# Homework 3

Solutions

2026-01-08

### Introduction

In this assignment you will practice (1) choosing appropriate plots based on variable type and analytic goal, and (2) using `dplyr` pipelines to move from raw data → transformed data → a final plot.

You will use two data sets:

- `ncbirths` (from the `openintro` package)
- `flights` (from the `nycflights13` package)

Use the following code chunk to load the necessary packages (`tidyverse`, `ggpubr` and `sjPlot`), and load the data listed above.

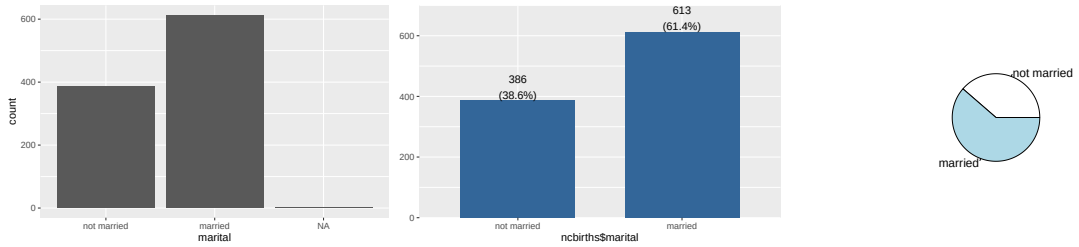
```
library(tidyverse)
library(ggpubr)
library(sjPlot)
library(gtsummary)
flights <- nycflights13::flights
ncbirths <- openintro::ncbirths
```

## Part I: Creating plots (ncbirths)

1. Create appropriate visualizations for the `marital` status, and the mothers age (`mage`).

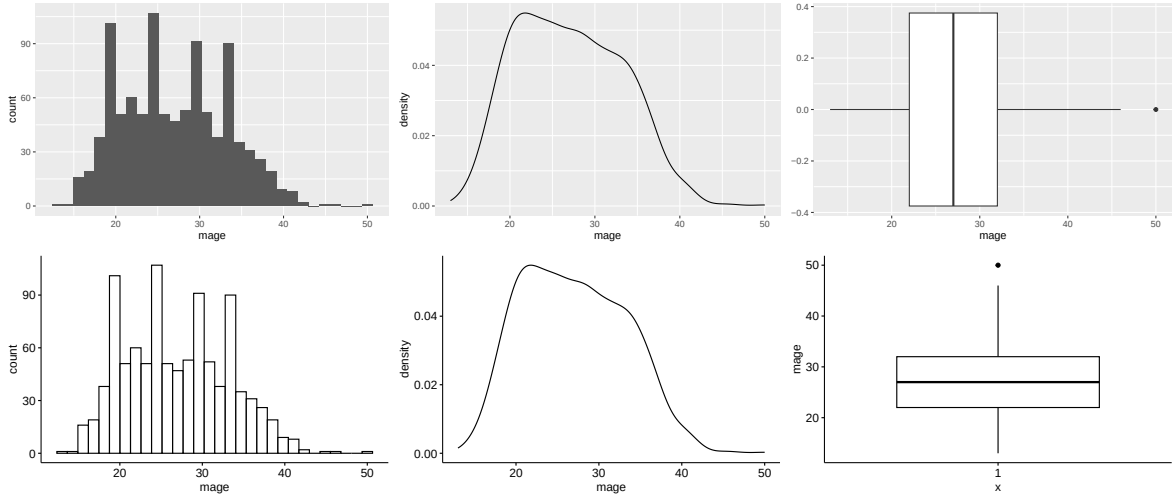
a. `marital` status

```
ggplot(ncbirths, aes(x = marital)) + geom_bar()
plot_frq(ncbirths$marital)
table(ncbirths$marital) |> pie()
```




b. Mothers age (`mage`)

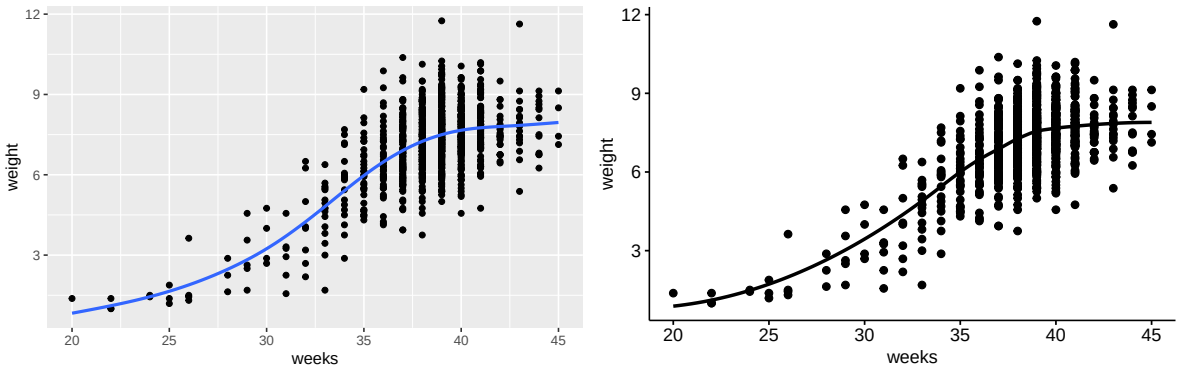
```
ggplot(ncbirths, aes(x = mage)) + geom_histogram()
ggplot(ncbirths, aes(x = mage)) + geom_density()
ggplot(ncbirths, aes(x = mage)) + geom_boxplot()
gghistogram(ncbirths, x="mage")
ggdensity(ncbirths, x="mage")
ggboxplot(ncbirths, y="mage")
```




2. Bivariate Relationships

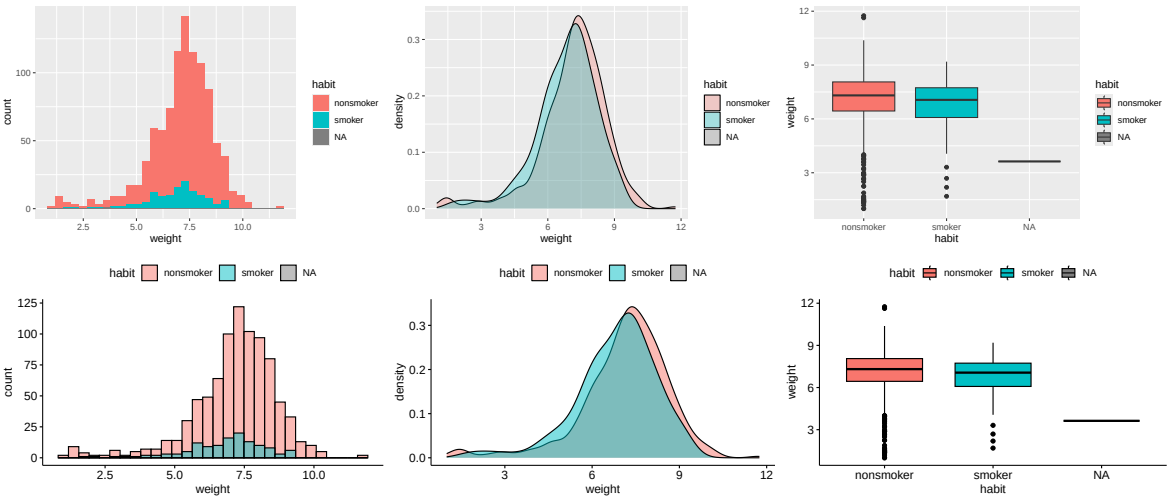
a. Is birth weight related to length of pregnancy? Create a scatterplot of **`weight`** versus **`weeks`**. Add **one** smoother to your plot (either a loess smoother *or* a linear model line).

```
ggplot(ncbirths, aes(x = weeks, y = weight)) +
  geom_point() + geom_smooth(se = FALSE)
ggscatter(ncbirths, x="weeks", y="weight", add = "loess")
```




b. Do babies of smokers tend to have different birth weights than babies of non-smokers? Create a plot that compares the distribution of `weight` across levels of `habit`.

```
ggplot(ncbirths, aes(x = weight, fill = habit)) + geom_histogram()
ggplot(ncbirths, aes(x = weight, fill = habit)) + geom_density(alpha = .3)
ggplot(ncbirths, aes(x = habit, y = weight, fill = habit)) +
  geom_boxplot()
gghistogram(ncbirths, x="weight", fill = "habit")
ggdensity(ncbirths, x="weight", fill = "habit")
ggboxplot(ncbirths, x = "habit", y = "weight", fill = "habit")
```




c. Does smoking habit differ by mothers maturity status? Create a table to display the distribution of smoking `habit` by the mothers maturity status (`mature`).

| **Characteristic** | **nonsmoker** N = 873[1] | **smoker** N = 126[1] |
|---|---|---|
| mature | | |
| mature mom | 121 (92%) | 11 (8.3%) |
| younger mom | 752 (87%) | 115 (13%) |

[1] n (%)

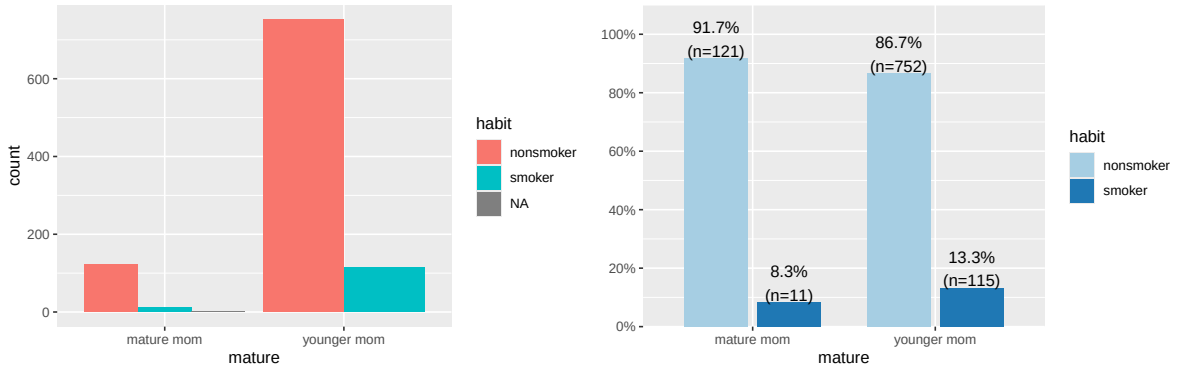
```
table(ncbirths$mature, ncbirths$habit) |> prop.table(margin=1)  |> round(3)
```

```
              nonsmoker smoker
  mature mom      0.917  0.083
  younger mom     0.867  0.133
```

```
tbl_summary(ncbirths, include = "mature", by = "habit", percent = "row")
```

d. Create a side by side bar chart that visually reflects the comparison above.

```
ggplot(ncbirths, aes(x=mature, fill=habit)) + geom_bar(position = "dodge")
plot_xtab(x = ncbirths$mature, grp = ncbirths$habit, show.total = "false",
          margin = "row")
```




## Part II: Efficient Data Management → Plot (NYC Flights)

Each question starts with the full `flights` data set and will use multiple dplyr verbs so chaining is advised. Your final result should be shown as a printed tibble.

1. Create a new data set that contains only flights that:

- departed from **JFK**
- traveled at least **1000 miles**
- have non-missing `air_time`

Keep only the variables `origin`, `dest`, `distance`, and `air_time`. Then create a new variable called `speed` defined as `distance / air_time * 60`. Finally, sort from fastest to slowest `speed` and display the first 10 rows.

```
flights %>%
  filter(origin == "JFK", distance >= 1000, !is.na(air_time)) %>%
  select(origin, dest, distance, air_time) %>%
  mutate(speed = distance / air_time * 60) %>%
  arrange(desc(speed)) %>%
  head(10)
```

```
# A tibble: 10 x 5
   origin dest  distance air_time speed
   <chr>  <chr>    <dbl>    <dbl> <dbl>
 1 JFK    SJU       1598      170  564
 2 JFK    SJU       1598      172  557.
 3 JFK    STT       1623      175  556.
 4 JFK    SJU       1598      173  554.
 5 JFK    SJU       1598      173  554.
 6 JFK    SJU       1598      173  554.
 7 JFK    SJU       1598      173  554.
 8 JFK    SJU       1598      173  554.
 9 JFK    SJU       1598      173  554.
10 JFK    SJU       1598      173  554.
```

2. What are the most popular destinations for each airport?

> This question is broken down into a few smaller parts to help you ensure that you are on the right track before moving to the next step. Only add one command at a time and "trust but verify" to make sure your code works as intended. You do not need to report the results of each step, just the final result.

a. Use `filter()` to keep only flights with a non-missing distance, then `group_by(origin, dest)` to group the data by origin first, then destination within origin.
b. Use `summarise()` to calculate, for each origin–destination pair:

- the number of flights as `n_flights`

- the average distance as `avg_distance`

c. Keep only the top 3 destination using `slice_head(n=3)` after sorting by the number of flights.

d. Use `relocate()` so the columns appear in the order: `origin`, `avg_distance`, `dest`, n_flights‘, and display the resulting table.

```
flights %>%
  filter(!is.na(distance)) %>%
  group_by(origin, dest) %>%
  summarise(
    n_flights = n(),
    avg_distance = mean(distance)) %>%
  arrange(desc(n_flights)) %>%
  slice_head(n=3) %>%
  relocate(c(dest, n_flights), .after = last_col())
```

```
# A tibble: 9 x 4
# Groups:   origin [3]
  origin avg_distance dest  n_flights
  <chr>         <dbl> <chr>     <int>
1 EWR             719 ORD        6100
2 EWR             200 BOS        5327
3 EWR            2565 SFO        5127
4 JFK            2475 LAX       11262
5 JFK            2586 SFO        8204
6 JFK             187 BOS        5898
7 LGA             762 ATL       10263
8 LGA             733 ORD        8857
9 LGA             544 CLT        6168
```

3. Exploring travel delays

Typically we don’t care if a departure delay is a few minutes late, but anything longer could be detrimental to connecting flights. Use `case_when` to bin the `dep_delay` variable into categories based on departure delay status.

- Create **at least three categories**
- Choose and briefly describe your cutoff values
- Use category names that correspond to intuitive delay types (e.g., “on time”, “short,” “medium,” “long”).
- A reasonable approach is to look at the distribution of the departure delay first (histogram), then choose cutoffs based on where the distribution naturally changes.

| **Characteristic** | **N = 336,776**[1] |
|---|---|
| delay_category | |
| long delay | 48,291 (15%) |
| on time or early | 200,089 (61%) |
| short delay | 80,141 (24%) |
| Unknown | 8,255 |

[1] n (%)

Display a frequency table showing how many flights fall into each delay category.

```
flights %>%
  mutate(delay_category = case_when(
    dep_delay <= 0 ~ "on time or early",
    dep_delay > 0 & dep_delay <= 30 ~ "short delay",
    dep_delay > 30 ~ "long delay"
  )) %>%
tbl_summary(include = delay_category)
```