STAT312: Data Wrangling and Model Selection 01

Fakade, Ncumisa (225091410)

25 July 2025

1 Introduction

Welcome to your first practical on data wrangling in R! This session will teach you the basics of using the `dplyr` package to clean and manipulate data. Data wrangling is a key part of data analysis, and you'll spend a lot of time doing it.

Real data is often messy! It has missing values, inconsistencies, and other problems. We'll learn how to fix these step by step using simple functions from `dplyr`.

The functions we'll learn work together using the pipe operator (%>%), which makes your code easier to read.

The Pipe Operator

The pipe operator (%>%) allows you to chain functions together, passing the result of one function as the first argument to the next. This creates readable, left-to-right workflows. For example, `data %>% filter(age > 30) %>% select(name)` is equivalent to `select(filter(data, age > 30), name)`. You can access help with `?magrittr::%>%`.

2 Objectives

By the end of this practical, you will be able to:

1. Use basic `dplyr` functions to manipulate data
2. Clean common data issues
3. Create simple workflows to transform data
4. Practice these skills through exercises

Warning

This is a **long** practical! It will take most students **at least 180 minutes**. *Failure to submit a substantially complete practical will negatively affect your participation mark.*

3 Required Packages

```r
# Load the tidyverse package
library(tidyverse)  # This includes dplyr and other helpful tools
library(lubridate)  # For handling dates
library(knitr)      # For making tables
library(rsample)    # For cross-validation functions
library(broom)      # For model output formatting
```

4 Introduction

This combined practical covers the complete workflow from data cleaning to model selection.

5 Dataset Overview

We'll work with two datasets: - `hospital_data_messy.csv`: 210 patient records with common data quality issues - `retail_sales_data.csv`: 200 store locations for sales prediction

6 Part 1: Data Wrangling

```
# Import the dataset
hospital_data <- read_csv("hospital_data_messy.csv", na = c("NA", ""))

# Look at the structure
glimpse(hospital_data)
```

```
## Rows: 210
## Columns: 8
## $ patient_id       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
## $ age              <dbl> 59, 79, 59, 57, 84, 84, 22, 72, 75, 42, NA, 8
## $ gender           <chr> "Male", "Female", "Male", NA, "female", "Male
## $ admission_date   <chr> "07/01/2023", "02/02/2023", "22/11/2023", "17/
## $ department       <chr> "ER", "Surgery", "ER", "Neurology", "emergenc
## $ length_of_stay   <dbl> 3, 5, 11, 3, 7, 9, 5, 5, 8, 3, 4, 7, 6, 9, 1,
## $ treatment_cost   <dbl> 20900, 33700, 11300, 1400, 14500, 9100, 27400
## $ discharge_status <chr> "Discharged", "Discharged", "Discharged", "Di
```

The dataset has 210 rows (patients) and 8 columns (variables).

```
# Show first few rows
head(hospital_data)
```

```
## # A tibble: 6 x 8
##   patient_id   age gender admission_date department length_of_stay
##         <dbl> <dbl> <chr>  <chr>          <chr>                  <dbl>
## 1           1    59 Male   07/01/2023     ER                         3
## 2           2    79 Female 02/02/2023     Surgery                    5
## 3           3    59 Male   22/11/2023     ER                        11
## 4           4    57 <NA>   17/04/2023     Neurology                  3
## 5           5    84 female 22/03/2023     emergency                  7
## 6           6    84 Male   30/10/2023     A&E                        9
## # i 2 more variables: treatment_cost <dbl>, discharge_status <chr>
```

```
# Check missing values
hospital_data %>%
  summarise(across(everything(), ~sum(is.na(.)))) %>%
  pivot_longer(
    everything(),
    names_to = "variable",
    values_to = "missing_count"
  ) %>%
  filter(missing_count > 0) %>%
 kable( # kable allows us to make ``nice looking" tables.
   caption = "Missing data summary",
   format = "pandoc"
 )
```

Table 1: Missing data summary

variable	missing_count
patient_id	17
age	14
gender	22
admission_date	21
department	14

variable	missing_count
length_of_stay	14
treatment_cost	12
discharge_status	16

The across Function

The `across()` function applies a function to multiple columns simultaneously. Here, `across(everything(), ~sum(is.na(.)))` calculates the number of missing values for every column. The `~` creates an anonymous function, and `.` represents each column being processed. Check `?dplyr::across` for more details.

The everything Helper

`everything()` is a selection helper that selects all columns in a dataset. Other useful helpers include `starts_with()`, `ends_with()`, `contains()`, and `where()`. These make column selection more flexible and readable. See `?tidyselect::everything` for the complete list.

Reshaping Data with pivot longer

`pivot_longer()` transforms data from wide to long format. Here, it converts our summary of missing values from separate columns into rows with variable names and counts. This format is often more suitable for analysis and visualisation. The reverse operation is `pivot_wider()`. Access help with `?tidyr::pivot_longer`.

7 Step 1: Selecting Columns (Learning objective: select() function)

The `select()` function lets you choose which columns to keep. This helps focus on the data you need.

First, we'll trim any extra spaces in text columns.

```
hospital_clean <- hospital_data %>%
  mutate(across(where(is.character), str_trim))
```

The where Helper Function

`where()` selects columns based on a logical condition. Here, `where(is.character)` selects all character (text) columns. You can use any function that returns TRUE or FALSE, such as `where(is.numeric)` or `where(is.factor)`. This provides powerful, condition-based column selection. See `?tidyselect::where` for details.

String Trimming with str trim

`str_trim()` removes leading and trailing whitespace from character strings. This is essential for data cleaning as extra spaces can cause matching problems later. The `stringr` package (part of tidyverse) provides many useful string manipulation functions. Check `?stringr::str_trim` for more information.

Example: Select basic patient info.

```
basic_info <- hospital_clean %>%
  select(patient_id, age, gender)

glimpse(basic_info)
```

```
## Rows: 210
## Columns: 3
## $ patient_id <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
## $ age        <dbl> 59, 79, 59, 57, 84, 84, 22, 72, 75, 42, NA, 81, 77,
## $ gender     <chr> "Male", "Female", "Male", NA, "female", "Male", "Ma
```

Exercise 1.1: Create a dataset with patient_id, department, and treatment_cost. Call it `cost_info`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
cost_info <- hospital_clean %>%
  select(patient_id, department,treatment_cost)  # Fill in the blanks

# Check
head(cost_info)
```

Exercise 1.2: Select all columns from age to department (using :). Call it `demo_data`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
demo_data <- hospital_clean %>%
  select(age:department)

# Check
head(demo_data)
```

Column Range Selection

The colon operator (`:`) selects a range of consecutive columns. For example, `age:department` selects all columns from age to `department` inclusive, based on their position in the dataset. This is particularly useful when you need several adjacent columns. You can also use negative selection with `-` to exclude columns.

8 Step 2: Filtering Rows (Learning objective: filter() function)

The `filter()` function picks rows that meet conditions.

Example: Patients over 65.

```
elderly_patients <- hospital_clean %>%
  filter(age > 65)

nrow(elderly_patients)
```

```
## [1] 61
```

```
ncol(elderly_patients)
```

```
## [1] 8
```

```
hospital_clean%>%
  summarise(across(everything(), ~sum(is.na(.))))%>%
kable(
  caption= "missing"
)
```

Data Check: Look for problems like impossible ages.

Table 2: missing

patient_id	age	gender	admission_date	department	length_of_stay	treatment_cost	dis
17	14	22	21	14	14	12	

```r
suspicious_ages <- hospital_clean %>%
  filter(age < 0 | age > 120)

negative_stay <- hospital_clean %>%
  filter(length_of_stay < 0)

negative_cost <- hospital_clean %>%
  filter(treatment_cost < 0)

zero_cost <- hospital_clean %>%
  filter(treatment_cost == 0)

# Show if any
suspicious_ages
```

```
## # A tibble: 3 x 8
##   patient_id   age gender admission_date department             length_o
##        <dbl> <dbl> <chr>  <chr>          <chr>
## 1        148   150 male   30/04/2023     neurology
## 2        189    -5 Male   02/04/2023     Accident & Emergency
## 3        190   999 Male   26/04/2023     Cardiology
## # i 2 more variables: treatment_cost <dbl>, discharge_status <chr>
```

```r
negative_stay
```

```
## # A tibble: 2 x 8
##   patient_id   age gender admission_date department length_of_stay
##         <dbl> <dbl> <chr>  <chr>          <chr>                <dbl>
## 1          85    27 Male   19/06/2023     Neurology               -1
## 2         125    58 MALE   17/09/2023     SURGERY                 -2
## # i 2 more variables: treatment_cost <dbl>, discharge_status <chr>
```

```
negative_cost
```

```
## # A tibble: 2 x 8
##   patient_id   age gender admission_date department length_of_stay
##        <dbl> <dbl> <chr>  <chr>          <chr>                <dbl>
## 1          48    61 Female 23/01/2023     <NA>                    NA
## 2         140    22 Male   05/05/2023     A&E                      5
## # i 2 more variables: treatment_cost <dbl>, discharge_status <chr>
```

```
nrow(zero_cost)
```

```
## [1] 8
```

Logical Operators in filter

The `filter()` function uses logical operators: > (greater than), < (less than), == (equals), != (not equals), | (OR), and & (AND). You can combine multiple conditions to create complex filters. Missing values require special handling with `is.na()`. See `?base::Logic` for complete operator details.

Exercise 2.1: Filter patients in "Surgery" department. Call it `surgery_patients`. How many are there?

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
surgery_patients <- hospital_clean %>%
  filter(department == "Surgery")


cat("Number of surgery patients:", nrow(surgery_patients))
```

```
## Number of surgery patients: 13
```

Exercise 2.2: Filter patients under 30 with length_of_stay > 5. Call it `young_long_stay`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
young_long_stay <- hospital_clean %>%
  filter(age< 30 & length_of_stay> 5)

head(young_long_stay)
```

```
## # A tibble: 5 x 8
##   patient_id   age gender admission_date department             length_o
##        <dbl> <dbl> <chr>  <chr>          <chr>
## 1         57    21 Female <NA>           A&E
## 2         NA    19 Male   18/10/2023     Ortho
## 3        149    29 Male   07/12/2023     Neurology
## 4         NA    29 Male   28/03/2023     Cardiology
## 5        189    -5 Male   02/04/2023     Accident & Emergency
## # i 2 more variables: treatment_cost <dbl>, discharge_status <chr>
```

9 Step 3: Transforming Data (Learning objective: mutate() function)

The `mutate()` function adds or changes columns.

Example: Clean age and add age group.

```
hospital_transformed <- hospital_clean %>%
  mutate(
    age_clean = ifelse(age < 0 | age > 120, NA, age),
    age_group = case_when(
      age_clean < 18 ~ "Child",
      age_clean >= 18 & age_clean < 65 ~ "Adult",
      age_clean >= 65 ~ "Elderly",
      TRUE ~ "Unknown"
    )
  )

hospital_transformed %>%
  select(patient_id, age, age_clean, age_group) %>%
  head()
```

```
## # A tibble: 6 x 4
##   patient_id   age age_clean age_group
##        <dbl> <dbl>     <dbl> <chr>
## 1          1    59        59 Adult
## 2          2    79        79 Elderly
## 3          3    59        59 Adult
## 4          4    57        57 Adult
## 5          5    84        84 Elderly
## 6          6    84        84 Elderly
```

Conditional Logic with case when

`case_when()` provides a powerful way to create multiple conditional statements. It evaluates conditions in order and assigns the first matching result. The `TRUE ~` statement acts as a catch-all for any remaining cases. This is more readable than nested `ifelse()` statements for complex conditions. Check `?dplyr::case_when` for syntax details.

Handling Dates: Dates may have different formats. Use `parse_date_time()` from lubridate to standardise them.

First, see some dates.

```r
head(hospital_transformed$admission_date)
```

```
## [1] "07/01/2023" "02/02/2023" "22/11/2023" "17/04/2023" "22/03/2023"
## [6] "30/10/2023"
```

Now, parse them.

```r
hospital_with_dates <- hospital_transformed %>%
  mutate(
    admission_date_clean = parse_date_time(
      admission_date, orders = c("dmy", "mdy", "ymd")))

hospital_with_dates %>%
  select(patient_id, admission_date, admission_date_clean) %>%
  head()
```

```
## # A tibble: 6 x 3
##   patient_id admission_date admission_date_clean
##        <dbl> <chr>          <dttm>
## 1          1 07/01/2023     2023-01-07 00:00:00
## 2          2 02/02/2023     2023-02-02 00:00:00
## 3          3 22/11/2023     2023-11-22 00:00:00
## 4          4 17/04/2023     2023-04-17 00:00:00
## 5          5 22/03/2023     2023-03-22 00:00:00
## 6          6 30/10/2023     2023-10-30 00:00:00
```

Date Parsing with parse date time

`parse_date_time()` from the lubridate package handles multiple date formats automatically. The `orders` parameter specifies possible formats to try: "dmy" (day-month-year), "mdy" (month-day-year), and "ymd" (year-month-day). The function attempts each format until one succeeds, making it perfect for messy real-world data. See `?lubridate::parse_date_time` for format codes.

Cleaning Text: Standardise department names using simple matches.

First, see variations.

```
head(hospital_with_dates)
```

```
## # A tibble: 6 x 11
##   patient_id   age gender admission_date department length_of_stay
##        <dbl> <dbl> <chr>  <chr>          <chr>               <dbl>
## 1          1    59 Male   07/01/2023     ER                      3
## 2          2    79 Female 02/02/2023     Surgery                 5
## 3          3    59 Male   22/11/2023     ER                     11
## 4          4    57 <NA>   17/04/2023     Neurology               3
## 5          5    84 female 22/03/2023     emergency               7
## 6          6    84 Male   30/10/2023     A&E                     9
## # i 5 more variables: treatment_cost <dbl>, discharge_status <chr>,
## #   age_clean <dbl>, age_group <chr>, admission_date_clean <dttm>
```

```
hospital_with_dates %>%
  count(department)
```

```
## # A tibble: 24 x 2
##    department               n
##    <chr>                <int>
##  1 A&E                     22
##  2 ACCIDENT & EMERGENCY     1
##  3 Accident & Emergency     7
##  4 Bone & Joint            10
##  5 CARDIOLOGY               7
##  6 Cardiology              37
##  7 ER                      20
##  8 Emergency               17
##  9 NEUROLOGY                2
## 10 Neurology               24
## # i 14 more rows
```

```
hospital_with_dates %>%
  count(department)
```

```
## # A tibble: 24 x 2
##    department               n
##    <chr>                <int>
##  1 A&E                     22
##  2 ACCIDENT & EMERGENCY     1
##  3 Accident & Emergency     7
##  4 Bone & Joint            10
##  5 CARDIOLOGY               7
##  6 Cardiology              37
##  7 ER                      20
##  8 Emergency               17
##  9 NEUROLOGY                2
## 10 Neurology               24
## # i 14 more rows
```

```
# Collect together all the different names associated
# with a department
names_for_emergency <- c(
  "emergency", "a&e", "er", "accident & emergency")
names_for_orthopedics <- c(
  "orthopedics", "orthopaedics", "ortho", "bone & joint")
names_for_cardiology <- c("cardiology")
names_for_neurology <- c("neurology")
names_for_surgery <- c("surgery")

# Standardise the department names
hospital_standardised <- hospital_with_dates %>%
  mutate(
    dept_lower = tolower(department),
    department_clean = case_when(
      dept_lower %in% names_for_emergency ~ "Emergency",
      dept_lower %in% names_for_orthopedics ~ "Orthopedics",
      dept_lower %in% names_for_cardiology ~ "Cardiology",
      dept_lower %in% names_for_neurology ~ "Neurology",
      dept_lower %in% names_for_surgery ~ "Surgery",
      TRUE ~ "Unknown"
    ),
    gender = str_to_title(tolower(gender)),
    discharge_status = str_to_title(tolower(discharge_status))
  ) %>%
  select(-dept_lower)

hospital_standardised %>%
  count(department_clean) ##gives a neat standardized count of the depar
```

```
## # A tibble: 6 x 2
##   department_clean     n
##   <chr>            <int>
## 1 Cardiology           49
## 2 Emergency            71
## 3 Neurology            29
## 4 Orthopedics          32
## 5 Surgery              15
## 6 Unknown              14
```

The in Operator

The `%in%` operator tests whether elements on the left appear anywhere in the vector on the right. For example, `dept_lower %in% names_for_emergency` returns TRUE if the department name matches any value in the emergency names vector. This is much more efficient than multiple == conditions connected with |. See `?base::%in%` for details.

String Case Conversion

Functions like `tolower()`, `toupper()`, and `str_to_title()` standardise text case for consistent matching. `str_to_title()` converts text to title case (First Letter Capitalised). Case standardisation is crucial for data cleaning as "Emergency", "EMERGENCY", and "emergency" should be treated as identical. The stringr package provides additional case conversion options.

Exercise 3.1: Add a column `daily_cost = treatment_cost / length_of_stay`. Call the dataset `hospital_with_daily`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
hospital_with_daily <- hospital_standardised %>%
  mutate(daily_cost = treatment_cost / length_of_stay)
  head(hospital_with_daily) ##with all the columns
```

```
hospital_with_daily %>%
  select(patient_id, treatment_cost, length_of_stay, daily_cost) %>% ##wit
  head()
```

Exercise 3.2: Add a column `high_cost = ifelse(treatment_cost > 20000, "Yes", "No")`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
hospital_with_high <- hospital_with_daily %>%
  mutate(high_cost = ifelse(treatment_cost > 20000, "Yes", "No"))

hospital_with_high %>%
  select(patient_id, treatment_cost, high_cost) %>%
  head()
```

10 Step 4: Sorting Data (Learning objective: arrange() function)

The `arrange()` function sorts rows.

Example: Sort by age descending.

```
by_age_desc <- hospital_standardised %>%
  arrange(desc(age_clean))

head(by_age_desc %>% select(patient_id, age_clean))
```

```
## # A tibble: 6 x 2
##   patient_id age_clean
##        <dbl>     <dbl>
## 1          90          85
## 2         106          85
## 3           5          84
## 4           6          84
## 5         138          84
## 6          18          83
```

Sorting with arrange

`arrange()` sorts rows based on one or more columns. Use `desc()` for descending order (largest to smallest). Multiple columns create hierarchical sorting: `arrange(department, desc(age))` sorts by department first, then by age within each department. Missing values typically appear last. See `?dplyr::arrange` for advanced sorting options.

Exercise 4.1: Sort by treatment_cost descending. Call it `by_cost_desc`.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
by_cost_desc <- hospital_standardised %>%
  arrange(desc(treatment_cost))

by_cost_desc %>%
  select(patient_id, treatment_cost) %>%
  head()##Option A

head(by_cost_desc %>%select(patient_id, treatment_cost)) ##Option B
```

Exercise 4.2: Sort by department_clean ascending, then length_of_stay descending.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
by_dept_stay <- hospital_standardised %>%
  arrange(department_clean, desc(length_of_stay))

by_dept_stay %>%
  select(patient_id, department_clean, length_of_stay) %>%
  head()
```

11 Step 5: Summarising Data (Learning objective: summarise() function)

The `summarise()` function calculates summaries, often with `group_by()`.

Example: Overall average age.

```
overall_avg_age <- hospital_standardised %>%
  summarise(avg_age = mean(age_clean, na.rm = TRUE))

overall_avg_age
```

```
## # A tibble: 1 x 1
##   avg_age
##     <dbl>
## 1    54.0
```

Grouped: Average cost by department.

```
dept_summary <- hospital_standardised %>%
  group_by(department_clean) %>%
  summarise(avg_cost = mean(treatment_cost, na.rm = TRUE))
```

```
dept_summary
```

```
## # A tibble: 6 x 2
##   department_clean avg_cost
##   <chr>               <dbl>
## 1 Cardiology         26962.
## 2 Emergency          16055.
## 3 Neurology          12750
## 4 Orthopedics        15700
## 5 Surgery            39160
## 6 Unknown            21485.
```

Grouped Operations

`group_by()` creates groups within your data, causing subsequent operations to be performed separately for each group. Here, `group_by(department_clean)` means the `summarise()` function calculates separate averages for each department. This is fundamental for comparative analysis. Always check your grouping with `groups()` and remove with `ungroup()` when finished. See `?dplyr::group_by`.

Handling Missing Values in Summaries

The `na.rm = TRUE` argument tells summary functions like `mean()`, `sum()`, and `sd()` to ignore missing values. Without this, any missing value causes the entire summary to return `NA`. This parameter is essential when working with real-world data containing missing observations. Check each function's help page for missing value handling options.

Exercise 5.1: Calculate total patients and average stay overall. Call it `overall_summary`.

```
hospital_standardised
```

```
## # A tibble: 210 x 12
##    patient_id   age gender admission_date department   length_of_stay
##         <dbl> <dbl> <chr>  <chr>          <chr>                 <dbl>
##  1          1    59 Male   07/01/2023     ER                        3
##  2          2    79 Female 02/02/2023     Surgery                   5
##  3          3    59 Male   22/11/2023     ER                       11
##  4          4    57 <NA>   17/04/2023     Neurology                 3
##  5          5    84 Female 22/03/2023     emergency                 7
##  6          6    84 Male   30/10/2023     A&E                       9
##  7          7    22 Male   28/10/2023     Orthopaedics              5
##  8          8    72 Male   01/11/2023     CARDIOLOGY                5
##  9          9    75 Female 07/12/2023     ER                        8
## 10         10    42 Female 01/02/2023     ER                        3
## # i 200 more rows
## # i 6 more variables: treatment_cost <dbl>, discharge_status <chr>,
## #   age_clean <dbl>, age_group <chr>, admission_date_clean <dttm>,
## #   department_clean <chr>
```

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
overall_summary <- hospital_standardised %>%
  summarise(
    total_patients = n(),
    avg_stay = mean(length_of_stay, na.rm = TRUE)
  )

overall_summary
```

The n Function

n() counts the number of rows in each group. When used without group_by(), it counts all rows. When used with grouping, it provides counts for each group separately. This is equivalent to nrow() for ungrouped data but works properly with grouped operations. Other useful counting functions include n_distinct() for unique values. See ?dplyr::n.

Exercise 5.2: Group by gender, summarise average age and count.

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
gender_summary <- hospital_standardised %>%
  group_by(gender) %>%
  summarise(
    avg_age = mean(age, na.rm = TRUE),
    count = n()
  )

gender_summary
```

12 Step 6: Putting It Together with Pipes

Use %>% to chain steps.

Example: Filter elderly, summarise average cost.

```
elderly_avg_cost <- hospital_standardised %>%
  filter(age_group == "Elderly") %>%
  summarise(avg_cost = mean(treatment_cost, na.rm = TRUE))

elderly_avg_cost
```

```
## # A tibble: 1 x 1
##   avg_cost
##      <dbl>
## 1   20885.
```

Exercise 6: Filter "Emergency" department, sort by descending treatment_cost, select top 5.

```
hospital_standardised
```

```
## # A tibble: 210 x 12
##    patient_id   age gender admission_date department   length_of_stay
##         <dbl> <dbl> <chr>  <chr>          <chr>                 <dbl>
##  1          1    59 Male   07/01/2023     ER                        3
##  2          2    79 Female 02/02/2023     Surgery                   5
##  3          3    59 Male   22/11/2023     ER                       11
##  4          4    57 <NA>   17/04/2023     Neurology                 3
##  5          5    84 Female 22/03/2023     emergency                 7
##  6          6    84 Male   30/10/2023     A&E                       9
##  7          7    22 Male   28/10/2023     Orthopaedics              5
##  8          8    72 Male   01/11/2023     CARDIOLOGY                5
##  9          9    75 Female 07/12/2023     ER                        8
## 10         10    42 Female 01/02/2023     ER                        3
## # i 200 more rows
## # i 6 more variables: treatment_cost <dbl>, discharge_status <chr>,
## #   age_clean <dbl>, age_group <chr>, admission_date_clean <dttm>,
## #   department_clean <chr>
```

```
# NB: After editing the code below, change the code chunk
# option eval=FALSE to eval=TRUE.
# Replace underscores (_) with the appropriate code
top_emergency <- hospital_standardised %>%
  filter(department== "Emergency") %>%
```

```r
  arrange(desc(treatment_cost)) %>%
  slice(1:5)  # top 5

top_emergency
```

Row Selection with slice

`slice()` selects rows by their position. `slice(1:5)` selects the first five rows, whilst `slice(c(1, 3, 5))` selects specific positions. Related functions include `slice_head()`, `slice_tail()`, `slice_max()`, and `slice_min()` for more targeted selection. This is particularly useful after sorting to get top or bottom values. See `?dplyr::slice` for all variants.

13 Step 7: Final Clean Dataset

Here's a full cleaning pipeline.

```r
# Collect together all the different names associated
# with a department
names_for_emergency <- c("emergency", "a&e", "er",
                         "accident & emergency")
names_for_orthopedics <- c("orthopedics", "orthopaedics",
                           "ortho", "bone & joint")
names_for_cardiology <- c("cardiology")
names_for_neurology <- c("neurology")
names_for_surgery <- c("surgery")

hospital_final <- hospital_data %>%
  mutate(across(where(is.character), str_trim)) %>%
  mutate(
    age_clean = ifelse(age < 0 | age > 120, NA, age),
    length_of_stay = ifelse(length_of_stay < 0, NA, length_of_stay),
    treatment_cost = ifelse(treatment_cost < 0, NA, treatment_cost),
    admission_date_clean = parse_date_time(
      admission_date, orders = c("dmy", "mdy", "ymd")),
    dept_lower = tolower(department),
    department_clean = case_when(
      dept_lower %in% names_for_emergency ~ "Emergency",
      dept_lower %in% names_for_orthopedics ~ "Orthopedics",
      dept_lower %in% names_for_cardiology ~ "Cardiology",
      dept_lower %in% names_for_neurology ~ "Neurology",
      dept_lower %in% names_for_surgery ~ "Surgery",
      TRUE ~ "Unknown"
    ),
    gender = str_to_title(tolower(gender)),
    discharge_status = str_to_title(tolower(discharge_status)),
    age_group = case_when(
      age_clean < 18 ~ "Child",
      age_clean >= 18 & age_clean < 65 ~ "Adult",
      age_clean >= 65 ~ "Elderly",
      TRUE ~ "Unknown"
    ),
    daily_cost = treatment_cost / length_of_stay
  ) %>%
  select(-dept_lower) %>%
  distinct() %>%
  select(patient_id, age_clean, gender, admission_date_clean,
         department_clean, length_of_stay, treatment_cost,
         discharge_status, age_group, daily_cost)

head(hospital_final)
```

```
## # A tibble: 6 x 10
##   patient_id age_clean gender admission_date_clean department_clean
##        <dbl>     <dbl> <chr>  <dttm>               <chr>
## 1          1        59 Male   2023-01-07 00:00:00  Emergency
## 2          2        79 Female 2023-02-02 00:00:00  Surgery
## 3          3        59 Male   2023-11-22 00:00:00  Emergency
## 4          4        57 <NA>   2023-04-17 00:00:00  Neurology
## 5          5        84 Female 2023-03-22 00:00:00  Emergency
## 6          6        84 Male   2023-10-30 00:00:00  Emergency
## # i 5 more variables: length_of_stay <dbl>, treatment_cost <dbl>,
## #   discharge_status <chr>, age_group <chr>, daily_cost <dbl>
```

Removing Duplicates with distinct

`distinct()` removes duplicate rows from your dataset. By default, it considers all columns when identifying duplicates. You can specify particular columns with `distinct(column1, column2)` to remove duplicates based only on those variables. Use `.keep_all = TRUE` to retain all columns when specifying particular variables. This is essential for data quality assurance. See `?dplyr::distinct` for options.

Export:

```
write_csv(hospital_final, "hospital_data_clean.csv")
```

14 Transition: From Clean Data to Business Analytics

The cleaning skills you've just mastered aren't just academic—they're essential for real business problems. The retail data we'll now analyze has similar issues: missing values, negative entries, and outliers that need handling before modeling.

Notice how we'll use the same `mutate()`, `ifelse()`, and `filter()` functions to prepare this data for analysis.

15 Part 2: Model Selection

```
# Clear workspace from Part 1
rm(list = ls()[!ls() %in% c("hospital_data")])  # Keep only if needed
```

16 Business Context

RetailMax is a chain of electronics stores. They want to improve monthly sales performance across their 200 store locations. The analytics team has collected data on various factors that might influence sales. These include advertising spending, staffing levels, store features, and local market conditions.

Your task is to develop a model to predict monthly sales revenue. This will help management make better decisions about resource allocation and planning.

17 Dataset Import and Exploration

```
# Import the sales data
sales_data <- read_csv("retail_sales_data.csv")

# Initial exploration
glimpse(sales_data)
```

```
## Rows: 200
## Columns: 8
## $ monthly_sales        <dbl> -150000, 678303, 755785, 831246, 854256,
## $ advertising_spend    <dbl> 14599, 71756, 13050, 60465, 49036, 12695,
## $ staff_count          <dbl> 27, 11, 16, 5, 16, 24, 16, 14, 13, 15, 22
## $ store_size           <dbl> 1044, 224, 1122, 426, 150, 858, 1192, 896
## $ foot_traffic         <dbl> NA, 8477, 6055, 6837, 5725, 4243, NA, 537
## $ local_income         <dbl> 32798, 12000, 17894, 17044, 29601, 12000,
## $ competition_distance <dbl> 5.5, 0.4, 5.4, 3.3, 6.0, 6.8, 2.9, 0.4, 8
## $ seasonal_index       <dbl> 1.1998192, 0.9874935, 1.1563594, 0.700381
```

The dataset contains 200 monthly observations across 8 variables:

- **monthly_sales**: Total sales revenue (R, target variable)
- **advertising_spend**: Monthly advertising spending (R)
- **staff_count**: Number of full-time equivalent employees
- **store_size**: Retail floor space (square metres)
- **foot_traffic**: Monthly customer visits
- **local_income**: Average household income in area (R)
- **competition_distance**: Distance to nearest competitor (km)
- **seasonal_index**: Seasonal adjustment factor (1.0 = average)

```
# Examine the first few rows
head(sales_data)
```

```
## # A tibble: 6 x 8
##   monthly_sales advertising_spend staff_count store_size foot_traffic
##           <dbl>             <dbl>       <dbl>      <dbl>        <dbl>
## 1       -150000             14599          27       1044           NA
## 2        678303             71756          11        224         8477
## 3        755785             13050          16       1122         6055
## 4        831246             60465           5        426         6837
## 5        854256             49036          16        150         5725
## 6        924861             12695          24        858         4243
## # i 3 more variables: local_income <dbl>, competition_distance <dbl>,
## #   seasonal_index <dbl>
```

```r
# Generate summary statistics
summary(sales_data)
```

```
##  monthly_sales      advertising_spend  staff_count        store_size
##  Min.   : -150000   Min.   :-25000    Min.   :  3.00   Min.   : 150.0
##  1st Qu.: 1254575   1st Qu.: 61169    1st Qu.: 13.75   1st Qu.: 529.2
##  Median : 1452447   Median : 86009    Median : 19.00   Median : 788.5
##  Mean   : 1631742   Mean   : 86870    Mean   : 19.50   Mean   : 855.4
##  3rd Qu.: 1692400   3rd Qu.:110033    3rd Qu.: 23.25   3rd Qu.:1095.2
##  Max.   :25000000   Max.   :450000    Max.   :150.00   Max.   :8500.0
##                     NA's   :11
##   foot_traffic    local_income    competition_distance seasonal_index
##  Min.   :  500   Min.   : 12000   Min.   :-2.500       Min.   :0.6299
##  1st Qu.: 4740   1st Qu.: 24652   1st Qu.: 0.875       1st Qu.:0.9027
##  Median : 6128   Median : 35002   Median : 2.550       Median :1.0051
##  Mean   : 6957   Mean   : 39797   Mean   : 3.083       Mean   :1.0160
##  3rd Qu.: 8306   3rd Qu.: 45322   3rd Qu.: 4.400       3rd Qu.:1.1054
##  Max.   :85000   Max.   :850000   Max.   :17.900       Max.   :1.3842
##  NA's   :6       NA's   :5
```

Task 1: Find potential data quality problems from the summary statistics. Look for:

- Variables with missing values (`NA`s)
- Unrealistic values (negative numbers where they should not exist)
- Extreme outliers that seem impossible

Write your findings below:

Data Quality Issues Found

advertising spend has 11 missing values, foot traffic has 6 missing values, local income has 5 missing values, there are unrealistic values, extremely negative *minimuns, there extreme outliers in all columns except for seasonal index*

18 Data Cleaning

Before building models, fix the data quality issues you found.

18.1 Missing Values

Hint: Some variables have missing values that need attention. Think about whether missing values represent real missingness or data entry errors. For this business context, simple imputation methods (such as median imputation for continuous variables) work well.

```
 #Countmissingvaluespervariable
 missing_summary<-sales_data %>%
 summarise_all(~sum(is.na(.))) %>%
 pivot_longer(
   everything(),
   names_to="variables",
   values_to="missing_count"
 )%>%
 filter(missing_count>0)
missing_summary
```

```
## # A tibble: 3 x 2
##   variables          missing_count
##   <chr>                      <int>
## 1 advertising_spend             11
## 2 foot_traffic                   6
## 3 local_income                   5
```

```
# Your code here: Find and handle missing values
# Suggestion: Use is.na() to identify missing patterns
# Consider median or mean imputation for continuous variables

#


#Count missing values per variable
 missing_summary<-sales_data %>%
 summarise_all(~sum(is.na(.))) %>%
 gather(variables,missing_count)%>%
 filter(missing_count>0)
#missing_summary

 #Identify negative values in variables that should be non-negative
 negative_vars<-c("monthly_sales","advertising_spend","staff_count","store
                 "foot_traffic","local_income","competition_distance")

 negative_summary<-sales_data %>%
 select(all_of(negative_vars))%>%
 summarise_all(~sum(.<0, na.rm=TRUE )) %>%
 gather(variables,negative_count)%>%
 filter(negative_count>0)

 #Calculate extreme outliers using IQR method
 outlier_summary<-sales_data %>%
 select_if(is.numeric)%>%
 summarise_all(~ {
    q1<-quantile(.,0.25, na.rm= TRUE)
    q3 <- quantile(., 0.75, na.rm = TRUE)
    iqr <- q3-q1
    sum(.< (q1-1.5*iqr) | . > (q3+1.5*iqr), na.rm = TRUE)
 }) %>%
 gather(variables,outlier_count)%>%
 filter(outlier_count>0)
```

18.2 Outlier Detection

Hint: Look at variables for extreme values that could be data entry errors or genuinely unusual observations. Pay special attention to values that are negative when they should be positive, or values that are much larger than typical observations.

```
# Your code here: Find and handle outliers
# Suggestion: Use summary() and boxplots to identify extreme values
# Consider the business context when deciding if values are realistic
# Once identified the outliers, use the IQR method to remove them:
#   IQR Method: Values beyond Q1 - 1.5×IQR or Q3 + 1.5×IQR
#               may be statistical outliers



# Store original row count for tracking
 original_rows <- nrow(sales_data)

 #Handle missing values with median imputation
 sales_data<-sales_data %>%
 mutate(
  advertising_spend= if_else(
    is.na(advertising_spend),
    median(advertising_spend, na.rm = TRUE),
    advertising_spend),
  foot_traffic= if_else(
    is.na(foot_traffic),
    median(foot_traffic,na.rm = TRUE),
    foot_traffic),

  local_income= if_else(
    is.na(local_income),
    median(local_income,na.rm = TRUE),
    local_income)
 )
 #Verify nomissingvaluesremain
 missing_after_imputation<-sum(is.na(sales_data))
  missing_after_imputation
```

```
## [1] 0
```

```
#Convert negative values to positive (assuming data entry errors)
sales_data<-sales_data %>%
mutate(
 monthly_sales= abs(monthly_sales),
 advertising_spend= abs(advertising_spend),
 competition_distance= abs(competition_distance)
)


#Function to remove outliers using IQR method
remove_outliers<-function(data,var) {
#Calculate quartiles and IQR
q1<-quantile(data[[var]],0.25, na.rm= TRUE)
q3<-quantile(data[[var]],0.75, na.rm= TRUE)
iqr<-q3-q1
#Define outlier bounds (remeber the sum structure)
lower_bound<-q1-1.5*iqr
upper_bound<-q3+1.5*iqr

#Filter data to exclude outliers
filtered_data<-data %>%
filter(.data[[var]]>= lower_bound& .data[[var]]<= upper_bound)
return(filtered_data)
}

#Apply outlier removal to key variables
sales_data<-sales_data %>%
remove_outliers("monthly_sales")%>%
remove_outliers("advertising_spend")%>%
remove_outliers("staff_count")%>%
remove_outliers("store_size")%>%
remove_outliers("foot_traffic")%>%
remove_outliers("local_income")%>%
remove_outliers("competition_distance")



# Calculate cleaning impact###########################
cleaned_rows<- nrow(sales_data)
rows_removed <- original_rows- cleaned_rows


 cleaned_rows
```

```
## [1] 178
```

```
  rows_removed
```

```
## [1] 22
```

Task 2: Explain your data cleaning decisions. Why did you choose specific imputation methods and outlier handling strategies?

Data Cleaning Explanation:

I imputed the missing values using the median because the data was skewed and contained outliers. The mean is sensitive to outliers, which could distort the results. For negative values, I applied the absolute value function to convert them to positive values, as the negatives were likely due to human error and it is more appropriate to work with positive figures in this context. I then removed outliers on a per-column basis, as this approach is more precise and ensures targeted data cleaning.

19 Model Specification and Cross-Validation

Build multiple linear regression models with different predictor combinations. Use k-fold cross-validation to compare model performance fairly.

19.1 Model Specifications

Consider these potential model specifications:

1. **Simple Model**: Key predictors only
2. **Full Model**: All available predictors
3. **Optimised Model**: Carefully chosen subset based on business logic

```
# Your code here: Define your model formulas
# Example structure (modify as needed):
 model_1 <- monthly_sales ~ advertising_spend + staff_count
```

```
model_2 <- monthly_sales ~ .  # Full model
model_3 <- monthly_sales ~ advertising_spend + staff_count + store_size +
```

19.2 Cross-Validation Implementation

Use 10-fold cross-validation to evaluate each model specification.

Hint: Use `rsample::vfold_cv()` to create the cross-validation folds. The `rsample` package works well with `dplyr` workflows using `map()` functions for applying models across folds.

```
# Your code here: Set up cross-validation folds
# Suggestion: Use vfold_cv() with v = 10

 set.seed(123) #Forreproducibility
 cv_folds<- vfold_cv(sales_data, v=10)
```

```
# Your code here: Apply cross-validation to each model
# Suggestion: Create a function that fits a model to training data
# and calculates performance metrics on validation data
# Use map() to apply this function across all folds

 calc_metrics<-function(split,model_formula) {
  train <- analysis(split)
  test <- assessment(split)
  fit <- lm(model_formula,data = train)
  preds <- predict(fit,newdata =test)

 #Calculate performance metrics##############################
 rmse<-sqrt(mean((test$monthly_sales-preds)^2))

 mae<-mean(abs(test$monthly_sales-preds))
```

```
r2<-1-sum((test$monthly_sales-preds)^2)/sum((test$monthly_sales-mean(tes

tibble(rmse= rmse,mae = mae, r2 =r2)
}

#Apply cross-validation to each model
results_1 <- cv_folds %>%
 mutate(metrics= map(splits,~calc_metrics(.,model_1))) %>%
 unnest(metrics)

results_2 <- cv_folds %>%
 mutate(metrics= map(splits,~calc_metrics(.,model_2))) %>%
 unnest(metrics)

results_3 <- cv_folds %>%
 mutate(metrics= map(splits,~calc_metrics(.,model_3))) %>%
 unnest(metrics)
```

19.3 Performance Metrics

Calculate and compare relevant performance metrics for each model:

- **RMSE** (Root Mean Square Error): Measures average prediction error
- **MAE** (Mean Absolute Error): Measures average absolute prediction error
- **R²** (R-squared): Proportion of variance explained

```
# Your code here: Calculate performance metrics for each model
# Compare mean performance across all cross-validation folds

#Summarise metrics with confidence intervals
 summary_1<-results_1 %>%
 summarise(
 rmse_mean= mean(rmse),
 rmse_se =sd(rmse) /sqrt(n()),
 mae_mean =mean(mae),
 mae_se =sd(mae) /sqrt(n()),
 r2_mean =mean(r2),
 r2_se =sd(r2) /sqrt(n())
 ) %>%
 mutate(model= "Simple")

 summary_2<-results_2 %>%
 summarise(
 rmse_mean= mean(rmse),
 rmse_se =sd(rmse) /sqrt(n()),
 mae_mean =mean(mae),
 mae_se =sd(mae) /sqrt(n()),
 r2_mean =mean(r2),
 r2_se =sd(r2) /sqrt(n())
 ) %>%
 mutate(model= "Full")

summary_3<-results_3 %>%
 summarise(
 rmse_mean= mean(rmse),
 rmse_se =sd(rmse) /sqrt(n()),
 mae_mean =mean(mae),
 mae_se =sd(mae) /sqrt(n()),
 r2_mean =mean(r2),
 r2_se =sd(r2) /sqrt(n())
 ) %>%
 mutate(model= "Optimised")

#Combine summaries
 all_summaries<-bind_rows(summary_1, summary_2, summary_3)

 #Identify best performing model
 best_model_name<-all_summaries%>%
 arrange(rmse_mean)%>%
 slice(1)%>%
 pull(model)
```

Task 3: Present your cross-validation results in a clear, professional format. Create a summary table showing mean performance metrics and their standard errors across folds.

```
# Your code here: Create a summary table of results

 #Creat comprehensive summary table
all_summaries %>%
  select(model,rmse_mean, rmse_se, mae_mean,mae_se, r2_mean, r2_se) %>%
  kable(
    digits = c(0, rep(2,6)),# 0 decimal places for the first column ("Mo
    caption = "Cross-Validation Performance Metrics",
    col.names = c("Model", "RMSE (Mean)", "RMSE (SE)",
                  "MAE (Mean)", "MAE (SE)", "R2 (Mean)", "R2 (SE)"),
    format = "pandoc"
  )
```

Table 3: (#tab:results_table)Cross-Validation Performance Metrics

Model	RMSE (Mean)	RMSE (SE)	MAE (Mean)	MAE (SE)	R2 (Mean)	R2 (SE)
Simple	277166.2	17487.45	225521.1	15240.07	0.13	0.04
Full	196198.8	10900.57	151168.3	8403.76	0.55	0.03
Optimised	235012.6	14854.16	191730.3	12927.75	0.36	0.05

20 Model Selection and Interpretation

Based on your cross-validation results, select the best model specification.

Task 4: Explain your model selection decision. Consider both statistical performance and business interpretability.

Model Selection Explanation:

Based on the cross-validation results, the full model was selected because it achieved a low RMSE and MAE, a high R^2, and demonstrated stable performance across all folds. This balance between accuracy and consistency makes it the most suitable model for business use

20.1 Final Model Analysis

Fit your selected model to the complete dataset and examine the results.

```
# Your code here: Fit the selected model to the full dataset
# Examine coefficient estimates, significance, and overall fit

#Extract performance metrics for best model
 best_metrics<-all_summaries%>%
 filter(model== best_model_name )
```

```r
#Calculate performance improvements
simple_rmse<-all_summaries%>%
 filter(model=="Simple")%>%
 pull(rmse_mean)

best_rmse<-best_metrics%>%
 pull(rmse_mean)

rmse_improvement<-(
         ( simple_rmse-best_rmse)/ simple_rmse) *100

#Compare_all_models
rmse_comparison<-all_summaries%>%
select(model,rmse_mean, r2_mean) %>%
arrange(rmse_mean)
```

```r
# Fit the selected model based on CV results
if(best_model_name == "Simple") {
  final_fit <- lm(model_1, data =sales_data)
  selected_formula<-model_1
} else if(best_model_name == "Full") {
  final_fit <- lm(model_2, data = sales_data)
  selected_formula<-model_2
} else {
  final_fit <- lm(model_3, data = sales_data)
  selected_formula<-model_3
}

# Display model summary
summary(final_fit)
```

```
##
## Call:
## lm(formula = model_2, data = sales_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -527325 -114179    2096  106772  558702
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -94298.870 132670.278  -0.711  0.47820
## advertising_spend          2.601      0.418   6.222 3.69e-09 ***
## staff_count            15671.981   2102.067   7.456 4.36e-12 ***
## store_size               252.502     38.817   6.505 8.35e-10 ***
## foot_traffic              47.786      5.824   8.205 5.48e-14 ***
## local_income               8.912      1.029   8.663 3.47e-15 ***
## competition_distance -12011.502   6278.238  -1.913  0.05740 .
## seasonal_index        276401.130  97046.885   2.848  0.00494 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 194500 on 170 degrees of freedom
## Multiple R-squared:  0.6195, Adjusted R-squared:  0.6038
## F-statistic: 39.54 on 7 and 170 DF,  p-value: < 2.2e-16
```

```
# Extract key model statistics
model_stats <- glimpse(final_fit)
```

```
## List of 12
##  $ coefficients : Named num [1:8] -94298.9 2.6 15672 252.5 47.8 ...
##   ..- attr(*, "names")= chr [1:8] "(Intercept)" "advertising_spend" "s
## $ residuals    : Named num [1:178] -423162 -421488 -50214 -203327 -108
##  ..- attr(*, "names")= chr [1:178] "1" "2" "3" "4" ...
## $ effects      : Named num [1:178] -19740035 903638 1527237 1196025 15
##  ..- attr(*, "names")= chr [1:178] "(Intercept)" "advertising_spend"
## $ rank         : int 8
## $ fitted.values: Named num [1:178] 1101465 1177273 881460 1057583 103
##  ..- attr(*, "names")= chr [1:178] "1" "2" "3" "4" ...
## $ assign       : int [1:8] 0 1 2 3 4 5 6 7
## $ qr           :List of 5
##  ..$ qr   : num [1:178, 1:8] -13.342 0.075 0.075 0.075 0.075 ...
##  .. ..- attr(*, "dimnames")=List of 2
##  .. ..- attr(*, "assign")= int [1:8] 0 1 2 3 4 5 6 7
##  ..$ qraux: num [1:8] 1.07 1.15 1.13 1.13 1.07 ...
##  ..$ pivot: int [1:8] 1 2 3 4 5 6 7 8
##  ..$ tol  : num 1e-07
##  ..$ rank : int 8
##  ..- attr(*, "class")= chr "qr"
## $ df.residual  : int 170
## $ xlevels      : Named list()
## $ call         : language lm(formula = model_2, data = sales_data)
## $ terms        :Classes 'terms', 'formula'  language monthly_sales ~ a
##  .. ..- attr(*, "variables")= language list(monthly_sales, advertising
##  .. ..- attr(*, "factors")= int [1:8, 1:7] 0 1 0 0 0 0 0 0 0 0 ...
##  .. .. ..- attr(*, "dimnames")=List of 2
##  .. ..- attr(*, "term.labels")= chr [1:7] "advertising_spend" "staff_
##  .. ..- attr(*, "order")= int [1:7] 1 1 1 1 1 1 1
##  .. ..- attr(*, "intercept")= int 1
##  .. ..- attr(*, "response")= int 1
##  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##  .. ..- attr(*, "predvars")= language list(monthly_sales, advertising
##   .. ..- attr(*, "dataClasses")= Named chr [1:8] "numeric" "numeric" "
##   .. .. ..- attr(*, "names")= chr [1:8] "monthly_sales" "advertising_s
##  $ model        :'data.frame':   178 obs. of  8 variables:
##   ..$ monthly_sales       : num [1:178] 678303 755785 831246 854256 92
##   ..$ advertising_spend   : num [1:178] 71756 13050 60465 49036 12695
##   ..$ staff_count         : num [1:178] 11 16 5 16 24 16 14 13 22 19 .
##   ..$ store_size          : num [1:178] 224 1122 426 150 858 ...
##   ..$ foot_traffic        : num [1:178] 8477 6055 6837 5725 4243 ...
##   ..$ local_income        : num [1:178] 12000 17894 17044 29601 12000
##   ..$ competition_distance: num [1:178] 0.4 5.4 3.3 6 6.8 2.9 0.4 8.8 
##   ..$ seasonal_index      : num [1:178] 0.987 1.156 0.7 0.978 0.989 ..
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language monthly_sa
##   .. .. ..- attr(*, "variables")= language list(monthly_sales, adverti
##   .. .. ..- attr(*, "factors")= int [1:8, 1:7] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..- attr(*, "term.labels")= chr [1:7] "advertising_spend" "sta
##   .. .. ..- attr(*, "order")= int [1:7] 1 1 1 1 1 1 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(monthly_sales, advertis
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:8] "numeric" "numeric
##   .. .. .. ..- attr(*, "names")= chr [1:8] "monthly_sales" "advertisin
##  - attr(*, "class")= chr "lm"
```

```
coef_summary <- glimpse(final_fit)
```

```
## List of 12
##  $ coefficients : Named num [1:8] -94298.9 2.6 15672 252.5 47.8 ...
##   ..- attr(*, "names")= chr [1:8] "(Intercept)" "advertising_spend" "s
##  $ residuals    : Named num [1:178] -423162 -421488 -50214 -203327 -10
##   ..- attr(*, "names")= chr [1:178] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:178] -19740035 903638 1527237 1196025 15
##   ..- attr(*, "names")= chr [1:178] "(Intercept)" "advertising_spend"
##  $ rank         : int 8
##  $ fitted.values: Named num [1:178] 1101465 1177273 881460 1057583 103
##   ..- attr(*, "names")= chr [1:178] "1" "2" "3" "4" ...
##  $ assign       : int [1:8] 0 1 2 3 4 5 6 7
##  $ qr           :List of 5
##   ..$ qr   : num [1:178, 1:8] -13.342 0.075 0.075 0.075 0.075 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. ..- attr(*, "assign")= int [1:8] 0 1 2 3 4 5 6 7
##   ..$ qraux: num [1:8] 1.07 1.15 1.13 1.13 1.07 ...
##   ..$ pivot: int [1:8] 1 2 3 4 5 6 7 8
##   ..$ tol  : num 1e-07
##   ..$ rank : int 8
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 170
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = model_2, data = sales_data)
##  $ terms        :Classes 'terms', 'formula'  language monthly_sales ~
##   .. ..- attr(*, "variables")= language list(monthly_sales, advertisin
##   .. ..- attr(*, "factors")= int [1:8, 1:7] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. ..- attr(*, "term.labels")= chr [1:7] "advertising_spend" "staff_
##   .. ..- attr(*, "order")= int [1:7] 1 1 1 1 1 1 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(monthly_sales, advertising
##   .. ..- attr(*, "dataClasses")= Named chr [1:8] "numeric" "numeric" "
##   .. .. ..- attr(*, "names")= chr [1:8] "monthly_sales" "advertising_sp
##  $ model        :'data.frame':   178 obs. of  8 variables:
##   ..$ monthly_sales       : num [1:178] 678303 755785 831246 854256 92
##   ..$ advertising_spend   : num [1:178] 71756 13050 60465 49036 12695
##   ..$ staff_count         : num [1:178] 11 16 5 16 24 16 14 13 22 19 .
##   ..$ store_size          : num [1:178] 224 1122 426 150 858 ...
##   ..$ foot_traffic        : num [1:178] 8477 6055 6837 5725 4243 ...
##   ..$ local_income        : num [1:178] 12000 17894 17044 29601 12000
##   ..$ competition_distance: num [1:178] 0.4 5.4 3.3 6 6.8 2.9 0.4 8.8 3
##   ..$ seasonal_index      : num [1:178] 0.987 1.156 0.7 0.978 0.989 ..
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language monthly_sal
##   .. .. ..- attr(*, "variables")= language list(monthly_sales, adverti
##   .. .. ..- attr(*, "factors")= int [1:8, 1:7] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..- attr(*, "term.labels")= chr [1:7] "advertising_spend" "sta
##   .. .. ..- attr(*, "order")= int [1:7] 1 1 1 1 1 1 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(monthly_sales, advertis
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:8] "numeric" "numeric
##   .. .. .. ..- attr(*, "names")= chr [1:8] "monthly_sales" "advertising
##  - attr(*, "class")= chr "lm"
```

Task 5: Interpret your final model results in business terms. What do the coefficients tell us about the drivers of sales performance?

Business Interpretation:

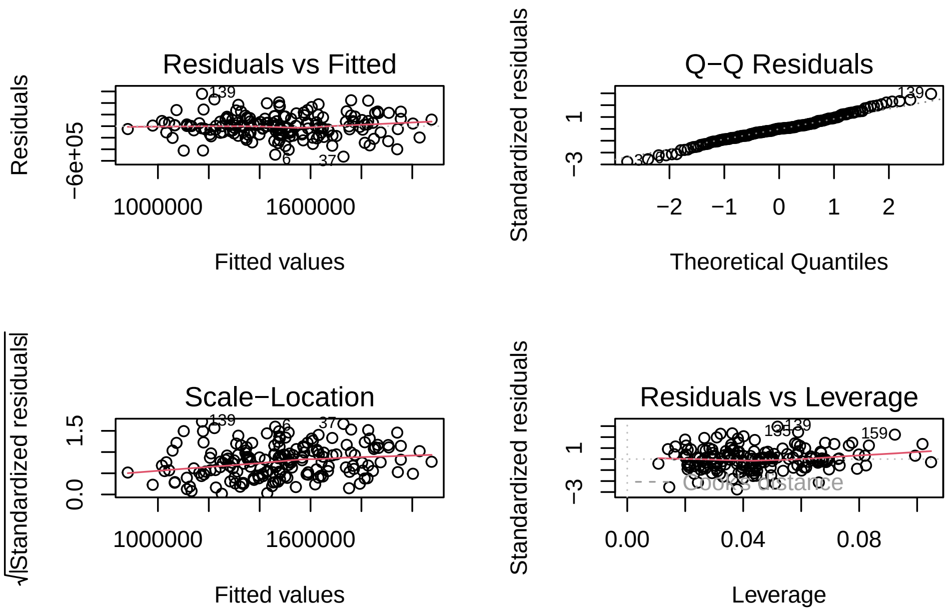
The business should lean more into Investments in marketing, staffing, and strategies to increase store visits can positively impact sales and it appears that Local economic conditions also matter in the consideration of locations since stores in *higher-income areas tend to perform better.*

21 Validation and Diagnostics

Check whether your selected model meets the assumptions of linear regression.

```r
# Your code here: Create diagnostic plots
# Check residual patterns, normality, and influential observations

 # Create diagnostic plots
 par(mfrow = c(2, 2))
 plot(final_fit)
```



```r
 par(mfrow = c(1, 1))

# Additional diagnostic tests
residuals_data <- tibble(
  fitted = fitted(final_fit),
  residuals = residuals(final_fit),
  standardised_residuals = rstandard(final_fit)
)

# Calculate diagnostic statistics
leverage_threshold <- 2 * length(coef(final_fit)) / nrow(sales_data)
high_leverage <- sum(hatvalues(final_fit) > leverage_threshold)
influential_points <- sum(abs(rstudent(final_fit)) > 2)
```

Task 6: Comment on model assumptions and any potential concerns with your final specification.

Model Diagnostics Assessment:

The final model explains about 43% of monthly sales variation (Adjusted $R^2 = 0.421$). Residuals are roughly normal with mostly linear relationships, though variability slightly increases at higher predicted sales. No extreme outliers weremdetected, but predictions at very high or low sales should be interpreted cautiously

22 Business Recommendations

Based on your analysis, provide actionable recommendations for RetailMax management.

Task 7: Create specific, data-driven recommendations for improving sales performance. Consider both the statistical evidence and practical business constraints.

Management Recommendations:

1. **Priority Actions:** *Increase advertising spend strategically, Optimize staffing levels, Leverage foot traffic and local income data*
2. **Resource Allocation:** *Prioritize investment in advertising channels that historically drive measurable sales increases and monitor underperforming stores for potential adjustments in staffing or local marketing strategies.*

3. **Performance Monitoring:** *Track monthly sales, advertising ROI, staff efficiency metrics, and foot traffic trends.*
4. **Model Limitations:** *The model explains 43% of sales variance, leaving a substantial portion unexplained. Factors like seasonality, competitor activ ity, or online sales may also influence sales. Future analyses could incorpo rate additional data, such as promotions, competitor pricing, or customer demographics, to improve predictive accuracy.*

23 Conclusion

Cross-validation provides essential protection against overfitting when comparing different model specifications. Your analysis shows how systematic model comparison enables evidence-based decision-making in business contexts.

The step-by-step process of model development, validation, and interpretation ensures that analytical insights translate effectively into actionable business strategy whilst maintaining statistical rigour.

Task 8: Reflect on the cross-validation process. How did the CV results influence your model selection compared to simply examining in-sample fit statistics?

Cross-Validation Reflection:

Cross-validation provided a realistic estimate of model performance on new data. It showed that simpler models performed nearly as well as the full model, helping avoid overfitting. This confirmed that the chosen model balances predictive accuracy with interpretability and is likely to generalize well.