Extended Methods Appendix: Data Cleaning Procedures for Housing Discrimination Study (2012) Analysis

1 Overview

This appendix provides comprehensive documentation of the data cleaning procedures applied to the Housing Discrimination Study (HDS) 2012 dataset. The cleaning process transforms raw SAS data files into analysis-ready datasets suitable for econometric analysis.

2 Data Sources and Initial Processing

2.1 Raw Data Structure

The HDS 2012 data consists of six primary SAS files in `.sas7bdat` format. Each file serves a distinct purpose in documenting the field experiment:

The **assignment** file links individual testers to specific test assignments and contains release status indicators. The **taf** (Test Assignment Form) file provides test-level characteristics and parameters. The **sales** file contains detailed records of each sales test visit, including appointment times and outcomes. The **tester_censored** file provides tester demographic and background information with personally identifiable information removed. The **rhgeo** file contains geocoded data for recommended homes shown during each test. Finally, the **rechomes** file provides additional information about recommended properties.

2.2 Sales Test Filtering

The HDS 2012 study encompasses both rental and sales discrimination tests. For this analysis, we focus exclusively on sales tests, which are identified through control numbers containing the pattern "-S[A-Z]-". This pattern corresponds to three test types: SA for Asian testers, SB for Black testers, and SH for Hispanic testers.

The filtering process is applied consistently across all datasets:

```
assignment <- assignment_raw %>%
  filter(grepl("-S[A-Z]-", CONTROL)) %>%
  filter(RELEASE == "1" & !is.na(TESTERID))
```

```r

taf <- taf_raw %>%
  filter(grepl("-S[A-Z]-", CONTROL))
```

For the assignment file, we apply an additional filter requiring `RELEASE = "1"`, which indicates completed and valid test assignments. This ensures our analysis includes only tests that were successfully executed in the field.

3 Date and Time Processing

3.1 The Challenge of Date Format Heterogeneity

One of the most significant data cleaning challenges involves parsing appointment dates (`SAPPTD`) and birth dates (`TDOB`), which exhibit remarkable format diversity. Our analysis identified over 15 distinct date format patterns in the raw data, reflecting the challenges of manual data entry across many distinct testers.

The date formats encountered include standard formats with various separators ("MM/DD/YYYY", "MM-DD-YYYY", "MM.DD.YYYY"), formats without separators ("MMDDYYYY", "MMDDYY"), ambiguous formats that could represent multiple date interpretations ("MM/YY", "DD/MM", "MMDD/YY"), and numerous data entry errors including impossible dates ("6/31/12"), leap year errors ("02/29/2011"), and clear typographical errors ("08/18/7977", "01/08/1064").

3.2 Date Parsing Algorithm

To address this heterogeneity, we developed a comprehensive parsing function that employs multiple strategies. First, the function standardizes all separators (dashes, dots, spaces) to forward slashes and removes extraneous characters. Second, it maintains a lookup table for known invalid dates that require manual correction. Third, it uses regular expression pattern matching to identify date formats, testing patterns in order of specificity. Fourth, it handles century ambiguity through context-aware logic: for birth dates, two-digit years are interpreted as 19XX (reflecting that testers were born in the 1900s), while for appointment dates, two-digit years are interpreted as 20XX (reflecting that tests occurred in 2011-2012).

The date parsing function handles ambiguous two-number date formats, such as "MM/YY" or "DD/MM" by first checking if the first number is greater than 12, which would make it invalid as a month. In this case, it assumes the date is in a "DD/MM" format and the year is inferred later in the data cleaning process.

If both numbers could plausibly represent either a month (1–12) or a day (1–31), the function uses additional context to resolve the ambiguity. For birth dates, it first attempts to interpret the date as "MM/YY" and calculates the resulting age as of 2011. If this age is reasonable—between 18 and 90 years old—it accepts this interpretation. For appointment dates, the function checks

if the second number is 11 or 12, which likely indicates the years 2011 or 2012. If so, it interprets the date as "MM/YY". If not, it defaults to interpreting the date as "DD/MM" with the placeholder year 2000, again to be corrected later.

Finally, if the first number is too large to represent a valid month or day (greater than 31), the function considers the possibility of a "YY/MM" format. For birth dates, it again validates the resulting age, ensuring it falls within a plausible range (18–90 years old in 2011). For appointment dates, it specifically checks if the first number is 11 or 12, indicating the years 2011 or 2012, and interprets the date accordingly.

If none of these logical steps yield a valid interpretation, the function returns a missing value (NA) and issues a warning. This structured and context-aware approach ensures ambiguous date formats are handled consistently, transparently, and logically, with clear reasoning guiding each decision.

```
# Example of ambiguous format handling
"^\\d{1,2}/\\d{2}$" = function(x) {
  parts <- strsplit(x, "/")[[1]]
  first_num <- as.numeric(parts[1])
  second_num <- as.numeric(parts[2])

  could_be_month <- function(n) !is.na(n) && n >= 1 && n <= 12
  could_be_day <- function(n) !is.na(n) && n >= 1 && n <= 31

  # If first number too large for month, likely DD/MM format
  if (!could_be_month(first_num) && could_be_day(first_num) &&
      could_be_month(second_num)) {
    if (!is_birth_date) {
      return(as.Date(sprintf("%02d/%02d/2011", second_num, first_
    num),
                     format = "%m/%d/%Y"))
    }
  }
  # Additional logic for other scenarios...
}
```

3.3 Systematic Date Correction

Some parsed appointment dates fall outside the study period (July 2011 through October 2012), indicating data entry errors where incorrect years or months were recorded. Other times, years are missing. Our correction algorithm addresses these systematically by leveraging the fact that multiple appointments within the same test (identified by `CONTROL` number) should occur within a reasonable timeframe.

The correction process identifies all dates outside the valid study period, then for each invalid date, searches for other appointments with the same `CONTROL` number that have valid dates. When valid reference dates exist, the algorithm preserves the day information from the invalid date but adopts the year from the valid appointments. If the corrected year matches the original (suggesting a month error rather than a year error), the algorithm adjusts the month instead.

When no valid reference date exists for a given CONTROL, the appointment date is set to missing rather than imputed.

3.4 Time Parsing and Correction

Appointment times (SBEGH) present similar challenges, with formats including military time ("0315", "1430"), three-digit formats ("357" interpreted as 3:57), various separated formats ("3:15", "3;15", "3.15"), and invalid entries ("N/A", non-numeric values). The complexity is compounded by AM/PM indicators (SBEGAM) that are sometimes missing or clearly incorrect.

Our time parsing function handles these variations systematically. For missing AM/PM indicators, the function infers the most likely period based on the hour: times between 9 and 12 are likely morning appointments, while times between 1 and 8 are likely afternoon or evening appointments. The function also corrects obviously incorrect indicators, such as times between 1:00 and 5:00 marked as AM (changed to PM) or times between 10:00 and 11:00 marked as PM (changed to AM).

A particular challenge involves appointments recorded at exactly midnight (00:00:00), which is implausible for real estate showings. Our analysis revealed that these entries typically represent data entry errors where time information was missing. To address this, we identify all midnight appointments and check whether other appointments exist on the same day for the same CONTROL. If multiple appointments exist on the same day, the midnight time is set to missing, indicating a time entry error. Midnight times are preserved only when they represent the sole appointment for that day, though even these cases likely represent missing data. This is done because appointment times mainly enter the analysis as covariates for visit order, and inaccurate timing does not impact visit order when no other appointments exist for that day.

4 Data Aggregation and Variable Construction

4.1 Visit Order Determination

The experimental design allowed testers to make multiple visits to the same real estate agent. To properly analyze outcomes, we must establish the chronological order of these visits. Our approach groups records by CONTROL number, sorts by the combined appointment date and time, and assigns visit order using sequential numbering. Records with missing datetime information are placed last in the sequence.

4.2 Creating the Analysis Dataset

The final sales dataset aggregates information across multiple visits for each tester-control combination. This aggregation preserves both visit-specific information from the first interaction and cumulative information across all visits. The aggregation process creates several categories of variables:

First visit metrics capture the initial interaction between tester and agent, including `SBEGAM_FIRST` (appointment time AM/PM indicator), `STOTUNIT_FIRST` (number of units shown), and `SAVLBAD_FIRST` (indicator for whether the tester was told the house in the ad was available). Aggregate metrics summarize the entire test sequence, including `num_visits` (total number of visits), `STOTUNIT_TOTAL` (total units shown across all visits), and `SAVLBAD_ANY` (indicator for whether the original ad house was indicated to be available in any visit). Temporal metrics provide timing information, including `first_visit_datetime` and `was_first_visitor` (indicator for whether this tester was the first to visit the agent in this test).

The aggregation also includes extraction of race categories from the `RACEID` variable. This extraction uses regular expressions to identify the numeric race code embedded at the end of the race identifier string. Additionally, the process includes special handling for infinite values that can arise when calculating minimum datetimes across empty sets, converting these to proper missing values.

```
sales_final <- sales_with_time %>%
  group_by(CONTROL) %>%
  arrange(appointment_datetime) %>%
  mutate(visit_order = row_number()) %>%
  ungroup() %>%
  group_by(CONTROL, TESTERID) %>%
  summarise(
    RACEID = first(RACEID),
    SBEGAM_FIRST = first(SBEGAM[which.min(visit_order)]),
    STOTUNIT_FIRST = first(STOTUNIT[which.min(visit_order)]),
    SAVLBAD_FIRST = first(SAVLBAD_BINARY[which.min(visit_order)]),
    first_visit_datetime = min(appointment_datetime, na.rm = TRUE),
    num_visits = n(),
    STOTUNIT_TOTAL = sum(STOTUNIT, na.rm = TRUE),
    SAVLBAD_ANY = as.numeric(any(SAVLBAD_BINARY == 1, na.rm = TRUE)
    ),
    was_first_visitor = any(visit_order == 1, na.rm = TRUE),
    .groups = 'drop'
  ) %>%
  mutate(
    RACE = as.numeric(str_extract(as.character(RACEID), "\\d$")),
    first_visit_datetime = if_else(is.infinite(first_visit_datetime
    ),
                                   as.POSIXct(NA), first_visit_
    datetime)
  )
```

Note: The only occurrence where `appointment_datetime` is missing while `appointment_date` is present represents a data entry correction. Specifically, this case occurs when an initial record contained an invalid time entry (typically 00:00), prompting data entry personnel to create a subsequent record with the corrected time information. The original record with the invalid time is retained in the dataset but lacks a valid `appointment_datetime`. This incomplete record can be and is safely excluded from analysis as it represents a superseded data entry rather than genuine missing information.

5 Tester Demographics Processing

Tester birth dates undergo the same parsing process as appointment dates, with the `is_birth_date` parameter set to TRUE to apply appropriate century logic. Ages are calculated as of January 1, 2012, representing the approximate midpoint of the study period. The calculation uses precise day-based computation (days divided by 365.25 to account for leap years), rounds to the nearest whole year, and excludes implausible ages (under 18 years old) by setting them to missing.

The final step merges the aggregated sales data with tester demographics using left joins on `TESTERID`, preserving all sales records even when tester demographic information may be incomplete.

6 Output Files

The cleaning process generates two primary output files. First, `sales_cleaned.csv` contains the aggregated sales data with one row per tester-control combination, suitable for most analyses of discrimination outcomes. Second, `sales_and_tester_merged.csv` includes the sales data merged with tester demographics, enabling analyses that incorporate tester characteristics.

7 Recommended Homes Data Integration

7.1 Analytical Sample Definition

Before merging recommended homes data, we first establish our analytical sample by applying key filtering criteria to ensure data quality and experimental validity. The analytical sample is constructed through a two-stage filtering process that addresses both outcome availability and experimental design requirements.

First, we exclude observations with missing outcome variables. Specifically, we require non-missing values for both `STOTUNIT_TOTAL` (total units shown across all visits) and `SAVLBAD_ANY` (whether the advertised property was available in any visit). These variables represent our primary measures of treatment by real estate agents and are essential for discrimination analysis.

Second, we enforce the paired testing experimental design by retaining only tests (`CONTROL` numbers) that include exactly two testers. This restriction ensures that each test includes one minority tester and one white tester, enabling direct pairwise comparisons that form the foundation of the experimental methodology.

```
analytic_sales_tester <- sales_and_tester %>%
  filter(!is.na(STOTUNIT_TOTAL), !is.na(SAVLBAD_ANY)) %>%
  group_by(CONTROL) %>%
  filter(n() == 2) %>%  # Exactly 2 testers per control
  ungroup()
```

This filtering process demonstrates the data attrition inherent in creating a high-quality analytical dataset. From an initial 12,030 observations in the merged sales and tester data, requiring non-missing outcomes reduces the sample to 8,662 observations. Subsequently, requiring exactly two testers per control yields 7,302 observations representing 3,651 unique test pairs. The loss of observations primarily reflects incomplete data collection for some tests and instances where one tester in a pair failed to complete their assignment or had their data excluded during quality control.

7.2 Recommended Homes Data Merging

The recommended homes (`rechomes`) dataset contains detailed information about properties shown to testers during their visits, including property characteristics, location data, and pricing information. This data is crucial for analyzing whether minority and white testers are shown homes of similar quality and in similar neighborhoods, representing a key dimension of potential discrimination.

The merging process employs a left join on `CONTROL` and `TESTERID`, preserving all observations from the analytical sample while adding available property information. This approach acknowledges that recommended homes data may not exist for all testers, either because they were shown no properties or because property data was not successfully collected or recorded.

```
sales_tester_rechomes <- analytic_sales_tester %>%
  left_join(rechomes, by = c("CONTROL", "TESTERID")) %>%
  mutate(has_rechomes_data = rowSums(!is.na(select(., starts_with("
    H")))) > 0)
```

The merge reveals important patterns in data availability. Of the 7,302 observations in our analytical sample, 6,122 observations (83.8%) successfully merge with recommended homes data, while 1,180 observations (16.2%) lack property information. Notably, 111 observations show positive values for `STOTUNIT_TOTAL` (indicating properties were shown) but lack corresponding recommended homes data, suggesting data collection gaps rather than genuine absence of property showings.

Further investigation of these problematic cases reveals two distinct patterns. First, 75 observations belong to `CONTROL` numbers that do have recommended homes data available, but the merge failed due to `TESTERID` mismatches or other data inconsistencies. Second, 25 observations belong to controls with no recommended homes data whatsoever, likely representing cases where property data collection was unsuccessful or incomplete.

7.3 Final Analytical Dataset Construction

To ensure the integrity of our paired testing analysis, we apply a final filtering step that retains only complete test pairs where both testers have recommended homes data available. This approach is conservative but necessary for analyses that require property-level comparisons between minority and white testers within the same test.

The filtering process requires that observations have at least some recommended homes data (identified by non-missing values in variables beginning with "H") and that each `CONTROL` number includes exactly two testers with such data. This dual requirement ensures that every observation in the final dataset has a valid comparison partner within the same test.

```
sales_tester_rechomes <- sales_tester_rechomes %>%
  filter(rowSums(!is.na(select(., starts_with("H")))) > 0) %>%
  group_by(CONTROL) %>%
  filter(n_distinct(TESTERID) == 2) %>%
  ungroup()
```

This final filtering step results in substantial sample reduction, yielding 5,368 observations representing 2,684 unique test pairs. The reduction from 3,651 to 2,684 test pairs (a loss of 967 pairs, or 26.5%) reflects the requirement that both testers in each pair have recommended homes data. This attrition is substantial but necessary for analyses examining discriminatory differences in property recommendations.

The resulting dataset contains several categories of variables for analysis. Tester-level variables include demographics (`TAGE`, `RACE`), test outcomes (`STOTUNIT_TOTAL`, `SAVLBAD_ANY`), and visit characteristics (`first_visit_datetime`, `num_visits`). Property-level variables from the recommended homes data include housing characteristics, location information, and pricing data, all prefixed with "H". Test-level identifiers include `CONTROL` (linking testers within tests) and `TESTERID` (identifying individual testers).

7.4 Data Quality Implications

The substantial sample reduction during recommended homes integration raises important considerations for analysis interpretation. The final analytical sample represents approximately 73.5% of valid test pairs, suggesting that results should be interpreted as applying to tests where property recommendation data is available rather than the entire population of housing discrimination tests.

However, this sample restriction is methodologically sound for several reasons. First, analyses of discriminatory differences in property recommendations require property data for both testers in each pair. Second, the majority of sample loss occurs randomly due to data collection challenges rather than systematic patterns that might bias results. Third, the remaining sample is sufficiently large (2,684 test pairs) to provide adequate statistical power for detecting meaningful differences in treatment.