**Data Cleaning Concepts**

1. **Domain-Specific Data Cleaning**
   * **Example: custdata\_v2**: The gas\_usage variable mixes numeric and symbolic data, where values greater than 3 represent monthly gas bills, and values from 1 to 3 are special codes.
   * **Issues Identified**:
     + **age variable**: Contains the problematic value of 0, likely meaning unknown, and some values are older than 100.
     + **income variable**: Has negative values assumed to be invalid.
2. **Dealing with Invalid Values**
   * **Converting Invalid Values to NA**: Convert problematic values in age and income to NA for further treatment.
   * **Example Code**:

r

Copy code

library(dplyr)

customer\_data <- mutate(custdata\_v2,

age = na\_if(age, 0),

income = ifelse(income < 0, NA, income))

1. **Handling Sentinel Values**
   * **Sentinel Values in gas\_usage**: Special codes (1, 2, 3) indicate specific meanings like gas bill included in rent, included in electricity payment, or no gas bill.
   * **Treatment**:
     + Convert sentinel values to NA.
     + Create new indicator variables (gas\_with\_rent, gas\_with\_electricity, no\_gas\_bill).
2. **Dealing with Outliers**
   * **Identifying Outliers**: Example with income where negative values and zeros could indicate non-working status or outliers.
   * **Using boxplot.stats()**: Helps identify outliers, but negative values should be handled separately.
   * **Outlier Treatment**:
     + Turn negative incomes into NA.
     + Evaluate whether high incomes identified as outliers should be omitted.
3. **Handling Missing Values**
   * **NA Representation**: In R, missing values are represented by NA, which can propagate through operations.
   * **Strategies**:
     + Drop rows with NAs.
     + Convert NAs to meaningful values.
   * **Example Code**: Counting and identifying NAs in a dataset using count\_missing() function.
4. **Decisions on Dropping Rows with NAs**
   * Consider dropping rows based on the proportion and impact of missing data.
   * Use subsetting or listwise deletion based on the nature of missing data.

**Converting Continuous Variables to Categorical**

1. **Recoding Variables**
   * **Purpose**: Transform continuous variables into categories, such as pass/fail, or correct miscoded values.
   * **Example**:
     + **Discretizing continuous variables**: Income and age can be categorized into ranges to reveal different patterns, e.g., income below $20,000, age below 25 or above 65.
2. **Converting Continuous Variables**
   * **Income Example**: Categorize income into below $20,000 using:

custdata$income.lt.20K <- custdata$income < 20000

* + **Age Example**: Convert age into ranges:

brks <- c(0, 25, 65, Inf)

custdata$age.range <- cut(custdata$age, breaks=brks, include.lowest=T)

1. **Handling Extreme Values**
   * **Example**: Age values over 120 may be data entry errors.
   * **Solution**: Convert these to NA or create a new variable to handle them separately.
2. **Explicit Categorization**
   * **Using within() function**: Explicitly define categories for age.
   * **Example**:

custdata <- within(custdata, {

agecat <- NA

agecat[age > 120] <- NA

agecat[age > 65 & age <= 120] <- "Elder"

agecat[age > 25 & age <= 65] <- "Middle Aged"

agecat[age <= 25] <- "Young"

})

**Renaming Variables**

1. **Interactive Editing**:
   * **Use fix()**: To invoke the interactive editor for renaming variables.
   * **Example**:

fix(custdata)

1. **Programmatic Renaming**:
   * **Using dplyr's rename()**:

custdata <- rename(custdata, age.cat=agecat, gender=sex)

**Dealing with Date and Time**

1. **Complexity of Dates and Times**:
   * **Challenges**:
     + Leap years.
     + Daylight savings time (DST) causing 23 or 25-hour days.
     + Leap seconds added to adjust for Earth's slowing rotation.
2. **Leap Year Rule**:
   * **100/400 Year Rule**: A year divisible by 100 is a leap year only if also divisible by 400.
   * **Example**: 1900 is not a leap year, but 2000 is.
3. **Parsing Dates in R**:
   * **Using as.Date()**: Convert character strings to date format.
   * **Symbols**:
     + %d for day, %m for month, %Y for 4-digit year.
   * **Example**:

r

Copy code

mydates <- as.Date(c("2007-06-22", "2004-02-13"))

1. **Current Dates**:
   * **Functions**:
     + Sys.Date() for today's date.
     + date() for current date and time.
2. **Formatting Dates**:
   * **Use format()**: To format date variables.
   * **Example**:

r

Copy code

today <- Sys.Date()

format(today, format="%B %d %Y")

1. **Extracting Information from Dates**:
   * **Functions**:
     + weekdays() and months() to get day names and month names.
     + quarters() for quarters (Q1 to Q4).
     + julian() for days since the origin date (1970-01-01).
2. **Date Calculations**:
   * **Difference Calculation**:
     + difftime() to find time differences.
     + Convert to weeks, days, etc.
   * **Example**:

r

Copy code

dob <- as.Date("1956-10-12")

today <- Sys.Date()

d <- difftime(today, dob, units="weeks")