances, and the selected auxiliary variables, compute MFH estimators $\hat{\delta}_{dt}^{MFH}, \quad d = 1, ..., D$ and $t = 1, ..., T$ and their corresponding estimated MSEs.

We will show below the use of the `eblupMFH2()` and `eblupMFH3()` from the R package msae [@permatasari2022package] compute the EBLUPs and their MSE estimates under the MFH models 2 and 3, respectively. The calls to these functions are:

```
eblupMFH2(formula, vardir, MAXITER = 100, PRECISION = 1e-04, data)
```

```
eblupMFH3(formula, vardir, MAXITER = 100, PRECISION = 1e-04, data)
```

MFH Estimation of Poverty Rates for T time periods

In this example, we use a synthetic data set adapted from R package sae called `incomedata`. The original data contains information for $n = 17,119$ fictitious individuals residing across $D = 52$ Spanish provinces. The variables include the name of the province of residence (`provlab`), province code (`prov`), as well as several correlates of income. We have added two additional income vectors corresponding to two additional years of data.

We will show how to estimate a poverty map for each year by using the Multivariate Fay Herriot modelling approach. This approach allows us to take advantage of the temporal correlation between poverty rates i.e. an individuals income in year t is likely correlated with their income in year $t + 1$.

The rest of this tutorial shows how to prepare MFH models using a random 10% sample of the `incomedata` to estimate the poverty rates.

```
if (sum(installed.packages()[,1] %in% "pacman") != 1){

  install.packages("pacman")

}

pacman::p_load(sf, data.table, tidyverse, car, msae,
               sae, survey, spdep, knitr, MASS, caret)

income_dt <- readRDS("data/incomedata_sample.RDS")

glimpse(income_dt)
```

```
Rows: 1,716
Columns: 31
Groups: provlab [52]
$ provlab     <fct> Alava, Alava, Alava, Alava, Alava, Alava, Alava, Alava, Al~
$ prov        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2~
$ ac          <int> 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 8, 8, 8, 8, 8, 8, ~
$ gen         <int> 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2~
$ age         <int> 4, 4, 5, 3, 3, 2, 5, 4, 4, 0, 0, 3, 3, 1, 2, 0, 3, 3, 0, 2~
$ nat         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ educ        <int> 3, 3, 1, 3, 3, 0, 1, 3, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0~
$ labor       <int> 1, 1, 3, 1, 3, 0, 3, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0~
$ age2        <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1~
$ age3        <int> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0~
$ age4        <int> 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
$ age5        <int> 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ educ1       <int> 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0~
$ educ2       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
$ educ3       <int> 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ nat1        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ labor1      <int> 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0~
$ labor2      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ labor3      <int> 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ income2012  <dbl> 26125.271, 27404.099, 13117.838, 27920.737, 8705.592, 6340~
$ weight      <dbl> 7424.759, 23577.245, 41976.969, 19471.566, 12738.217, 2463~
$ income2013  <dbl> 27500.673, 28174.985, 13581.315, 28508.978, 9182.841, 6272~
$ income2014  <dbl> 34739.7326, 32137.8461, 15980.5806, 31512.1529, 11699.9025~
$ povline2012 <dbl> 6477.484, 6477.484, 6477.484, 6477.484, 6477.484, 6477.484~
$ povline2013 <dbl> 6515.865, 6515.865, 6515.865, 6515.865, 6515.865, 6515.865~
$ povline2014 <dbl> 6515.865, 6515.865, 6515.865, 6515.865, 6515.865, 6515.865~
$ abs         <dbl> 0.8493392, 0.8493392, 0.8493392, 0.8493392, 0.8493392, 0.8~
$ ntl         <dbl> 0.2091905, 0.2091905, 0.2091905, 0.2091905, 0.2091905, 0.2~
$ aec         <dbl> 1.1651876, 1.1651876, 1.1651876, 1.1651876, 1.1651876, 1.1~
$ schyrs      <dbl> 1.7099321, 1.7099321, 1.7099321, 1.7099321, 1.7099321, 1.7~
$ mkt         <dbl> 2.578254, 2.578254, 2.578254, 2.578254, 2.578254, 2.578254~
```

Step 1: Direct Estimation

We will use the direct HT estimators that use the survey weights in `weight` variable. First, we calculate the total sample size, the number of provinces, the sample sizes for each province and extract the population sizes for each province/target area from the `sizeprov` file. For those using the household/individual level survey data, this may be obtained from the sum of the household or individual weights as appropriate.

```
data(sizeprov)

## quickly compute sample size for each province
sampsize_dt <-
income_dt |>
  group_by(prov) |>
  summarize(N = n())

## the poverty line for each is already included within the data.
## Lets compute the direct estimates for each year of data and create a list of data.frames
## containing the direct estimate, the standard errors and the coefficient of variation.

direct_list <-
```

```r
  mapply(FUN = function(y, threshold){

     z <- emdi::direct(y = y,
                       smp_data = income_dt %>% as.data.table(),
                       smp_domains = "prov",
                       weights = "weight",
                       threshold = unique(income_dt[[threshold]]),
                       var = TRUE)

     z <-
       z$ind |>
       dplyr::select(Domain, Head_Count) |>
       rename(Direct = "Head_Count") |>
       merge(z$MSE |>
               dplyr::select(Domain, Head_Count) |>
               rename(MSE = "Head_Count"),
             by = "Domain") |>
       mutate(SD = sqrt(MSE)) |>
       mutate(CV = SD / Direct) |>
       merge(sampsize_dt |>
               mutate(prov = as.factor(prov)),
             by.x = "Domain",
             by.y = "prov")


     return(z)

  }, SIMPLIFY = FALSE,
  y = c("income2012", "income2013", "income2014"),
  threshold = c("povline2012", "povline2013", "povline2014"))
```

```
Registered S3 method overwritten by 'openxlsx':
  method               from
  as.character.formula formula.tools
```

```
Warning in emdi::direct(y = y, smp_data = income_dt %>% as.data.table(), : For
the following domains at least one bootstrap failed, this may be due to a very
small sample size. For these domains the variance estimation is based on a
reduced number of bootstrap iteration.  To see the number of successful
bootstrap iterations for each domain and indicator, have a look at the value
successful_bootstraps in the returned object.
```

```
Warning in emdi::direct(y = y, smp_data = income_dt %>% as.data.table(), : For
the following domains at least one bootstrap failed, this may be due to a very
small sample size. For these domains the variance estimation is based on a
reduced number of bootstrap iteration.  To see the number of successful
bootstrap iterations for each domain and indicator, have a look at the value
successful_bootstraps in the returned object.
Warning in emdi::direct(y = y, smp_data = income_dt %>% as.data.table(), : For
the following domains at least one bootstrap failed, this may be due to a very
small sample size. For these domains the variance estimation is based on a
reduced number of bootstrap iteration.  To see the number of successful
bootstrap iterations for each domain and indicator, have a look at the value
successful_bootstraps in the returned object.
```

```r
direct_list %>%
  lapply(X = .,
         FUN = function(x){

           y <- head(x)

           y %>% kable()

         })
```

$income2012

Domain	Direct	MSE	SD	CV	N
1	0.3088580	0.0254863	0.1596442	0.5168853	10
10	0.2774950	0.0061535	0.0784441	0.2826864	28
11	0.2130526	0.0050686	0.0711944	0.3341634	40
12	0.1932215	0.0368154	0.1918734	0.9930232	12
13	0.2259104	0.0052792	0.0726584	0.3216247	25
14	0.3341656	0.0137867	0.1174167	0.3513728	22

$income2013

Domain	Direct	MSE	SD	CV	N
1	0.3088580	0.0254863	0.1596442	0.5168853	10
10	0.2425857	0.0074776	0.0864729	0.3564634	28

```
|11      | 0.1983854| 0.0038739| 0.0622405| 0.3137353| 40|
|12      | 0.1932215| 0.0368154| 0.1918734| 0.9930232| 12|
|13      | 0.1672609| 0.0045837| 0.0677029| 0.4047743| 25|
|14      | 0.3341656| 0.0137867| 0.1174167| 0.3513728| 22|

$income2014


|Domain |    Direct|       MSE|        SD|        CV|  N|
|:------|---------:|---------:|---------:|---------:|--:|
|1       | 0.3088580| 0.0254863| 0.1596442| 0.5168853| 10|
|10      | 0.3035343| 0.0109776| 0.1047742| 0.3451807| 28|
|11      | 0.3277464| 0.0055597| 0.0745637| 0.2275042| 40|
|12      | 0.2636266| 0.0306991| 0.1752116| 0.6646202| 12|
|13      | 0.1951704| 0.0044149| 0.0664445| 0.3404436| 25|
|14      | 0.3901024| 0.0126833| 0.1126202| 0.2886938| 22|
```

Estimating the Variance-Covariance Matrix of the Sampling Error Distribution

Next, we quickly estimate the sample variance and covariance for the direct estimator using the survey R package as follows:

```r
## quickly creating the poverty indicator
income_dt <-
  income_dt |>
  mutate(poor2012 = ifelse(income2012 < povline2012, 1, 0),
         poor2013 = ifelse(income2013 < povline2013, 1, 0),
         poor2014 = ifelse(income2014 < povline2014, 1, 0))


### creating a survey object
design_obj <- svydesign(ids = ~1, ###replace this argument with the PSU, cluster or enumerat
                        weights = ~weight, ### survey weights
                        data = income_dt)

var_dt <- svymean(~poor2012 + poor2013 + poor2014, design_obj)

var_dt <- vcov(var_dt)

var_dt <- as.numeric(c(diag(var_dt), var_dt[lower.tri(var_dt, diag = FALSE)]))

names(var_dt) <- c("v1", "v2", "v3", "v12", "v13", "v23")
```

```
### here is what the result should look like

var_dt
```

```
          v1           v2           v3           v12          v13          v23
1.497531e-04 1.520211e-04 1.935070e-04 1.368008e-04 8.529235e-05 1.000923e-04
```

Handling Low Sample Size: Variance Smoothing

A quick inspection of the preceding results will show some provinces contain low sample sizes which sometimes result in extreme value poverty rates and hence 0 variance. To avoid this, we will show you how to apply the variance smoothing method suggested by [@you2023application]. Please see the code and Roxygen comments below explaining the use of the `varsmoothie_king()` function which computes smoothed variances.

```
#' A function to perform variance smoothing
#'
#' The variance smoothing function applies the methodology of (Hiridoglou and You, 2023)
#' which uses simply log linear regression to estimate direct variances for sample
#' poverty rates which is useful for replacing poverty rates in areas with low sampling
#'
#' @param domain a vector of unique domain/target areas
#' @param direct_var the raw variances estimated from sample data e=
#' @param sampsize the sample size for each domain
#' @param y indicator variable at the survey sample level for each household/individual/unit
#'
#' @export

varsmoothie_king <- function(domain,
                             direct_var,
                             sampsize,
                             y){

  dt <- data.table(Domain = domain,
                   var = direct_var,
                   n = sampsize)

  dt$log_n <- log(dt$n)
  dt$log_var <- log(dt$var)

  lm_model <- lm(formula = log_var ~ log_n,
```

```
                  data = dt[!(abs(dt$log_var) == Inf),])

  dt$pred_var <- predict(lm_model, newdata = dt)

  dt$var_smooth <- exp(dt$pred_var) * exp(var(y, na.rm = TRUE)/2)

  return(dt[, c("Domain", "var_smooth"), with = F])

}
```

OK, the goal now is to use the above `varsmoothie_king()` function to add an additional column of smoothed variances into our `direct_list` object.

```
direct_list <-
direct_list %>%
  mapply(x = .,
         y = list("poor2012", "poor2013", "poor2014"),
         FUN = function(x, y){

           z <- varsmoothie_king(domain = x[["Domain"]],
                                 direct_var = x[["MSE"]],
                                 sampsize = x[["N"]],
                                 y = income_dt[[y]])

           z <- x %>% merge(z, by = "Domain")

           return(z)

         },
         SIMPLIFY = FALSE)
```

- Now, you can replace the zero/near zero sample size area MSEs with their smoothed variances.

Step 2: Variable Preparation and Model Selection

Variable Preparation

We have now shown how to compute direct estimates of poverty. In addition, we compute the variances for the direct estimates. This will allow us to compare the value of small area estimation with the multivariate Fay Herriot Model. The large direct variances point to the imprecision in estimating poverty estimates from the survey. MFH modelling should

improve model precision particularly in the areas with very little sampling. With the direct estimation above, we are able to compare the statistical gains/improvements from performing MFH modelling.

Now, let us begin by preparing the set of variables for MFH estimation. First, we will create more variable interactions. We have developed a function `create_interactions()`. See documentation with the Roxygen comments in the following chunk:

```
#' Create Interaction Terms Between a Variable and a List of Other Variables
#'
#' This function generates interaction terms between a specified variable (`interacter_var`)
#' and a list of other variables (`var_list`) in a given dataset. The resulting interaction
#' terms are returned in a new data frame.
#'
#' @param dt A `data.frame` or `data.table` containing the data.
#' @param interacter_var A string specifying the name of the variable to interact with each v
#' @param var_list A character vector of variable names in `dt` to be interacted with `intera
#'
#' @return A `data.frame` containing the interaction terms. Each column is named using the pa
#'
#' @examples
#' \dontrun{
#' dt <- data.frame(a = 1:5, b = 6:10, c = 11:15)
#' create_interactions(dt, interacter_var = "a", var_list = c("b", "c"))
#' }
#'
#' @export


create_interactions <- function(dt, interacter_var, var_list) {
  # Ensure dt is a data.table
  if (!"data.frame" %in% class(dt)) {
    dt <- as.data.table(dt)
  }

  # Check if interacter_var exists in the dataset
  if (!(interacter_var %in% names(dt))) {
    stop(paste(interacter_var, "not found in dataset"))
  }

  # Check if var_list contains valid variables that exist in the dataset
  if (any(!var_list %in% names(dt))) {
    stop("Some variables in var_list are not found in the dataset.")
```

```
  }

  # Create an empty data.table to store interactions
  int_dt <- data.frame(matrix(nrow = nrow(dt)))

  # Loop over var_list to create interaction terms
  for (var in var_list) {
    interaction_name <- paste0(var, "_X_", interacter_var)
    int_dt[[interaction_name]] <- dt[[var]] * dt[[interacter_var]]
  }

  int_dt <- int_dt[, -1]

  return(int_dt)
}

### applying the create_interactions() functions
income_dt <-
  income_dt %>%
  cbind(create_interactions(dt = .,
                            interacter_var = "gen",
                            var_list = c("age2", "age3", "age4", "age5",
                                         "educ1", "educ2", "educ3", "nat1",
                                         "labor1", "labor2", "labor3"))) %>%
  cbind(create_interactions(dt = .,
                            interacter_var = "educ3",
                            var_list = c("age2", "age3", "age4", "age5",
                                         "nat1", "labor1", "labor2", "labor3")))
```

More (and probably better correlates of income and poverty) can be created to improve the predictive power of the model. The next step is to take target level variables. The MFH model is a model of poverty rates at the target area level, hence the data format required for this exercise has the province as its unit of observation. This format has a few essential columns:

- (1) Variables for poverty rates for each time period
- (2) The set of candidate variables from which the most predicted of poverty rates will be selected
- (3) The target area variable identifier (i.e. in this case the province variable `prov` and `provlab`)

We prepare this dataset as follows:

```
## create the candidate variables
candidate_vars <- colnames(income_dt)[!colnames(income_dt) %in%
                                        c("provlab", "prov", "income",
                                          "weight", "povline", "y",
                                          "poverty", "y0", "y1", "y2",
                                          "p0_prov", "p1_prov", "p2_prov",
                                          "ac", "nat", "educ", "labor",
                                          "age")]

candidate_vars <- candidate_vars[!grepl("sampsize|prov|poverty|income|povline|^v[0-9]|^y[0-9]
                                        candidate_vars)]

### computing the province level data
prov_dt <-
income_dt |>
  group_by(prov, provlab) |>
  summarize(
    across(
      any_of(candidate_vars),
      ~ weighted.mean(x = ., na.rm = TRUE),
      .names = "{.col}"
    )
  )
```

```
`summarise()` has grouped output by 'prov'. You can override using the
`.groups` argument.
```

Model Selection

Next, we apply a simple variable selection process which employs the stepwise regression algorithm using the AIC selection criteria as in described by [@yamashita2007stepwise]. The function `stepAIC_wrapper()` implemented below is a wrapper to the `stepAIC()` function carries all the perfunctory cleaning necessary use the `stepAIC()` function. This includes dropping columns that are entirely missing (`NA`) and keep only complete cases/observations and remove perfectly or near collinear variables and combinations using the variance inflation method.

```
#' A function to perform stepwise variable selection with AIC selection criteria
#'
#' @param dt data.frame, dataset containing the set of outcome and independent variables
#' @param xvars character vector, the set of x variables
```

```r
#' @param y chr, the name of the y variable
#' @param vif_threshold integer, a variance inflation factor threshold
#'
#' @import data.table
#' @importFrom cars vif
#' @importFrom MASS stepAIC

stepAIC_wrapper <- function(dt, xvars, y, cor_thresh = 0.95) {

  dt <- as.data.table(dt)

  # Drop columns that are entirely NA
  dt <- dt[, which(unlist(lapply(dt, function(x) !all(is.na(x))))), with = FALSE]

  xvars <- xvars[xvars %in% colnames(dt)]

  # Keep only complete cases
  dt <- na.omit(dt[, c(y, xvars), with = FALSE])

  # Step 1: Remove aliased (perfectly collinear) variables
  model_formula <- as.formula(paste(y, "~", paste(xvars, collapse = " + ")))
  lm_model <- lm(model_formula, data = dt)
  aliased <- is.na(coef(lm_model))
  if (any(aliased)) {
    xvars <- names(aliased)[!aliased & names(aliased) != "(Intercept)"]
  }

  # Step 2: Remove near-linear combinations
  xmat <- as.matrix(dt[, ..xvars])
  combo_check <- tryCatch(findLinearCombos(xmat), error = function(e) NULL)
  if (!is.null(combo_check) && length(combo_check$remove) > 0) {
    xvars <- xvars[-combo_check$remove]
    xmat <- as.matrix(dt[, ..xvars])
  }

  # Step 3: Drop highly correlated variables
  cor_mat <- abs(cor(xmat))
  diag(cor_mat) <- 0
  while (any(cor_mat > cor_thresh, na.rm = TRUE)) {
    cor_pairs <- which(cor_mat == max(cor_mat, na.rm = TRUE), arr.ind = TRUE)[1, ]
    var1 <- colnames(cor_mat)[cor_pairs[1]]
    var2 <- colnames(cor_mat)[cor_pairs[2]]
```

```
    # Drop the variable with higher mean correlation
    drop_var <- if (mean(cor_mat[var1, ]) > mean(cor_mat[var2, ])) var1 else var2
    xvars <- setdiff(xvars, drop_var)
    xmat <- as.matrix(dt[, ..xvars])
    cor_mat <- abs(cor(xmat))
    diag(cor_mat) <- 0
  }

  # Step 4: Warn if still ill-conditioned
  cond_number <- kappa(xmat, exact = TRUE)
  if (cond_number > 1e10) {
    warning("Design matrix is ill-conditioned (condition number > 1e10). Consider reviewing v
  }

  # Final model fit
  model_formula <- as.formula(paste(y, "~", paste(xvars, collapse = " + ")))
  full_model <- lm(model_formula, data = dt)

  # Stepwise selection
  stepwise_model <- stepAIC(full_model, direction = "both", trace = 0)

  return(stepwise_model)

}
```

We apply the variable selection process for the selection of each time period model. See the application below:

```
### the set of outcome variables
outcome_list <- c("poor2012", "poor2013", "poor2014")

candidate_vars <- candidate_vars[!candidate_vars %in% outcome_list]

### applying variable selection to each out variable
stepaicmodel_list <-
  mapply(x = outcome_list,
         FUN = function(x){

           y <- stepAIC_wrapper(dt = prov_dt,
                                xvars = candidate_vars,
                                y = x,
                                cor_thresh = 0.8)
```

```
            coef_list <- y[["coefficients"]]

            coef_list <- names(coef_list)[!names(coef_list) %in% "(Intercept)"]

            return(coef_list)

          },
          SIMPLIFY = FALSE)

### now lets create a list of equations
### create the formulas

mfh_formula <-
  mapply(x = outcome_list,
          y = stepaicmodel_list,
          FUN = function(x, y){

            formula <- as.formula(paste0(x, " ~ ", paste(y, collapse = " + ")))

            return(formula)

          }, SIMPLIFY = FALSE)

mfh_formula
```

```
$poor2012
poor2012 ~ schyrs + nat1_X_gen + labor1_X_gen + age2_X_educ3 +
    age3_X_educ3 + educ2_X_gen
<environment: 0x000001f3a69169e8>

$poor2013
poor2013 ~ age5 + educ1 + nat1 + schyrs + age2_X_gen + age3_X_gen +
    age4_X_gen + educ2_X_gen + nat1_X_gen + labor1_X_gen + nat1_X_educ3 +
    labor2_X_educ3
<environment: 0x000001f3a6922780>

$poor2014
poor2014 ~ schyrs + age4_X_gen + nat1_X_gen + labor2_X_gen +
    age2_X_educ3 + age3_X_educ3
<environment: 0x000001f3a6ca6dd8>
```

Let's run some linear regressions for set of selected variables to get a sense for the predicted power of the models independently

```
lmcheck_obj <-
lapply(X = mfh_formula,
       FUN = lm,
       data = prov_dt)


lapply(X = lmcheck_obj,
       FUN = summary)
```

```
$poor2012

Call:
FUN(formula = X[[i]], data = ..1)

Residuals:
      Min        1Q    Median        3Q       Max
-0.166165 -0.043304 -0.003908  0.042823  0.203688

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.18448    0.13627   1.354   0.1826
schyrs        0.07755    0.01324   5.859 5.05e-07 ***
nat1_X_gen    0.10144    0.07550   1.344   0.1858
labor1_X_gen -0.19158    0.08123  -2.358   0.0228 *
age2_X_educ3  0.77876    0.48599   1.602   0.1161
age3_X_educ3  0.55438    0.21893   2.532   0.0149 *
educ2_X_gen  -0.10837    0.05855  -1.851   0.0708 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08245 on 45 degrees of freedom
Multiple R-squared:  0.6112,    Adjusted R-squared:  0.5594
F-statistic: 11.79 on 6 and 45 DF,  p-value: 6.658e-08


$poor2013

Call:
FUN(formula = X[[i]], data = ..1)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-0.15309 -0.04765  0.01047  0.03119  0.19595

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.41864    0.24592   1.702 0.096644 .
age5          -0.50889    0.25300  -2.011 0.051229 .
educ1          0.59853    0.25347   2.361 0.023298 *
nat1          -0.79662    0.29784  -2.675 0.010876 *
schyrs         0.06170    0.01379   4.475 6.47e-05 ***
age2_X_gen    -0.32766    0.23370  -1.402 0.168809
age3_X_gen    -0.30830    0.19561  -1.576 0.123092
age4_X_gen    -0.56869    0.18239  -3.118 0.003415 **
educ2_X_gen    0.18938    0.14821   1.278 0.208877
nat1_X_gen     0.48678    0.13185   3.692 0.000679 ***
labor1_X_gen  -0.14310    0.07837  -1.826 0.075523 .
nat1_X_educ3   0.87472    0.31484   2.778 0.008360 **
labor2_X_educ3 3.54825    2.63438   1.347 0.185787
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.07922 on 39 degrees of freedom
Multiple R-squared:  0.6385,    Adjusted R-squared:  0.5273
F-statistic: 5.741 on 12 and 39 DF,  p-value: 1.406e-05


$poor2014

Call:
FUN(formula = X[[i]], data = ..1)

Residuals:
      Min        1Q    Median        3Q       Max
-0.311810 -0.050819 -0.006537  0.051198  0.217400

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.31476    0.13542  -2.324 0.024683 *
schyrs        0.05077    0.01391   3.651 0.000678 ***
age4_X_gen   -0.20399    0.09274  -2.200 0.033007 *
nat1_X_gen    0.38566    0.09315   4.140 0.000150 ***
```

```
labor2_X_gen  0.34676    0.21861   1.586 0.119693
age2_X_educ3  1.91993    0.55725   3.445 0.001246 **
age3_X_educ3  0.90222    0.25560   3.530 0.000972 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0962 on 45 degrees of freedom
Multiple R-squared:  0.508, Adjusted R-squared:  0.4424
F-statistic: 7.745 on 6 and 45 DF,  p-value: 9.449e-06
```

Step 3: Model Estimations

Next, we show how to use the `msae` R package to estimate the Empirical Best Linear Unbiased Predictor (EBLUP) for the poverty map using the `eblupMFH2()` and `eblupMFH3()` functions which allow for time series fay herriot estimation under homoskedastic and heteroskedastic assumptions. We show appropriate tests for deciding whether random effects variance and covariance matrix exhibit homoskedastic or heteroskedastic. For completeness, we also briefly perform the previous described direct estimation in step 1, using the `eblupUFH()` function as well as the `eblupMFH1()` for the fay herriot model. This allows us to show the benefit of MFH estimation over the simple direct estimation as well as the time invariant Fay Herriot Model.

```
### first lets add the sampling variance and covariance matrix to the prov_dt dataset

prov_dt <-
prov_dt |>
  mutate(!!!as.list(var_dt))

#### now we estimate all 4 models including the multivariate fay herriot models

model0_obj <- eblupUFH(mfh_formula, vardir = names(var_dt), data = prov_dt)
model1_obj <- eblupMFH1(mfh_formula, vardir = names(var_dt), data = prov_dt)
model2_obj <- eblupMFH2(mfh_formula, vardir = names(var_dt), data = prov_dt, MAXITER = 10000)
model3_obj <- eblupMFH3(mfh_formula, vardir = names(var_dt), data = prov_dt)
```

```
Warning in sqrt(diag(Qh)): NaNs produced
```

With the MFH3 estimation, we can check if the variance-covariance matrix structure points to heteroskedastic random effects. We do so as follows:

```
model3_obj$fit$refvarTest
```

```
                  refvar    T-test   p-value
1 poor2012  vs  poor2013 -0.019587   0.98437
2 poor2012  vs  poor2014 -0.081893   0.93473
3 poor2013  vs  poor2014   -3.1243 0.0017823
```

This tests the null hypothesis that the variances σ_t^2 at each pair of time instants t and s are equal against the alternative that they are not. In this case, at significance level of 0.05, we reject the equality of variances between $t = 2013$ and $t = 2014$. Hence we can use the MFH3 model (`eblupMFH3()` function). If these tests supported the equality of variances, then we would use instead the function `eblupMFH2` for estimation.

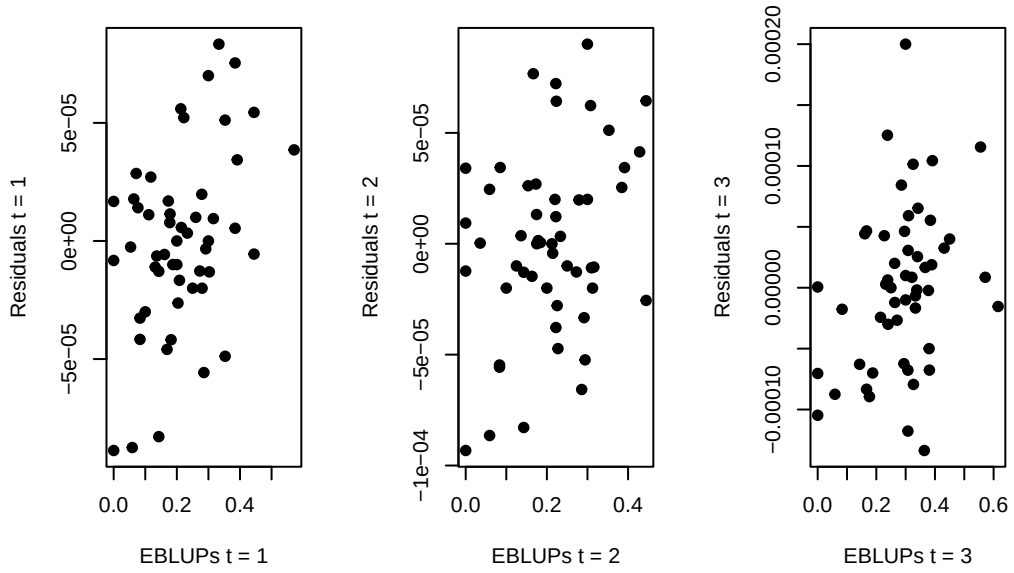
Step 4: Post Estimation Diagnostics: Model Assumption Checks for Linearity, Normality and Outliers

We now verify the assumptions of the MFH3 model. This includes assessing linearity, the normality of the predicted area effects and standardized residuals, as well as checking for the presence of outlying areas.

The Check for Linearity

We first check the linearity assumption. This can be addressed by regressing the model residuals against the predicted values (EBLUPs).

```
resids_3 <- cbind(prov_dt$poor2012 - model3_obj$eblup$poor2012,
                  prov_dt$poor2013 - model3_obj$eblup$poor2013,
                  prov_dt$poor2014 - model3_obj$eblup$poor2014)

layout(matrix(1:3, nrow = 1, byrow = TRUE))

plot(model3_obj$eblup$poor2012, resids_3[,1], pch = 19, xlab = "EBLUPs t = 1", ylab = "Residu
plot(model3_obj$eblup$poor2013, resids_3[,2], pch = 19, xlab = "EBLUPs t = 2", ylab = "Residu
plot(model3_obj$eblup$poor2014, resids_3[,3], pch = 19, xlab = "EBLUPs t = 3", ylab = "Residu
```




```r
fits_3 <- model3_obj$eblup  # EBLUPs (predicted values)

# Run regression of residuals on fitted values for each time period
lm_resid_fit_t1 <- lm(resids_3[,1] ~ model3_obj$eblup$poor2012)
lm_resid_fit_t2 <- lm(resids_3[,2] ~ model3_obj$eblup$poor2013)
lm_resid_fit_t3 <- lm(resids_3[,3] ~ model3_obj$eblup$poor2014)

# View summaries
summary(lm_resid_fit_t1)
```

```
Call:
lm(formula = resids_3[, 1] ~ model3_obj$eblup$poor2012)

Residuals:
       Min          1Q      Median          3Q         Max
-7.360e-05 -2.089e-05  4.150e-07  2.288e-05  6.858e-05

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                -2.730e-05  9.693e-06  -2.816  0.00694 **
model3_obj$eblup$poor2012  1.262e-04  3.936e-05   3.206  0.00235 **
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.491e-05 on 50 degrees of freedom
Multiple R-squared:  0.1705,    Adjusted R-squared:  0.1539
F-statistic: 10.28 on 1 and 50 DF,  p-value: 0.002349
```

```
summary(lm_resid_fit_t2)
```

```

Call:
lm(formula = resids_3[, 2] ~ model3_obj$eblup$poor2013)

Residuals:
       Min         1Q     Median         3Q        Max
-7.614e-05 -2.417e-05  3.067e-06  2.148e-05  8.086e-05

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                -2.195e-05  1.157e-05  -1.898   0.0635 .
model3_obj$eblup$poor2013  1.066e-04  4.878e-05   2.186   0.0335 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.013e-05 on 50 degrees of freedom
Multiple R-squared:  0.08722,   Adjusted R-squared:  0.06896
F-statistic: 4.777 on 1 and 50 DF,  p-value: 0.03355
```

```
summary(lm_resid_fit_t3)
```

```

Call:
lm(formula = resids_3[, 3] ~ model3_obj$eblup$poor2014)

Residuals:
       Min         1Q     Median         3Q        Max
-1.460e-04 -4.263e-05  2.724e-06  4.093e-05  1.979e-04

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                -4.592e-05  2.181e-05  -2.106   0.0403 *
```

```
model3_obj$eblup$poor2014  1.603e-04  6.949e-05   2.307   0.0253 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.393e-05 on 50 degrees of freedom
Multiple R-squared:  0.09617,   Adjusted R-squared:  0.07809
F-statistic:  5.32 on 1 and 50 DF,  p-value: 0.02526
```

Evaluating the Normality Assumption

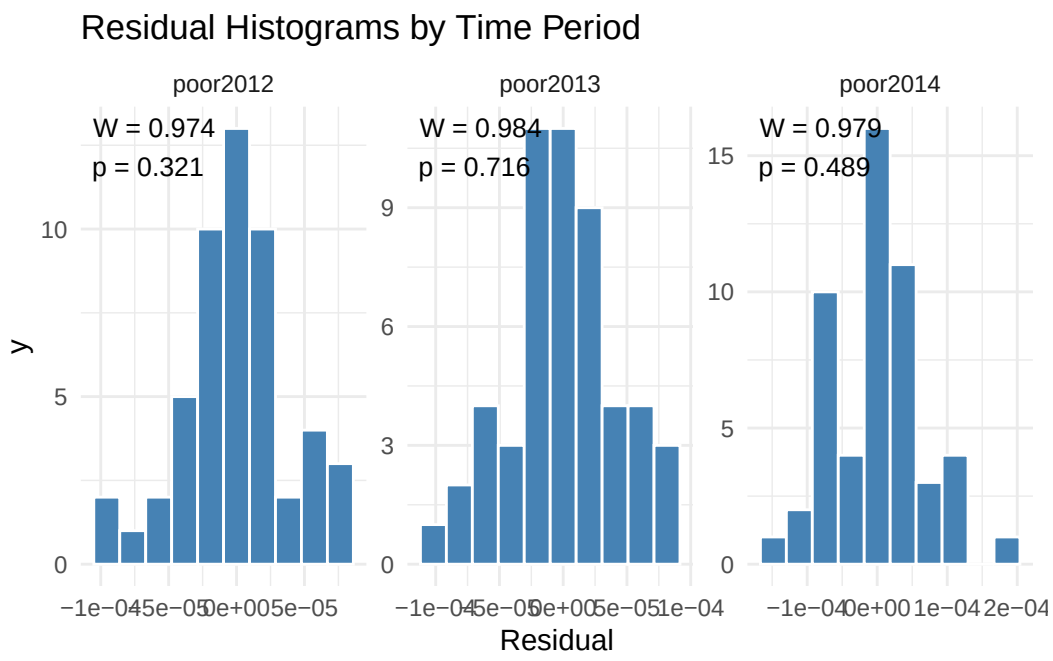
The Shapiro Wilks Test

We use the shapiro wilks test of normality using the `shapiro.test()` function in base R. The Shapiro-Wilk test assesses whether a sample of data is drawn from a normally distributed population. It does so by comparing the order statistics (i.e., sorted values) of the sample to the expected values under a normal distribution. Specifically, the test statistic W is a ratio of the squared correlation between the observed sample quantiles and the corresponding normal quantiles.

First, we perform the shapiro wilks normality test on the model errors, ε. We show both the normality distribution histogram as well as the qqplots as below:

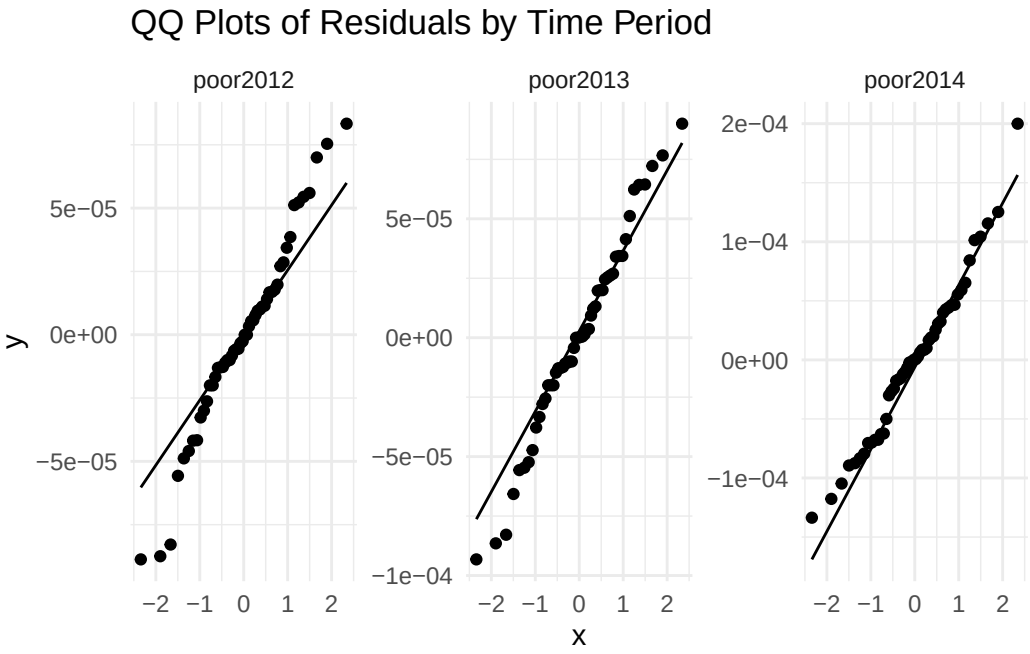
```
### evaluating the normality assumption
resid_dt <- prov_dt[,c("poor2012", "poor2013", "poor2014")] - model3_obj$eblup

### perform the shapiro test

shapiro_obj <- apply(resid_dt, 2, shapiro.test)

summary_dt <-
  data.frame(Time = names(shapiro_obj),
             W = lapply(X = shapiro_obj,
                        FUN = function(x){

                          return(x$statistic[[1]])

                        }) %>%
               as.numeric(),
             p_value = lapply(X = shapiro_obj,
                              FUN = function(x){

                                return(x$p.value)
```

```
                                  }) %>%
                as.numeric())

### plot the results
summary_dt <-
  summary_dt %>%
  mutate(label = paste0("W = ", round(W, 3), "\n", "p = ", signif(p_value, 3)))

resid_dt %>%
  pivot_longer(cols = everything(),
               names_to = "Time",
               values_to = "Residual") %>%
  ggplot(aes(x = Residual)) +
  geom_histogram(bins = 10, fill = "steelblue", color = "white") +
  geom_text(data = summary_dt, aes(x = -Inf, y = Inf, label = label),
            hjust = -0.1, vjust = 1.2, inherit.aes = FALSE, size = 3.5) +
  facet_wrap(~Time, scales = "free") +
  theme_minimal() +
  labs(title = "Residual Histograms by Time Period")
```




```
### here's how to create qqplots
resid_dt %>%
```

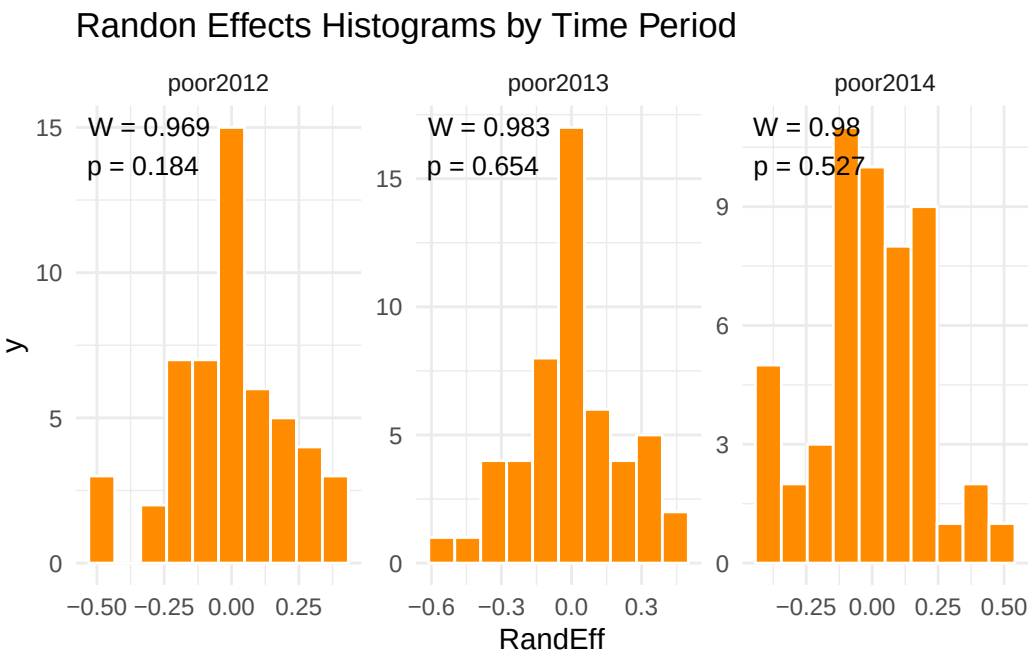
```
  pivot_longer(cols = everything(),
               names_to = "Time",
               values_to = "Residual") %>%
  ggplot(aes(sample = Residual)) +
  stat_qq() +
  stat_qq_line() +
  facet_wrap(~Time, scales = "free") +
  theme_minimal() +
  labs(title = "QQ Plots of Residuals by Time Period")
```



Likewise, we test the normality of the random effect variable

```
#### For the random effects
raneff_dt <- as.data.frame(model3_obj$randomEffect)

### lets run the shapiro wilks tests again
shapiro_obj <- apply(raneff_dt, 2, shapiro.test)


summary_dt <-
  data.frame(Time = names(shapiro_obj),
             W = lapply(X = shapiro_obj,
                        FUN = function(x){
```

```
                        return(x$statistic[[1]])

                      }) %>%
            as.numeric(),
          p_value = lapply(X = shapiro_obj,
                           FUN = function(x){

                             return(x$p.value)

                           }) %>%
            as.numeric())

### plot the results
summary_dt <-
  summary_dt %>%
  mutate(label = paste0("W = ", round(W, 3), "\n", "p = ", signif(p_value, 3)))

raneff_dt %>%
  pivot_longer(cols = everything(),
               names_to = "Time",
               values_to = "RandEff") %>%
  ggplot(aes(x = RandEff)) +
  geom_histogram(bins = 10, fill = "darkorange", color = "white") +
  geom_text(data = summary_dt, aes(x = -Inf, y = Inf, label = label),
            hjust = -0.1, vjust = 1.2, inherit.aes = FALSE, size = 3.5) +
  facet_wrap(~Time, scales = "free") +
  theme_minimal() +
  labs(title = "Randon Effects Histograms by Time Period")
```



In both cases, we compare the p-value to the 0.05 level of significance. The results suggest we cannot reject the null hypothesis of normally distributed model errors and random effects.

In some cases, the assumptions described by the MFH3 model are not. Steps 5 and 6 recommend re-estimate the models by creating additional variables via transformation or variable interactions.