Lesson 11

by Lorraine Gaudio

Lesson generated on August 21, 2025

BOISE STATE UNIVERSITY

Contents

1. Welcome Back to R!

In lesson 8, 9, and 10, we explored several functions from dplyr (select(), filter(), mutate(), ifelse(), group_by and summarize()). You now know the key dplyr verbs one at a time. This week, we'll be putting dplyr verbs together in more complex pipelines. The real power comes from *chaining* them so data flows logically from raw to insight in a single readable statement.

To begin Lesson 11, follow these steps:

1. Open your course project for RStudio
2. Create a new file. From the file types we have used so far, pick which file type you want to use. (File > New File > ???).
3. Type in the code provided in this document as you follow along with the video. Pause the video at anytime to answer assignment questions, dig deeper or add memo notes.

Lesson Overview

By the end of Lesson 11 you will be able to:

1. Remember – List the actions of main dplyr verbs: select(), filter(), mutate(), group_by(), summarize(), arrange().
2. Understand – Describe how the pipe %>% passes results step by step.
3. Apply – Build a multi verb pipeline to clean and summarize data.
4. Analyze – Inspect intermediate output to verify each step.
5. Evaluate – Decide when a single pipeline is clearer than separate objects

Keep these goals in mind as you move through each section.

2. 📦 Packages

Install once (if needed): install.packages("dplyr"); install.packages("dslabs")

Load the packages at the start of every session:

```
library(dslabs)  # Data science labs package
library(dplyr)   # Data manipulation package
```

3. Warm-Up

 Review: The main dplyr verbs (functions) are select(), filter(), mutate(), group_by(), summarize(), arrange(). Type ?filter in the Console, then ?mutate, etc. Skim the usage sections.

 Comment: Which verb does what? For example, which verb changes *rows* and which changes *columns*?

4. ☒ Building a Pipeline

□ The GOAL: Find the average fuel consumption (liters/100 km) for cars in mtcars that use greater than 8 L/100 km, broken down by transmission and cylinders.

1. Start with the data

```
data("mtcars")

mtcars %>%                    # rows = 32 cars, many columns
  head(2)                     # preview first two rows

head(mtcars, 2)  # base R version, same output
```

2. Create a new variable with mutate()

```
mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  head(2)
```

□ Check□ in: Why divide 235.215 by mpg?

□ Check□ in: What does round() do here?

□ Check□ in: What does mutate() do here?

3. Keep only cars using greater than 8 L/100 km

□ NOTICE: Order Matters in dplyr Pipelines. You can't filter by l_100km before you create it with mutate().

```
mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  filter(l_100km >= 8) %>%
  head()
```

□ Check□ in: What is the filtering line doing?

4. Select relevant columns

```
mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  filter(l_100km >= 8) %>%
  select(cyl, am, l_100km) %>%
  head()
```

□ Check□ in: What does select() do here?

5. Group and summarize

```
mtcars_summary <- mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  filter(l_100km >= 8) %>%
  select(cyl, am, l_100km) %>%
  group_by(am, cyl) %>%
  summarize(mean_l_100km = round(mean(l_100km), 2),
            n = n(), .groups = "drop")
mtcars_summary
```

□ Explanation:

Order Matters in dplyr Pipelines. You can't summarize groups before you define them with group_by().

- group_by(am, cyl) tells dplyr to treat each unique combination of am and cyl separately. group_by() divides your data into separate groups for analysis. It doesn't change how your data looks. It adds metadata that tells **subsequent functions** to operate on each group separately. It's like telling R: "For each combination of these variables, do the following…"
- summarize() then runs mean(l_100km) *per group* and counts rows with n().
- .groups = "drop" prevents grouping in the final output.

□ Practice: How might you check which combination of am + cyl is least fuel□ efficient?

6. Recode transmission for readability

Method 1:

```
# Recode transmission for readability Method 1
mtcars_summary %>%
  mutate(am = ifelse(am == 0, "Automatic", "Manual"))
```

```
# □ Pipeline complete! One readable flow from raw data to insight.

mtcars_summary <- mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  filter(l_100km >= 8) %>%
  select(cyl, am, l_100km) %>%
```

```
  group_by(am, cyl) %>%
  summarize(mean_l_100km = round(mean(l_100km), 2),
            n = n(), .groups = "drop")%>%
  mutate(am = ifelse(am == 0, "Automatic", "Manual"))
mtcars_summary
```

□ Reflect: Which combination of am + cyl is least fuel□ efficient?

□ Look deeper: What other methods check which combination of am + cyl is least fuel□ efficient?

Method 2:

```
# Look for the highest mean_l_100km value)
max(mtcars_summary$mean_l_100km)  # Find the highest mean_l_100km
```

Method 3:

Pipe in arrange() to sort by mean_l_100km descending.

```
mtcars_summary2 <- mtcars %>%
  mutate(l_100km = round(235.215 / mpg, 1)) %>%
  filter(l_100km >= 8) %>%
  select(cyl, am, l_100km) %>%
  group_by(am, cyl) %>%
  summarize(mean_l_100km = round(mean(l_100km), 2),
            n = n(), .groups = "drop") %>%
  arrange(desc(mean_l_100km)) %>% # Sort by fuel consumption (highest first)
  head(1) # The first row will be the least efficient combination
```

□ NOTICE: We rename to mtcars_summary2 instead of mtcars_summary

□ Explanation: arrange() sorts the data frame by mean_l_100km in descending order, so the first row is the least fuel-efficient combination of am and cyl.

□ Compare mtcars_summary with mtcars_summary2.

```
mtcars_summary2
```

5. Pipe with **unique()**

 Sometimes a base R function (like unique()) is the simplest tool. Pipes work with base R functions.

```
Movie_Night <- movielens %>%
  filter(rating >= 4.5) %>%
  select(title) %>%
  unique()
head(Movie_Night)
```

 Check in: What does unique() do here?

6. Multiple Columns, One **mutate()**

```
mtcars %>%
  mutate(l_100km     = round(235.215 / mpg, 2),
         l_per_km    = l_100km / 100,
         l_per_meter = l_per_km / 1000) %>%
  select(mpg, l_100km:l_per_meter) %>%
  head()
```

 Order Matters in dplyr Pipelines: later columns can use ones created earlier in the same mutate().

7. ⛶ Practice Space

□ Practice: Using the built□ in CO2 dataset, calculate the mean uptake for each treatment (chilled vs. non□ chilled) within each type. Add a mutate() step that labels mean_uptake greater than 30 as "High" and less then or equal too 30 as "Low".

```
data("CO2")
?CO2
```

Fill in the Blanks

```
CO2 %>%
 ___(Type, Treatment) %>%
 ___(mean_uptake = mean(uptake))
 ___(uptake_level = ifelse(mean_uptake > 30, "High", "Low"))
```

□ Practice: Use this sandbox to design your own pipeline.

□ Ideas:

- Choose a dataset from dslabs (e.g., heights, gapminder, murders).
- Write a pipeline that filters rows, calculates a new metric, groups, summarizes, and arranges the result.
- Add at least one □ check□ in comment to predict an outcome before you run.

8. ☒ Assignment

Replace each ____ placeholder (and any TODO comments) with working code or a short written answer. Run each section; be sure the requested objects appear in the Environment. When finished, save **BOTH** this script and your .RData workspace and upload.

When you're done, your workspace should contain FOUR new objects: mpg_pipeline, Movie_Facts, Tall_Cars, Euro_07

8.1 Task 1

☐ Library it up!

Make sure there is script in your document to load dplyr and dslabs packages so their functions / datasets load.

8.2 Task 2

☐ Quick Recall

Explain how *three* dplyr verbs from Lessons 7-11 work.

☐ EXPLANATION: "___"

☐ EXPLANATION: "___"

☐ EXPLANATION: "___"

8.3 Task 3

☐ Pipeline

Build **mpg_pipeline** from mtcars:

- add l_100km (235.215/mpg)
- retain rows where l_100km is ≥ 8
- create a new data frame that returns one row for each combination of grouping by cyl and provides the summary statistic for mean_l_100km, rounding to 1 decimal.
- order the rows of the new data frame by descending mean_l_100km.

```
____ <- _____
```

```
# Quick check
glimpse(mpg_pipeline)
```

8.4 Task 4

 Intermediate Peek

Add ONE line of code below that pipes mpg_pipeline into head() so you can inspect the first rows

```
____ <- _____
```

8.5 Task 5

 Multi-step Challenge

Using movielens (dslabs) create **Movie_Facts**:

- keep title, year, rating
- mutate decade = floor(year/10)*10
- create a new data frame that returns one row for each combination of grouping by decade and provides the summary statistic for avg_rating (mean) and n_movies (n)
- retain rows where n_movies ≥ 200

```
____ <- _____
```

8.6 Task 6

 Logical filter + mutate

With mtcars again, make **Tall_Cars** that keeps cars with:

- hp > 150 OR qsec < 16
- then mutate power_to_weight equal to hp/wt
- Keep the model name, hp, wt,power_to_weight.

```
____ <- _____
```

8.7 Task 7

Apply on gapminder

Use gapminder (dslabs) to create **Euro_07**:

- retain rows where continent == "Europe", year == 2007
- pick columns country, life_expectancy, gdp
- create a new column called gdp_billions that rounds gdp/1e9 to 1.

```
____ <- _____
```

8.8 Task 8

 Reflect

 Write a short paragraph reflecting on when might **separate objects** be clearer than a single long pipeline?

 EXPLANATION: "___"

9. Save and Upload

1. You will be submitting **both** the Quarto Document and the workspace file. The workspace file saves all the objects in your environment that you created in this lesson. You can save the workspace by running the following command in a code chunk of the Quarto Document document:

```
save.image("Assignment11_Workspace.RData")
```

Or you can click the "Save Workspace" button in the Environment pane.

 Always save the R documents before closing.

2. Find the assignment in this week's module in Canvas and upload **both** the RMD and the workspace file.

10. Today you practiced:

- Connected dplyr verbs with the pipe to form readable workflows.
- Built a pipeline that mutated, filtered, selected, grouped, summarized, and recoded in one statement.
- Used a base R function (unique) inside a pipe.
- Created multiple new variables in a single mutate().
- Reflected on the order of steps