50 Days Data Science

Module 6

George K. Agyen

Day 6: Introduction to Data Cleaning

What is Data Cleaning?

Data cleaning is the process of preparing raw data for analysis by fixing errors, handling missing values, removing duplicates, and ensuring consistency. Clean data is essential for accurate and reliable analysis.

Step 1: Inspect Your Data

```
# Load a dataset
data(mtcars)

# View the first few rows
head(mtcars)

# View the structure of the dataset
str(mtcars)

# Get a summary of the dataset
summary(mtcars)
```

Step 2: Identify Common Data Issues

- **Missing Values**: Data points that are not recorded.
- **Outliers**: Extreme values that don't fit the pattern.
- **Inconsistent Formatting**: Differences in text case, date formats, etc.

- **Duplicates**: Repeated rows in the dataset.

```
# Eg data with duplicates and missing values
data <- data.frame(
  id = c(1, 2, 3, 4, 4, 5, 6, 6, 7),
  name = c('Alice', 'Joyce', 'Beatrice', 'Mike', 'Mike', 'Ernest', 'Agnes', 'Agnes', 'Jake')
  score = c(88, 91, 84, 89, NA, 79, 93, 93, 81)
)
print(data)
```

```
  id     name score
1  1    Alice    88
2  2    Joyce    91
3  3 Beatrice    84
4  4     Mike    89
5  4     Mike    NA
6  5   Ernest    79
7  6    Agnes    93
8  6    Agnes    93
9  7     Jake    81
```

```
janitor::get_dupes(data)
```

```
No variable names specified - using all columns.
```

```
  id  name score dupe_count
1  6 Agnes    93          2
2  6 Agnes    93          2
```

```
colSums(is.na(data))
```

```
   id  name score
    0     0     1
```

notice **duplicate** row id 4 and 6 and missing value **(NA)** in scores

Step 3: Install and Load Useful Packages

```r
install.packages("dplyr")    # For data manipulation
install.packages("tidyr")    # For reshaping data
install.packages("janitor")  # For cleaning column names
install.packages("mice")     # For handling missing data
install.packages("stringr")  # For string manipulation
install.packages("lubridate") # For date manipulation

library(dplyr)
library(tidyr)
library(janitor)
library(mice)
library(stringr)
library(lubridate)
```

Day 6 Challenge

1. Load the `mtcars` dataset and inspect it using `head()`, `str()`, and `summary()`.
2. Identify any potential issues in the dataset (e.g., missing values, outliers).

Day 7: Handling Missing Data & Outliers

Handling Missing Data

Missing data can skew your analysis. Here's how to handle it:

1. **Detect Missing Values**:

```r
# Section 1: Understanding Missing Data

## 1.1 Create a Sample Dataset

set.seed(123)
missing_data_example <- tibble(
  id = 1:100,
  age = as.integer(c(runif(90, 18, 65), rep(NA, 10))),  # Some missing ages
  income = c(rnorm(80, 50000, 15000), rep(NA, 20)),  # Some missing incomes
  education = sample(c("High School", "Bachelors", "Masters", "PhD"), 100,
                     replace = TRUE),
  health_score = c(runif(95, 0, 100), rep(NA, 5))
)
```

```r
# count missing values per column
colSums(is.na(missing_data_example))
```

2. **Remove Missing Values**:

```r
# Remove rows with missing values uing na.omit
clean_data <- na.omit(missing_data_example)

# Remove NAs with drop_na
complete_data <- missing_data_example %>%
  drop_na()

# Replace NAs conditionally
conditional_replace <- missing_data_example %>%
  mutate(
    # Replace NA in income with median salary
    income = replace_na(income, median(income, na.rm = TRUE)),

    # Replace NA in age with mean age
    age = replace_na(age, mean(age, na.rm = TRUE)),

    # Replace NA in healthscore with 0
    health_score = replace_na(health_score, 0)
  )
```

3. **Impute Missing Values** (using the `mice` package):

   mice stands for **`multivariate imputation by chained equation`**

```r
# Impute missing values

# Prepare data for imputation
# We'll impute 'age', 'income', and 'health_score'
imputation_data <- missing_data_example |>
  select(age, income, health_score)

# Perform multiple imputation
imputed_data <- mice(imputation_data,
                     m = 3,            # 3 imputed datasets
                     method = "pmm",   # Predictive Mean Matching
                     maxit = 5,       # Maximum iterations
                     seed = 500)       # Reproducibility

# Inspect imputation diagnostics
```

```
print(imputed_data)

# Complete case analysis
completed_data <- complete(imputed_data, action = "long",
                           include = TRUE)
```

We can also set some parameters like the method of imputation for each variable during imputation by editing the predictor matrix

```
# initialise imputation with zero iterations
initial = mice(imputation_data, maxit=0)

meth = initial$method # methods used for imputation
predM = initial$predictorMatrix # predictor matrix for imputation
print(meth)
print(predM)

# Specify methods for imputing each variable
meth["age"] = "pmm"
meth["income"] = "cart"
meth['health_score'] = 'rf'

imputed <- mice(imputation_data,
                method = meth,
                predictorMatrix = predM,
                maxit = 5,
                printFlag = FALSE)

# shows a list of the imputed data
imputed$imp


# Get completed data
imputed_data <- complete(imputed,
                         action = 2L)

imputed_data |>
  is.na() |> colSums() # we check for nas
```

Handling Outliers

Outliers can distort your analysis. Here's how to identify and handle them:

1. **Detect Outliers**:

```r
## Detecting Outliers
# Create a sample dataset with potential outliers
set.seed(111)
outlier_data <- data.frame(
  value = c(rnorm(100, mean = 50, sd = 10),
            150, 160, 170)
 )

# Boxplot to visualise outliers
boxplot(outlier_data$value, main = "Boxplot of value")

# Calculate outliers using IQR method
Q1 <- quantile(outlier_data$value, 0.25)
Q3 <- quantile(outlier_data$value, 0.75)
IQR <- Q3 - Q1

# Calculate outliers using IQR method
Q1 <- quantile(outlier_data$value, 0.25)
Q3 <- quantile(outlier_data$value, 0.75)
IQR <- Q3 - Q1

# Define outlier boundaries (Tukey's Fences)
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

# Identify outliers
outliers <- outlier_data$value[outlier_data$value < lower_bound | outlier_data$value > u

print(outliers)
```

2. **Dealing with Outliers**:

```r
# Remove outliers based on a threshold
clean_data_removed <- outlier_data |>
  filter(value >= lower_bound & value <= upper_bound)

# Cap values withing boundaries (winsorisation)
clean_data_capped <- outlier_data %>%
  mutate(value = pmin(pmax(value, lower_bound), upper_bound))
# or
clean_data_capped <- outlier_data %>%
  mutate(value = case_when(
```

```
    value < lower_bound ~ lower_bound,
    value > upper_bound ~ upper_bound,
    TRUE ~ value
  ))

# Transformation (Log transformation)
clean_data_transformed <- outlier_data %>%
  mutate(log_value = log(value))
```

Day 7 Challenge

1. Load the `airquality` dataset (`data(airquality)`).
2. Check for missing values and impute them using the `mice` package.
3. Identify and remove outliers in the `Ozone` column.

Day 8: String & Date Manipulations

String Manipulation with `stringr`

The `stringr` package makes it easy to work with text data.

String Operations

1. **String Length and Counting with `str_length` and `str_count`**

   `str_length()` returns the number of characters

```
library(stringr)

str_length(c("a", "R", "programming", NA))

# With non-ASCII characters
str_length("café")
str_length(" ")
```

   `str_count()` returns pattern matches

```r
# Count vowels in words
str_count(c("apple", "banana", "pear"), "[aeiou]")

# Count occurrences of s
str_count("Mississippi", "s")

# Count words (simple approach)
str_count("This is a sentence.", "\\w+")
```

2. **Case Conversion**

   `str_to_lower()`, `str_to_upper()` and `str_to_title()`

```r
str_to_lower("Hello WORLD")
str_to_lower("Îstanbul")

str_to_upper("hello world")
str_to_upper("straße")

str_to_title("hello world")
str_to_title("the quick brown fox")
str_to_title("it's a nice day")
```

3. **Sub-string Extraction**

```r
text <- "Hello, World!"

# Extract from position 1 to 5
str_sub(text, 1, 5)  # "Hello"

# Extract from position 8 to end
str_sub(text, 8, -1)  # "World!"

# Extract last 6 characters
str_sub(text, -6, -1)  # "World!"

# Replace "World" with "R"
str_sub(text, 8, 12) <- "R"
print(text)  # "Hello, R!"
```

4. **Pattern matching and extraction**

   `str_extract()`

```r
# Extract first number
str_extract("Price: $25.99", "\\d+\\.?\\d*")

# Extract first word starting with 'A'
str_extract(c("Apple", "Banana", "Avocado"), "^A\\w+")
```

```r
# Extract all numbers
str_extract_all("1a2b3c", "\\d")  # returns a list

# Simplify to a matrix
str_extract_all("1a2b3c", "\\d", simplify = TRUE)
```

`str_match()`

```r
# Extract date components (day, month, year)
date <- "2024-05-21"
str_match(date, "(\\d{4})-(\\d{2})-(\\d{2})")

# Extract email parts (user, domain)
email <- "user@example.com"
str_match(email, "(\\w+)@(.+)")
```

5. **Replacing Text**

```r
# Replace first digit with "X"
str_replace("abc123def456", "\\d", "X")

# Replace first color
colors <- c("red car", "blue sky", "green apple")
str_replace(colors, "red|blue|green", "COLOUR")

# Replace all digits with "X"
str_replace_all("abc123def456", "\\d", "X")

# Multiple replacements
str_replace_all("banana", c("a" = "X", "n" = "Y"))

# Using backreferences (swap day-month)
dates <- "21-05-2024"
str_replace_all(dates, "(\\d{2})-(\\d{2})", "\\2-\\1")
```

6. **Splitting and Combining**

`str_split()`

```r
# Split by whitespace
str_split("The quick brown fox", " ")

# Return maximum 2 pieces
str_split("a-b-c-d", "-", n = 2)

# Get matrix output
str_split(c("a-b", "1-2-3"), "-", simplify = TRUE)
```

`str_c()`

```r
# Simple combination
str_c("The", "quick", "brown", "fox", sep = " ")

# Vectorized operation
str_c("prefix_", c("a", "b", "c"), "_suffix")

# Handling NAs
str_c(c("a", NA, "b"), "-suffix")

# Collapsing to single string
str_c(c("a", "b", "c"), collapse = "|")
```

7. **Trimming**

```r
messy_text <- "   Hello World!   "
str_trim(messy_text)

# Left trim only
str_trim("   text", "left")

# Right trim only
str_trim("text   ", "right")

df <- tibble(
  names = c("  Alice", "Bob   ", "  Charlie  ")
)
df %>% mutate(clean_names = str_trim(names))
```

Date Manipulation with `lubridate`

The `lubridate` package simplifies working with dates.

1. **Parse Dates**:

```
library(lubridate)

# Parse a date
ymd("20230515")       # Works with numbers
mdy("May 15, 2023")   # Understands month names
dmy("15/05/23")       # Works with different separators
```

2. **Handling Times**

```
# Date with time
ymd_hms("2023-03-15 14:30:45")  # Returns "2023-03-15 14:30:45 UTC"

# Alternative formats work too
mdy_hm("03/15/2023 14:30")  # Returns "2023-03-15 14:30:00 UTC"
```

3. **Automatic parsing**

`parse_date_time()` can handle multiple formats automatically

```
# Specify multiple possible formats
parse_date_time(c("2023-03-15", "03/15/2023"),
                orders = c("ymd", "mdy"))

messy_dates <- c("May 15", "15/05/23")
# Parse with default year for incomplete dates
parse_date_time(messy_dates,
                orders = c("md", "dmy"),
                truncated = 1)
```

4. **Extract Date Components**:

```
date <- ymd("2023-03-15")

# Extract components
year(date)    # Returns 2023
month(date)   # Returns 3
day(date)     # Returns 15
wday(date)    # Returns day of week (1-7, starting from Sunday)
wday(date, label = TRUE)  # Returns "Wed"
```

for times

```
datetime <- ymd_hms("2023-03-15 14:30:45")

# Extract time components
```

```r
hour(datetime)    # Returns 14
minute(datetime)  # Returns 30
second(datetime)  # Returns 45
```

5. **Adding and subtracting time periods**

   Adding

```r
# Basic Addition
start_date <- ymd("2023-01-01")

start_date + days(5)    # "2023-01-06"
start_date + months(2)  # "2023-03-01"

# Adding multiple periods
start_date + days(5) + months(1)  # "2023-02-06"

# vectorised operations
dates <- ymd(c("2023-01-01", "2023-02-15"))
dates + months(3)  # Adds 3 months to each date
```

   Subtraction

```r
# Basic subtraction
end_date <- ymd("2023-12-31")

end_date - days(10)    # "2023-12-21"
end_date - years(1)    # "2022-12-31"

# Calculating differences
date1 <- ymd("2023-06-15")
date2 <- ymd("2023-06-20")

date2 - date1
```

   getting exact time with `duration`

```r
# Durations (exact time)
# Using duration functions (exact time)
as.duration(date2 - date1)  # gives duration in seconds

duration <- dweeks(2)- ddays(8)
time_length(duration, 'day') # change to days
```

   Create time intervals with `interval`

```r
# Create an interval
interval <- interval(start_date, end_date)

# Calculate length in various units
time_length(interval, "day")    # Returns 45
time_length(interval, "week")   # Returns 6.428571
time_length(interval, "month")  # Returns 1.483871
```

Special cases

```r
# Month boundaries
# Adding months respects month lengths
ymd("2023-01-31") %m+% months(1)

# Leap years
ymd("2020-02-28") + days(1)  # "2020-02-29" (leap year)
ymd("2021-02-28") + days(1)  # "2021-03-01"

leap <- ymd('2012-02-29')
leap %m-% years(1)  # "2011-02-28"
```

6. **Converting numeric values to Date**

```r
# converting numeric values that represent days
as_date(12290)

# if the number represents seconds
as_datetime(845000000)

# if the number is excel date
as_date(12290, origin = "1899-12-30")
as_date(12290-2, origin = "1900-01-01")

# excel datetime (days.fraction)
n = 45325.75
as_datetime(n * 86400, origin = "1899-12-30")
```

Day 8 Challenge

- Create a vector of 4 messy names (e.g., " JOHN DOE", "mary-jane", "JOhn_wiCK"). Use `str_to_title()` and `str_replace()` and `str_trim()` to clean them.
- Make a vector of 3 dates in different formats undefined (e.g., ("2023-01-15", "01/25/2023", "02.05.2023"). Parse them into dates using `parse_date_time()`

- Extract the month and year from your parsed dates using `month()` and `year()` from `lubridate`.

Day 9: Data Wrangling & Transformation

Data wrangling with `dplyr`

The `dplyr` package provides functions for manipulating data seamlessly.

1. **Select**

```
library(dplyr)
# Pick just these three columns
select(mtcars, mpg, hp, wt)

# Remove columns with the negative (-) sign
select(mtcars, -carb, -gear)
```

2. **Filter**

```
# Get only efficient cars (mpg > 20)
filter(mtcars, mpg > 20)

# Narrow it down further to 4-cylinder cars
filter(mtcars, mpg > 20, cyl == 4)
```

3. **Mutate**

```
# Add power-to-weight ratio
mutate(mtcars, hp_per_ton = hp/(wt*1000))

# Multiple transformations at once
mutate(mtcars,
       efficient = mpg > 20,
       weight_class = ifelse(wt > 3.5, "heavy", "light"))
```

4. **Summarise**

```
# Basic summary statistics
mtcars %>%
  summarise(
    avg_mpg = mean(mpg),
    max_hp = max(hp),
    min_wt = min(wt),
    count = n()  # n() counts rows
```

```
  )

# Grouped summaries
mtcars %>%
  group_by(cyl) %>%  # First group by cylinder count
  summarise(
    avg_mpg = mean(mpg),
    sd_mpg = sd(mpg),  # Standard deviation
    sample_size = n()
  )
```

5. **Arrange**

```
# Basic ascending sort
mtcars %>% arrange(mpg)

# Descending sort
mtcars %>% arrange(desc(mpg))

# Multiple column sort
mtcars %>% arrange(cyl, desc(mpg))
```

6. **Merging Data**

```
# Customer information
customers <- tibble(customer_id = c(101, 102, 103),
name = c("Alice", "Bob", "Charlie"),
join_date = ymd(c("2023-01-15","2023-02-20","2023-03-10")))

# Purchase records
orders <- tibble(order_id = c(5001, 5002, 5003),
customer_id = c(102, 103, 104),
amount = c(150.50, 200.75, 99.99))
```

Joining the data

```
# Inner join: Only perfect matches
inner <- inner_join(customers, orders, by = "customer_id")

# Left join: Keep all customers, even without orders
left <- left_join(customers, orders, by = "customer_id")

# Full join: Keep all records from both tables
full <- full_join(customers, orders, by = "customer_id")
```

Data Transformation with `tidyr`

The `tidyr` package helps reshape your data.

1. **Pivot Longer**

```r
library(tidyr)

# Convert multiple measurement columns to rows
mtcars_long <- mtcars %>%
  select(mpg, hp, wt, cyl)
  pivot_longer(cols = c(mpg, hp, wt),
               names_to = "metric",
               values_to = "value")
```

2. **Pivot Wider**

```r
# Recreate the original wide format
mtcars_wide <- mtcars_long %>%
  pivot_wider(names_from = "metric",
              values_from = "value")

# create a wide format widout a unique col_id
mtcars_wide <- mtcars_long %>%
  select(-model) |>
  pivot_wider(
        names_from = "metric",
        values_from = "value",
        ) # code works fine but returns unexpected ouput with warnings
```

3. **Separate**

```r
# Create the custom dataset
employee_data <- tibble(
  full_name = c("John Wick", "Mad Max", "Jack Sparrow", "Matt Murdock"),
  department = c("HR", "IT", "Sales", "Legal"),
  salary = c(65000, 60000, 75000, 62000),
  hire_date = c("2022-03-15", "2021-11-01", "2023-01-10", "2020-06-22")
)

# Split combined name fields
separate(employee_data,
         full_name,
         into = c("first_name", "last_name"),
         sep = " ")
```

Day 9 Challenge

1. `dplyr` exercise with `mtcars`
    1. Find all 8-cylinder (`cyl`)cars with horsepower (`hp`) > 150
    2. Create a new column: miles per horsepower (`mpg/hp`)
    3. Calculate the average weight (`wt`) by cylinder (`cyl`)
2. `tidyr` transformation with `iris`
    1. Reshape the `iris` data to show **`Species`**, **`Measurement`** (Sepal.Length, Petal.Length, etc) and **`Value`**
    2. Then pivot back to the original format
3. Merging data
    1. Use the data `products` and `sales` created from the code below

```
# Products table
products <- tibble(
  product_id = c(1, 2, 3, 4, 5),
  product_name = c("Laptop", "Mouse", "Keyboard", "Monitor", "Printer"),
  status = c("Active", "Active", "Discontinued", "Active", "Discontinued")
)

# Sales table
sales <- tibble(
  sale_id = c(101, 102, 103, 104),
  product_id = c(1, 3, 3, 4),
  amount = c(1200.50, 150.75, 180.25, 300.00),
  sale_date = as.Date(c("2023-05-01", "2023-06-15", "2023-07-01", "2023-08-10"))
)
```

    2. Experiment with the different join types we have learnt so far
    3. Identify which join gives you:
        - All sales including those for discontinued products
        - Products that never sold

Day 10: Break Day & Hands-on Practice

Take a Break!

You've made it through another 5 days of learning! Take some time to relax and review what you've learned.

Hands-on Practice

To reinforce your skills, try these tasks using this data:

```
# Patient records
patients <- tibble(
  patient_id = c("P001", "P002", "P003", "P004"),
  full_name = c("John SMITH", "mary Johnson", "ALICE WONG", "bob brown"),
  dob = c("15-03-1990", "1985/12/25", "03 July 2000", "01-Jan-1988"),
  last_visit = c("2023-05-15 14:30", "15/06/2023 09:15", "20230720", "2023-04-01")
)

# Hospital transactions
transactions <- tibble(
  tx_id = 1:8,
  patient_id = rep(c("P001", "P002", "P003", "P004"), each = 2),
  procedure = c("CT Scan", "Blood Test", "MRI", "X-Ray", "ECG", "Blood Test", "Ultrasound",
  cost = c(250, 50, 500, 120, 80, 50, 300, 500),
  datetime = c(
    "2023-05-15 15:00", "2023-05-16 10:30",
    "2023-06-15 11:00", "2023-06-16 14:00",
    "2023-07-20 09:15", "2023-07-21 13:45",
    "2023-04-01 08:00", "2023-04-02 16:30"
  )
)
```

1. String Manipulation (`stringr`)

a) Standardize all **`full_name`** to "Title Case" (e.g., "Mary Johnson")
b) Extract last names from **`full_name`** into a new column
c) Identify patients whose last names contain "son" (case insensitive)

2. Date Parsing (`lubridate`)

a) Convert **`dob`** to Date type (handle all 4 different formats)
b) Calculate current age for each patient (in years)
c) Parse **`last_visit`** to POSIXct (note mixed formats)

3. Date Operations (`lubridate`)

a) Find patients who visited in Q2 (April-June) 2023
b) Calculate days since last visit for each patient
c) Extract month and weekday name from **`last_visit`**

4. Data Filtering (`dplyr`)

a) Select patients born before 1995
b) Filter transactions costing more than $100
c) Find the most expensive procedure per patient

5. Data Transformation (`dplyr`)

a) Create a column **`discount_cost`** with 10% discount for Blood Tests
b) Flag transactions occurring on weekends
c) Convert **`procedure`** to uppercase

6. Grouped Operations (`dplyr`)

a) Calculate total spending per patient
b) Count procedures by type, ordered by frequency
c) Find each patient's first and last procedure dates

7. Data Merging (`dplyr`)

a) Left join patient details to transactions
b) Inner join to find patients with transactions
c) Full join to retain all records from both tables

8. Pivoting Data (`tidyr`)

a) Reshape transactions to wide format (one row per patient, columns for procedure counts)
b) Pivot to show min/max costs per procedure type

9. String Splitting (`tidyr` + `stringr`)

a) Separate **`full_name`** into **`first_name`** and **`last_name`**
b) Split **`datetime`** into date and time columns
c) Extract year from **`dob`** using string operations