Laboratory Exercise

Data Wrangling with Tidyverse

EPID634 - Population Health Dashboard

Learning Objectives

By the end of this laboratory session, students will be able to:

- • Import and export data using readr
- • Select, filter, and arrange data using dplyr
- • Create new variables and perform calculations with mutate
- • Summarize data with group_by and summarise
- • Reshape data between wide and long formats using tidyr
- • Join multiple datasets using various join operations
- • Chain operations efficiently using the pipe operator
- • Handle missing data and outliers appropriately

Required R Packages

Install and load the following packages before beginning:

```
# Install tidyverse (includes dplyr, tidyr, readr, tibble, etc.)

install.packages('tidyverse')

# Load the library

library(tidyverse)
```

The tidyverse includes these core packages:

- dplyr - data manipulation (select, filter, mutate, summarise)
- tidyr - data tidying (pivot, separate, unite)
- readr - data import/export (read_csv, write_csv)
- tibble - modern data frames
- stringr - string manipulation
- forcats - factor handling

PART 1: Data Import and Basic Operations

1.1 Reading and Exploring Data

The readr package provides fast and user-friendly functions for reading rectangular data.

Exercise 1.1: Create and Explore Sample Data

```
# Create a sample sales dataset

sales_data <- tibble(
```

```
  order_id = 1:100,
  customer_id = sample(1:20, 100, replace = TRUE),
  product = sample(c('Laptop', 'Phone', 'Tablet', 'Monitor', 'Keyboard'),
                   100, replace = TRUE),
  quantity = sample(1:5, 100, replace = TRUE),
  price = c(1200, 800, 500, 300, 50)[match(product,
                c('Laptop', 'Phone', 'Tablet', 'Monitor', 'Keyboard'))],
  date = sample(seq(as.Date('2024-01-01'), as.Date('2024-12-31'),
by='day'),
                100, replace = TRUE),
  region = sample(c('North', 'South', 'East', 'West'), 100, replace = TRUE)
)
# Explore the data
glimpse(sales_data)     # View structure
head(sales_data)        # First 6 rows
summary(sales_data)     # Statistical summary
```

Questions:

1. 1. How many rows and columns does the dataset contain?
2. 2. What data types are present in each column?
3. 3. Export this data to a CSV file and read it back in

PART 2: Selecting and Filtering Data

2.1 The Select Function

Use select() to choose specific columns from your dataset.

Exercise 2.1: Column Selection

```
# Select specific columns
sales_data %>%
  select(order_id, product, quantity, price)
# Select range of columns
sales_data %>%
  select(order_id:quantity)
# Exclude specific columns
sales_data %>%
  select(-customer_id, -date)
# Select columns by pattern
sales_data %>%
  select(starts_with('order'))
```

```
# Rename while selecting
sales_data %>%
  select(order = order_id, item = product, qty = quantity)
```

Practice Task:

4. Select only the columns related to product information (product, quantity, price)

2.2 The Filter Function

Use filter() to keep rows that meet specific conditions.

Exercise 2.2: Row Filtering

```
# Filter by single condition
sales_data %>%
  filter(product == 'Laptop')
# Filter by multiple conditions (AND)
sales_data %>%
  filter(product == 'Laptop' & quantity >= 3)
# Filter by multiple conditions (OR)
sales_data %>%
  filter(product == 'Laptop' | product == 'Phone')
# Filter using %in% for multiple values
sales_data %>%
  filter(product %in% c('Laptop', 'Phone', 'Tablet'))
# Filter by numeric range
sales_data %>%
  filter(between(price, 300, 800))
# Filter by string pattern
sales_data %>%
  filter(str_detect(region, 'North|South'))
```

Practice Tasks:

5. 1. Filter for orders with quantity greater than 2 in the North region
6. 2. Find all Laptop orders with a total value (price * quantity) over 2000

2.3 Arranging Data

Use arrange() to sort your data by one or more columns.

Exercise 2.3: Sorting Data

```
# Sort by single column (ascending)
sales_data %>%
  arrange(price)
```

```
# Sort descending
sales_data %>%
  arrange(desc(price))
# Sort by multiple columns
sales_data %>%
  arrange(region, desc(price))
```

Practice Task:

7. Sort the data by date (newest first), then by total value (highest first)

PART 3: Creating and Transforming Columns

3.1 The Mutate Function

Use mutate() to create new columns or modify existing ones.

Exercise 3.1: Creating New Variables

```
# Create a single new column
sales_data %>%
  mutate(total_value = price * quantity)
# Create multiple columns at once
sales_data %>%
  mutate(
    total_value = price * quantity,
    discount = total_value * 0.1,
    final_price = total_value - discount
  )
# Conditional mutations with case_when
sales_data %>%
  mutate(
    total_value = price * quantity,
    order_size = case_when(
      total_value < 500 ~ 'Small',
      total_value < 2000 ~ 'Medium',
      TRUE ~ 'Large'
    )
  )
# Conditional with if_else
sales_data %>%
  mutate(
```

```
    high_value = if_else(price >= 500, 'Yes', 'No')
  )
```

Practice Tasks:

8. 1. Create a column for tax (8% of total value)
9. 2. Create a priority column: High if total value > 1500, Medium if > 750, else Low
10. 3. Extract month and quarter from the date column

PART 4: Grouping and Summarizing Data

4.1 Group By and Summarise

Combine group_by() and summarise() to calculate aggregate statistics for groups.

Exercise 4.1: Aggregate Calculations

```
# Simple summary
sales_data %>%
  summarise(
    total_orders = n(),
    avg_price = mean(price),
    max_quantity = max(quantity)
  )
# Group by single variable
sales_data %>%
  group_by(product) %>%
  summarise(
    total_orders = n(),
    total_quantity = sum(quantity),
    avg_price = mean(price)
  )
# Group by multiple variables
sales_data %>%
  mutate(total_value = price * quantity) %>%
  group_by(region, product) %>%
  summarise(
    orders = n(),
    total_revenue = sum(total_value),
    avg_revenue = mean(total_value)
  ) %>%
  arrange(desc(total_revenue))
```

Practice Tasks:

11. 1. Calculate total revenue by region
12. 2. Find the product with highest average order value per region
13. 3. Calculate monthly sales trends

4.2 Window Functions with Mutate

Combine group_by() with mutate() to create group-wise calculations without collapsing rows.

Exercise 4.2: Group-wise Mutations

```
# Calculate percentage of total by group
sales_data %>%
  mutate(total_value = price * quantity) %>%
  group_by(product) %>%
  mutate(
    product_total = sum(total_value),
    pct_of_product = total_value / product_total * 100
  ) %>%
  ungroup()
# Rank within groups
sales_data %>%
  mutate(total_value = price * quantity) %>%
  group_by(region) %>%
  mutate(rank = row_number(desc(total_value))) %>%
  filter(rank <= 3) %>%
  ungroup()
```

Practice Task:

14. Calculate the deviation from the regional average price for each order

PART 5: Reshaping Data with Tidyr

5.1 Pivoting Data

Transform data between wide and long formats using pivot_longer() and pivot_wider().

Exercise 5.1: Wide to Long Format

```
# Create sample wide data
sales_wide <- tibble(
  region = c('North', 'South', 'East', 'West'),
  Q1 = c(1000, 1200, 950, 1100),
```

```
  Q2 = c(1100, 1300, 1050, 1200),
  Q3 = c(1250, 1400, 1100, 1300),
  Q4 = c(1350, 1500, 1200, 1400)
)
# Convert to long format
sales_long <- sales_wide %>%
  pivot_longer(
    cols = Q1:Q4,           # Columns to pivot
    names_to = 'quarter',   # New column for names
    values_to = 'sales'     # New column for values
  )
print(sales_long)
```

Exercise 5.2: Long to Wide Format

```
# Convert back to wide format
sales_wide_again <- sales_long %>%
  pivot_wider(
    names_from = quarter,   # Column with names
    values_from = sales     # Column with values
  )
print(sales_wide_again)
```

Practice Task:

15. Reshape the sales_data to show products as rows and regions as columns with total sales values

5.2 Separate and Unite

Split or combine columns using separate() and unite().

Exercise 5.3: Splitting and Combining Columns

```
# Create data with combined column
customer_data <- tibble(
  id = 1:5,
  name = c('John_Doe', 'Jane_Smith', 'Bob_Johnson',
           'Alice_Williams', 'Charlie_Brown')
)
# Separate into two columns
customer_data %>%
  separate(name,
           into = c('first_name', 'last_name'),
```

```
            sep = '_')
# Unite columns back together
customer_data %>%
  separate(name, into = c('first_name', 'last_name'), sep = '_') %>%
  unite(full_name,
        first_name, last_name,
        sep = ' ')
```

Practice Task:

16. Create a product_code column by combining product and region with a hyphen

PART 6: Joining Multiple Datasets

6.1 Types of Joins

Combine datasets using various join operations: left_join, right_join, inner_join, full_join.

Exercise 6.1: Basic Joins

```
# Create customer information dataset
customers <- tibble(
  customer_id = 1:25,
  customer_name = paste('Customer', 1:25),
  segment = sample(c('Enterprise', 'SMB', 'Startup'), 25, replace = TRUE),
  signup_date = sample(seq(as.Date('2023-01-01'),
                           as.Date('2024-12-31'), by='day'),
                       25, replace = TRUE)
)
# LEFT JOIN - keep all rows from left table
sales_with_customer <- sales_data %>%
  left_join(customers, by = 'customer_id')
# INNER JOIN - keep only matching rows
sales_inner <- sales_data %>%
  inner_join(customers, by = 'customer_id')
# FULL JOIN - keep all rows from both tables
all_data <- sales_data %>%
  full_join(customers, by = 'customer_id')
# Compare row counts
nrow(sales_data)           # Original
nrow(sales_with_customer)  # Left join
```

```
nrow(sales_inner)          # Inner join
nrow(all_data)             # Full join
```

Practice Tasks:

17. 1. Calculate average order value by customer segment
18. 2. Find customers who have never made a purchase (use anti_join)
19. 3. Identify which customer segment generates the most revenue

PART 7: Handling Missing Data

7.1 Detecting and Removing Missing Values

Exercise 7.1: Working with NAs

```
# Create data with missing values
data_with_na <- sales_data %>%
  mutate(
    price = if_else(row_number() %in% sample(1:100, 10), NA_real_, price),
    quantity = if_else(row_number() %in% sample(1:100, 8), NA_real_,
as.numeric(quantity))
  )
# Check for missing values
data_with_na %>%
  summarise(across(everything(), ~sum(is.na(.))))
# Remove rows with any NA
data_with_na %>%
  drop_na()
# Remove rows with NA in specific columns
data_with_na %>%
  drop_na(price, quantity)
# Replace NA with specific value
data_with_na %>%
  mutate(
    price = replace_na(price, median(price, na.rm = TRUE)),
    quantity = replace_na(quantity, mean(quantity, na.rm = TRUE))
  )
# Fill NA with previous/next value
data_with_na %>%
  arrange(date) %>%
  fill(price, .direction = 'down')  # Forward fill
```

Practice Task:

20. Identify which products have the most missing price values and impute them with product-specific medians

Comprehensive Exercise: Complete Analysis Pipeline

Challenge: Using all the techniques learned, perform a complete analysis of the sales data.

Requirements:

21. 1. Load and merge the sales_data with customer information
22. 2. Create calculated columns for total value, tax, and final price
23. 3. Clean the data by handling missing values appropriately
24. 4. Calculate summary statistics by product and region
25. 5. Identify the top 5 customers by total purchase value
26. 6. Create a monthly trend analysis showing revenue growth
27. 7. Reshape data to show products as rows and months as columns with total revenue
28. 8. Export final results to CSV files

Starter Code:

```
# Your analysis pipeline

final_analysis <- sales_data %>%

  # Step 1: Join with customer data

  left_join(customers, by = 'customer_id') %>%


  # Step 2: Create calculated columns

  mutate(

    # Your calculations here

  ) %>%


  # Continue with remaining steps...
```

Best Practices and Tips

Coding Style

- Use meaningful variable names that describe the data
- Keep pipelines readable: one operation per line when chaining
- Add comments to explain complex operations
- Save intermediate results for debugging and inspection
- Use consistent naming conventions (snake_case for variables)

Performance Tips

- Filter early in your pipeline to reduce data size
- Select only needed columns before heavy operations
- Use group_by() judiciously - ungroup() when done

- For large datasets, consider data.table or dtplyr backends

Common Pitfalls

- Forgetting to ungroup() after group operations
- Not handling NA values before calculations
- Overwriting original data - always assign to new variable
- Mixing up select() and filter() operations
- Not checking join results for unexpected duplicates

Useful Cheat Sheets

- dplyr cheat sheet: https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf
- tidyr cheat sheet: https://github.com/rstudio/cheatsheets/blob/main/tidyr.pdf
- Complete tidyverse documentation: https://www.tidyverse.org/