# Research Design

2023-07-23

## Step: select_sample

### Select experimental sample

#### Content

Reads data, selects the observations for anaylsis and identifies the treatment windows.

#### Choice

`unit_of_observation`: A character value containing one of the following values:

- `dish`: The outcome variables are constructed at the main dish choice level. Not-treated side dishes are excluded from the analysis.
- `transaction`: The outcome variables are constructed at the transaction level. Non-treated side dishes are included.

`edays_include`: A character value containing one of the following values:

- `all_days`: Include all experimental days. Please note that this overweights two treatments as these were included in eday 9 and 11.
- `no_eday9`: Exclude the low-powered eday 9. Instead include the replacement eday 11.
- `no_eday11`: Include the low-powered eday 9 and exclude the replacement day.

`only_first_timers`: A character value containing one of the following values:

- `yes`: Limit the sample to observations where the according participant visited the canteen for the first time.
- `no`: Use all observations.

`use_counter_info`: A character value containing one of the following values:

- `yes`: Determine the treatment windows sepearately per counter.
- `no`: Determine identical treatment windows regardless of the counter.

`limit_twindows_to_stable_choice_sets`: A character value containing one of the following values:

- `yes`: Make sure that no choice set changes happened during the treatment windows. This reduces the number of observations.

- yes_no_stop_at_2pm: Make sure that no choice set changes happened. during the treatment windows until 2pm. Include post 2pm observations. After 2pm, the number of diners decreases and small choice set changes become very frequent.
- no_stop_at_2pm: Include all observations regardless of choice set changes but stop at 2pm.
- no: Include all dining observations regardless of choice set changes

identify_tment_change: A character value containing one of the following values:

- on_counter: Change treatment when diners that received the new treatment likely reached the check-out registers (latest point in time)
- on_info: Change treatment on when the first CO2 displays were changed (earliest point in time).
- median: Change treatment on the median of all treatment change events
- exclude: Exclude all observations within the treatment change period

**Code**

```r
function(input = NULL, choice = NULL) {

  if (choice[[1]] == "dish") df <- input[[1]] else df <- input[[2]]
  if (choice[[2]] == "no_eday9") {df <- df %>% filter(eday != 9)}
  if (choice[[2]] == "no_eday11") {df <- df %>% filter(eday != 11)}

  if (choice[[3]] == "yes") {df <- df %>% filter(first_time)}

  if (choice[[4]] == "yes") {
    df <- df %>%
      mutate(
        tment_change_max = ifelse(
          counter == "yellow", tment_change_max_yellow, tment_change_max_red
        ),
        tment_change_min = ifelse(
          counter == "yellow", tment_change_min_yellow, tment_change_min_red
        ) ,
        tment_end = ifelse(counter == "yellow", tment_end_yellow, tment_end_red)
      )
  } else {
    df <- df %>%
      mutate(
        tment_change_max = tment_change,
        tment_change_min = pmin(tment_change_min_yellow, tment_change_min_red)
      )
  }

  df <- df %>% mutate(
    tment_end = ifelse(is.finite(tment_end), tment_end, Inf)
  )

  if (choice[[5]] == "yes") {
    df <- df %>% filter(
      minute  >= tment_start,
      minute < ifelse(is.finite(tment_end), tment_end, 180)
    )
  }

  if (choice[[5]] == "yes_no_stop_at_2pm") {
    df <- df %>% filter(minute  >= tment_start, minute < tment_end)
  }
  if (choice[[5]] == "no_stop_at_2pm") {
    df <- df %>% filter(minute  < 180)
  }

  switch (choice[[6]],
    "on_counter" = df <- df %>% mutate(tment_change = tment_change_max),
    "on_info" = df <- df %>% mutate(tment_change = tment_change_min),
    "median" = df <- df %>% mutate(
      tment_change = (tment_change_max + tment_change_min)/2
      ),
    "exclude" = {
      df <- df %>% filter(
        minute < tment_change_min | minute > tment_change_max
      ) %>% mutate(tment_change = tment_change_max)
    }
  )

  smp <- df %>%
    mutate(
      tslot = ifelse(minute < tment_change, 1, 2),
      tment = factor(ifelse(minute < tment_change, tslot1, tslot2), TMENTS)
    ) %>%
    filter(
      ! dish %in% "Stuffed Peppers",
      !is.na(customer_group)
    ) %>%
    arrange(eday, minute, register_id, trans_id) %>%
    select(
      eday, counter, qhour, minute, pcard_id, register_id, trans_id,
      customer_group, visits, tslot, tment, dish, n_dishes, to_go,
      mtfsh, weight, co2eg
    )
  return(list(
    data = smp,
    protocol = list(choice)
  ))
}
```

## Step: estimate_effects

### Specify models and estimate treatment effects

#### Content

You can select the fixed effect structure, the type of the standard errors, whether you want to include covariate controls and the estimtion type for the meat/fish choice.

**Choice**

`mtfsh_model`: A character value containing one of the following values:

- `ols`: A linear probability model is being estimated.
- `logit`: A logit model is being estimated

`dep_vars`: A character value containing one of the following values:

- `level`: The non-binary dependent variables `weight` and `co2e` are estimated at their levels
- `log`: he non-binary dependent variables `weight` and `co2e` are estimated at their logged values

`fixed_effects_choice_set`: A character value containing one of the following values:

- `none`: No choice set fixed effects are included. Please note that in this case the variance in the dish choice sets over experimental days will liekly swamp out most of the treament effects.
- `edaycounter`: Fixed effects for each experimental day are included to control for the variance in the dish choice sets over experimental days. If treatments are determined separately for both counters, these are interacted with the two counters yielding 20 fixed effects,

`fixed_effects_time_of_day`: A character value containing one of the following values:

- `none`: No time of day fixed effects are included.
- `tslot`: Treatment slot fixed effects are included.
- `qhour`: Quarter hour fixed effects are included.
- `minute`: Minute fixed effects are included.
- `tslot x wday`: Separate treatement slot fixed effects for each day of the week are included.
- `qhour x wday`: Seperate quarter hour fixed effects for each day of the week are included.
- `minute x wday`: Minute fixed effects for each day of the week are included.

`standard_errors`: A character value containing one of the following values:

- `plain`: Do report plain unclustered standard errors
- `robust`: Do report robust unclustered standard errors
- `cluster edaycounter_tslot`: Cluster by choice set (20 clusters, or 40 if choiced sets are determined separately for both counters)
- `cluster edaycounter_qhour`: Cluster by experimental day (times counter) and quarter hour (~ 120 cluster or ~ 240 clusters when determined separately by counter)
- `cluster edaycounter x qhour`: Two-way cluster by experimental day (times counter) and quarter hour (10 x ~12 clusters or 20 x ~12 clusters when determined separately by counter)

`controls`: A character value containing one of the following values: - `none`: Include no covariate controls (besides the fixed effects) - `customer_group`: Include indicator variables for the three customer groups. - `returning_customer`: Include indicator variable whether a customer is a repeated diner - `nobs`: A count variable capturing the number of dining observations - `all`: All suitable conrrol variables

**Code**

```r
function(input = NULL, choice = NULL) {

  df <- input$data %>%
    mutate(
      eday_counter = factor(10 * eday + (counter == "yellow")),
      wday = factor(ifelse(eday == 11, 4, eday %% 5))
    )

  use_counter <- (input$protocol[[1]]$use_counter_info == "yes")

  add_fe <- function(str) {
    fe_str <<- ifelse(fe_str == "", paste("|", str), paste(fe_str, "+", str))
  }

  parms <- list(data = df)

  if (choice[[1]] == "ols") {
    estfun_mtfsh <- feols
    parms_mtfsh <- list(data = df)
  } else {
    estfun_mtfsh <- feglm
    parms_mtfsh <- list(data = df, family = binomial(link = "logit"))
  }

  fstr_mtfsh <- "mtfsh ~ tment "
  if (choice[[2]] == "log") {
    fstr_weight <- "log(weight) ~ tment "
    fstr_co2eg <- "log(co2eg) ~ tment "
  } else {
    fstr_weight <- "weight ~ tment "
    fstr_co2eg<- "co2eg ~ tment "
  }

  switch(
    choice[[3]],
    "none" = fe_str <- "",
    "edaycounter" = fe_str <- ifelse(use_counter, "| eday_counter", "| eday")
  )
  switch(
    choice[[4]],
    "tslot" =  add_fe("tslot"),
    "qhour" = add_fe("qhour"),
    "minute" = add_fe("minute"),
    "tslot x wday" = add_fe("tslot^wday"),
    "qhour x wday" = add_fe("qhour^wday"),
    "minute x wday" = add_fe("minute^wday")
  )
  switch(
    choice[[5]],
    "plain" = vcov <- "iid",
    "robust" = vcov <- "hetero",
    "cluster edaycounter_tslot" = {
      if (use_counter) {vcov <- cluster ~ eday^counter^tslot}
      else {vcov <- cluster ~ eday^tslot}
    },
    "cluster edaycounter_qhour" = {
      if (use_counter) {vcov <- cluster ~ eday^counter^qhour}
      else {vcov <- cluster ~ eday^qhour}
    },
    "cluster edaycounter x qhour" = {
      if (use_counter) {vcov <- cluster ~ eday^counter + qhour}
      else {vcov <- cluster ~ eday + qhour}
    }
  )

  switch(
    choice[[6]],
    "none" = ctrls_str <- "",
    "customer_group" = ctrls_str <- "+ customer_group ",
    "returning_customer" = ctrls_str <- "+ (visits > 1) ",
    "nobs" = ctrls_str <- "+ visits  ",
    "all" = ctrls_str <- " + customer_group + (visits > 1) + visits "
  )

  fml_mtfsh <- as.formula(paste0(fstr_mtfsh, ctrls_str, fe_str))
  fml_weight <- as.formula(paste0(fstr_weight, ctrls_str, fe_str))
  fml_co2eg <- as.formula(paste0(fstr_co2eg, ctrls_str, fe_str))

  suppressMessages({
    mod_mtfsh <- do.call(estfun_mtfsh, c(parms_mtfsh, fml = fml_mtfsh, vcov = vcov))
    mod_weight <- do.call(feols, c(parms, fml = fml_weight, vcov = vcov))
    mod_co2eg <- do.call(feols, c(parms, fml = fml_co2eg, vcov = vcov))
  })

  get_ci <- function(mod_str, mult = 1) {
    mod <- get(paste0("mod_", mod_str))
    coef_str <- paste0("tment", TMENTS[2:5])
    if (mod_str == "mtfsh" & choice[[1]] == "logit") {
      rv <- avg_slopes(mod)
      return(tibble(
        model = mod_str,
        tment = TMENTS[2:5],
        lb = rv$conf.low[rv$term == "tment"],
        est = rv$estimate[rv$term == "tment"],
        ub = rv$conf.high[rv$term == "tment"]
      ))
    }
    tibble(
      model = mod_str,
      tment = TMENTS[2:5],
      lb = confint(mod)[coef_str, 1] * mult,
      est = unname(mod$coefficients[coef_str] * mult),
      ub = confint(mod)[coef_str, 2] * mult
    )
  }
```

```r
cis <- bind_rows(
  get_ci("mtfsh"),
  get_ci("weight"),
  get_ci("co2eg")
)

if (choice[[2]] == "log") {
  cis <- cis %>%
    mutate(
      lb = ifelse(model == "mtfsh", lb, exp(lb) - 1),
      est = ifelse(model == "mtfsh", est, exp(est) - 1),
      ub = ifelse(model == "mtfsh", ub, exp(ub) - 1),
    )
} else {
  cis <- cis %>%
    mutate(
      lb = case_when(
        model == "weight" ~ lb/mean(df$weight),
        model == "co2eg" ~ lb/mean(df$co2eg),
        model == "mtfsh" ~lb
      ),
      est = case_when(
        model == "weight" ~ est/mean(df$weight),
        model == "co2eg" ~ est/mean(df$co2eg),
        model == "mtfsh" ~est
      ),
      ub = case_when(
        model == "weight" ~ ub/mean(df$weight),
        model == "co2eg" ~ ub/mean(df$co2eg),
        model == "mtfsh" ~ub
      )
    )
}

vn <- expand_grid(
  model = c("mtfsh", "weight", "co2eg"),
  tment = tolower(TMENTS[2:5]),
  coef = c("lb", "est", "ub"),
)
var_names = paste(vn$model, vn$tment, vn$coef, sep = "_")
cis_wide <- cis %>%
  pivot_wider(
    names_from = c(model, tment), values_from = c(lb, est, ub),
    names_glue = "{model}_{tolower(tment)}_{.value}", names_sort = TRUE
  ) %>%
  select(all_of(var_names))
lh <- construct_lin_hypotheses(df)

get_hypotheses_ests <- function(mod_str) {
  get_hypothesis_est <- function(mod_str, no) {
    mod <- get(paste0("mod_", mod_str))
    rv <- test_hypothesis(mod, lh[no]) %>% select(lb, est, ub, pvalue)
    if (mod_str == "mtfsh" & choice[[1]] == "logit") {
      if (fe_str == "") {
        mn_fe <- mod$coeftable[1,1]
      } else {
        fes <- fixef(mod)
        mn_fe <- mean(unlist(lapply(fes, mean)))
      }
      exp_coefs <- function(c) {
        exp(mn_fe + c)/(1 + exp(mn_fe + c)) - exp(mn_fe)/(1 + exp(mn_fe))
      }
      rv <- tibble(
        lb = exp_coefs(rv$lb),
        est = exp_coefs(rv$est),
        ub = exp_coefs(rv$ub),
        pvalue = rv$pvalue
      )
    } else {
      if (mod_str != "mtfsh") {
        if (choice[2] == "level") {
          if (mod_str == "weight") avg_y <- mean(df$weight)
          else avg_y <- mean(df$co2eg)
          rv <- rv %>% mutate(
            lb = lb/avg_y,
            est = est/avg_y,
            ub = ub/avg_y
          )
        } else {
          rv <- rv %>% mutate(
            lb = exp(lb) - 1,
            est = exp(est) - 1,
            ub = exp(ub) - 1
          )
        }
      }
    }

    names(rv) <- paste0(mod_str, "_h", no, "_", names(rv))
    rv
  }

  bind_cols(
    get_hypothesis_est(mod_str, 1),
    get_hypothesis_est(mod_str, 2),
    get_hypothesis_est(mod_str, 3),
    get_hypothesis_est(mod_str, 4)
  )
}

tests_wide <- bind_cols(
  get_hypotheses_ests("mtfsh"),
  get_hypotheses_ests("weight"),
  get_hypotheses_ests("co2eg")
)
```

```r
  results <- bind_cols(cis_wide, tests_wide)
  protocol <- input$protocol
  protocol[[length(protocol) + 1]] <-  choice

  return(list(
    data = results,
    protocol = protocol,
    models = list(
      mod_mtfsh = mod_mtfsh, mod_weight = mod_weight, mod_co2eg = mod_co2eg
    )
  ))
}
```