Resping Data with tidyr

Saul Diaz Infante Velasco

3/16/23

Table of contents

Preface

Preface for this part

Introduction

Data in the wild can be scary—when confronted with a complicated and messy dataset you may find yourself wondering, where do I even start? The `tidyr` package allows you to wrangle such beasts into nice and tidy datasets. Inaccessible values stored in column names will be put into rows, JSON files will become data frames, and missing values will never go missing again. You'll practice these techniques on a wide range of messy datasets, learning along the way how many dogs the Soviet Union sent into space and what bird is most popular in New Zealand. With the tidyr package in your tidyverse toolkit, you'll be able to transform almost any dataset in a tidy format which will pay-off during the rest of your analysis.

1 Tidy Data

You'll be introduced to the concept of tidy data which is central to this course. In the first two lessons, you'll jump straight into the action by separating messy character columns into tidy variables and observations ready for analysis. In the final lesson, you'll learn how to overwrite and remove missing values.

1.1 Multiple variables per column

Being a busy person, you don't want to spend too much time on Netflix, so you decide to crunch some numbers on TV show and movie durations before deciding what to watch. You've managed to obtain a dataset named `netflix_df`, but its `duration` column has an issue. It contains strings with both a value and unit of duration (`"min"` or `"Season"`).

You'll tidy this dataset so that each variable gets its own column.

As will always be the case in this course, load the tidyr package.

Instructions `100 XP`

- Inspect `netflix_df` by typing its name directly in the R console and hitting Enter to see what string separates the value from the unit in the `duration` column.
- Separate the `duration` column over two variables named `value` and `unit`. Pass the string separating the number from the unit to the `sep` argument.

ex_001.R

```
netflix_df %>%
  # Split the duration column into value and unit columns
  separate(duration, into =c("value","unit"),sep = " ", convert = TRUE)
```

1.2 International phone numbers

You work for a multinational company that uses auto-dialer software to contact its customers. When new customers subscribe online they are asked for a phone number but they often forget to add the country code needed for international calls. You were asked to fix this issue in the database. You've been given a data frame with national numbers and country codes named `phone_nr_df`. Now you want to combine the country_code and `national_number` columns to create valid international numbers.

Instructions `100 XP`

Use the `unite()` function to create a new `international_number` column, using an empty string as the separator.

ex_002.R

```
phone_nr_df %>%
  # Unite the country_code and national_number columns
  unite(
    "international_number",
    country_code,
    national_number,
    sep=""
  )
```

1.3 Extracting observations from values

Extracting observations from values You're given a sample of the Netflix dataset containing TV shows and their casts called `tvshow_df`. You want to learn which six actors have the most appearances.

However, the dataset only has one row per TV show, and multiple actors are listed in the cast column.

Transform the data so that for each TV show, every actor has a row. The number of appearances will be calculated for you.

Load dplyr package.

Instructions 100 XP

- Use `separate_rows()` on the cast column, using the appropriate separator for the sep argument.
- Use the `head()` function to keep just the top six.

ex_003.R

```
tvshow_df %>%
  # Separate the actors in the cast column over multiple rows
  separate_rows(cast, sep=", ") %>%
  rename(actor = cast) %>%
  count(actor, sort = TRUE) %>%
  head()
```

1.4 Separating into columns and rows

Remember the drink ingredients data from the video? You've been given a similar version (`drink_df`) that also includes quantities and units. Now you want to create an overview of how much of each ingredient you should buy to make these drinks.

Load `dplyr`.

Instructions 100 XP

- Inspect `drink_df` in the console to find the right separator in the `ingredients` column.
- Separate the `ingredients` column so that for each drink each ingredient gets a row. -Inspect the output of the previous step to find the separator that splits the ingredients column into three columns: `ingredient`, `quantity`, and `unit`. -Make sure to convert data types to numeric when possible.

ex_004.R

```
drink_df %>%
  # Separate the ingredients over rows
  separate_rows(ingredients, sep = "; ") %>%
  # Separate ingredients into three columns
  separate(
    ingredients,
    into = c("ingredient", "quantity", "unit"),
    sep = " ",
```

```
    convert = TRUE
  ) %>%
  # Group by ingredient and unit
  group_by(ingredient, unit) %>%
  # Calculate the total quantity of each ingredient
  summarize(quantity = sum(quantity))
```

1.5 And the Oscar tfor best director goet to ..

You're working on a sample of the Netflix dataset pre-loaded as `director_df`. This time, the data frame contains just the directors and movie titles. Your goal is to identify the directors who created the most movies. Since the `director` column contains multiple names, you'll first separate its values over multiple rows and then count the directors.

Since you don't want movies without directors polluting your overview, you'll apply the `drop_na()` function.

Load dplyr package.

Instructions `100 XP`

- Inspect `director_df` in the console to see what string separates directors in the `director` column.
- Spread the values in the `director` column over separate rows.
- Count the number of times each director appears in the data. Make sure to sort the output.
- Drop rows containing NA values in the director column.

ex_005.R

```
director_df %>%
  # Spread the director column over separate rows
  separate_rows(
    director,
    sep = ", "
  )

director_df %>%
  # Spread the director column over separate rows
  separate_rows(director, sep = ", ") %>%
```

```
  # Count the number of movies per director
  count(director)

director_df %>%
  # Spread the director column over separate rows
  separate_rows(director, sep = ", ") %>%
  # Count the number of movies per director
  count(director, sort=TRUE)

director_df %>%
  # Drop rows with NA values in the director column
  drop_na(director) %>%
  # Spread the director column over separate rows
  separate_rows(director, sep = ", ") %>%
  # Count the number of movies per director
  count(director, sort = TRUE)
```

1.6 Imputing sales data

You've been asked to create a report that allows management to compare sales figures per quarter for two years. The problem is that the dataset (`sales_df`) contains missing values. You'll need to impute the values in the year column so that you can visualize the data.

Load `ggplot2`.

Instructions `100 XP`

- Inspect `sales_df` in the console, pay attention to the year column.
- Use the `fill()` function to impute the year column in the correct direction.
- Create a line `plot` where each year has a different color.

ex_006.R

```
sales_df %>%
  # Impute the year column
  fill(year, .direction = "up") %>%
  # Create a line plot with sales per quarter colored by year.
  ggplot(
    aes(
      x = quarter,
```

```
      y = sales,
      color = year,
      group = year
    )
  ) +
  geom_line()
```

1.7 Nuclear bombs per continent

Since WWII, a number of nations have been detonating nuclear bombs for military research. A tally of bombs detonated per nation has been calculated from the Nuclear Explosion DataBase (NEDB) and provided as `nuke_df`. You are interested in finding out how many bombs have been detonated by nations grouped per continent. To achieve this goal, `nuke_df` will be joined to `country_to_continent_df` which is a mapping of nation to continent. You will need to overwrite missing values with zeros so that you can create a nice plot.

Load `dplyr` and `ggplot2`.

Side note 1: Bombs detonated by the Soviet Union were attributed to the Russian Federation.
Side note 2: The Russian Federation is solely mapped to Europe for simplicity.

Instructions `100 XP`

- Inspect `nuke_df` and `country_to_continent_df` in the console.
- Replace the missing values in the `n_bombs` columns with `0L`. Adding the L sets the data type to integer.
- Group the dataset by `continent` and aggregate the data by summing the number of bombs.
- Plot the summed number of bombs detonated by nations from each continent.

ex_007.R

```
country_to_continent_df %>%
left_join(nuke_df, by = "country_code") %>%
# Impute the missing values in the n_bombs column with 0L
replace_na(list(n_bombs = 0L)) %>%
# Group the dataset by continent
group_by(continent) %>%
# Sum the number of bombs per continent
summarize(n_bombs_continent = sum(n_bombs)) %>%
```

```r
# Plot the number of bombs per continent
ggplot(aes(x = continent, y = n_bombs_continent)) +
geom_col()
```

2 From Wide to Long and Back

This chapter is all about pivoting data from a wide to long format and back again using the `pivot_longer()` and `pivot_wider()` functions. You'll need these functions when variables are hidden in messy column names or when variables are stored in rows instead of columns. You'll learn about space dogs, nuclear bombs, and planet temperatures along the way.

2.1 Nuclear bombs per country

You've been given a version of the Nuclear Explosion DataBase (NEDB) where country names are specified in the column headers (nuke_df). You want to visualize how many nukes were detonated per year per country. You'll need to pivot the data and replace NA values first.

The ggplot2 package is needed.

Instructions `100 XP`

Pivot all columns except for year to a longer format.

ex_008.R

```
nuke_df %>%
  # Pivot the data to a longer format
  pivot_longer(-"year")

nuke_df %>%
  # Pivot the data to a longer format
  pivot_longer(
    -year,
    # Overwrite the names of the two new columns
    names_to = "country",
    values_to = "n_bombs"
  )

nuke_df %>%
```

```r
# Pivot the data to a longer format
pivot_longer(
  -year,
  # Overwrite the names of the two new columns
  names_to = "country",
  values_to = "n_bombs"
) %>%
# Replace NA values for n_bombs with 0L
replace_na(list(n_bombs = 0L))

nuke_df %>%
# Pivot the data to a longer format
pivot_longer(
  -year,
  # Overwrite the names of the two new columns
  names_to = "country",
  values_to = "n_bombs"
) %>%
# Replace NA values for n_bombs with 0L
replace_na(list(n_bombs = 0L)) %>%
# Plot the number of bombs per country over time
ggplot(aes(x=year, y=n_bombs, group=country, color=country)) +
  geom_line()
```

2.2 WHO obesity per country

According to the World Health Organization (WHO), worldwide obesity has nearly tripled since 1975. You're interested in the severity of this global health issue per country and whether males and females are affected differently. You'll use the WHO's obesity data (obesity_df) to investigate this issue. The data holds the percentage of females, males, and both sexes combined that are considered obese (BMI > 30) per country.

You want to create a scatterplot where, per nation, you can see the obesity data colored differently for females and males. This implies that sex should become a variable with its own column.

Load the ggplot2 package.

Instructions 100 XP

- Pivot the male and female columns. The old column names should go in the sex column, the original values should go in the pct_obese column.

ex_009.R

```
obesity_df %>%
  # Pivot the male and female columns
  pivot_longer(
    c("male", "female"),
    names_to = "sex",
    values_to = "pct_obese"
  )
obesity_df %>%
  # Pivot the male and female columns
  pivot_longer(c(male, female),
               names_to = "sex",
               values_to = "pct_obese") %>%
  # Create a scatter plot with pct_obese per country colored by sex
  ggplot(aes(x = country, color = sex,
             y = forcats::fct_reorder(country, both_sexes))) +
  geom_point() +
  scale_y_discrete(breaks = c("India", "Nauru", "Cuba", "Brazil",
                              "Pakistan", "Gabon", "Italy", "Oman",
                              "China", "United States of America")) +
  labs(x = "% Obese", y = "Country")
```

2.3 Bond... James Bond

You've been given a James Bond movie dataset (`bond_df`) and want to visualize the number of movies that Bond actors have featured in per decade. However, the data is in an untidy format with the decade values captured in the column headers. You'll tidy this dataset to give each variable its own column.

The ggplot2 package is needed.

Instructions 100 XP

ex_010.R

```r
bond_df %>%
  # Pivot the data to long format and set the column names
pivot_longer(
  c(
    `1960`,
    `1970`,
    `1980`,
    `1990`,
    `2000`,
    `2010`,
    `2020`
  ),
  names_to = "decade",
  values_to = "n_movies"
)
bond_df %>%
  # Pivot the data to long format
  pivot_longer(
    -Bond,
    # Overwrite the names of the two newly created columns
    names_to = "decade",
    values_to = "n_movies",
    # Drop na values
    values_drop_na = TRUE
  )
  bond_df %>%
  # Pivot the data to long format
  pivot_longer(
    -Bond,
    # Overwrite the names of the two newly created columns
    names_to = "decade",
    values_to = "n_movies",
    # Drop na values
    values_drop_na = TRUE,
    # Transform the decade column data type to integer
    names_transform = list(decade = as.integer)
  ) %>%
  ggplot(aes(x = decade + 5, y = n_movies, fill = Bond))+
  geom_col()
```

2.4 New-Zealand's bird of the year

Every year New Zealanders vote en masse to decide which species gets the bird of the year trophy. The contest is organized by the Forest & Bird agency which allows each person to give points to up to five birds (first pick gets 5 points, second 4, ...). Your job is to decide this year's winner from the messy dataset that's been pre-loaded for you as bird_df.

The dplyr package is needed. ### Instructions `100 XP` {.unnumbered}

ex_011.R

```
bird_df %>%
  # Pivot the data to create a two column data frame
  pivot_longer(
    c(points_5, points_4, points_3, points_2, points_1),
    names_to = "points",
    names_prefix = "points_",
    names_transform = list(points = as.integer),
    values_to = "species",
    values_drop_na = TRUE
  )
bird_df %>%
  # Pivot the data to create a 2 column data frame
  pivot_longer(
    starts_with("points_"),
    names_to = "points",
    names_prefix = "points_",
    names_transform = list(points = as.integer),
    values_to = "species",
    values_drop_na = TRUE
  ) %>%
  group_by(species) %>%
  summarize(total_points=sum(points)) %>%
  slice_max(total_points, n = 5)
```

2.5 Big tech stock prices

You're an analyst at an investment firm and want to visualize the weekly closing prices of five big tech firms' stocks. However, the dataset you've been handed (`stock_df`) is messy and has the year and week variables stored in the column headers. You'll pivot this data into a tidy format, extract the variables from the headers, and create a line plot.

Load the ggplot2 package. ### Instructions `100 XP` {.unnumbered} - Pivot stock_df so that the integer columns year and week are created from the column names and the original values are moved to the price column. Use the `names_sep` argument to separate the column names.

ex_012.R

```
stock_df %>%
  # Pivot the data to create 3 new columns: year, week, price
pivot_longer(
  -company,
  names_to = c("year", "week"),
  names_transform = list(year=as.integer, week=as.integer),
  values_to = "price",
  names_sep = "_week"
)

stock_df %>%
  # Pivot the data to create 3 new columns: year, week, price
  pivot_longer(
    -company,
    names_to = c("year", "week"),
    values_to = "price",
    names_sep = "_week",
    names_transform = list(
      year = as.integer,
      week = as.integer)
  ) %>%
  # Create a line plot with price per week, color by company
  ggplot(aes(x=week, y=price, group(company), color=company)) +
  geom_line() +
  facet_grid(. ~ year)
```

2.6 Soviet space dogs, the dogs perspective

You'll be working on an pre-processed sample of the USSR space dogs database compiled by Duncan Geere and pre-loaded for you as `space_dogs_df`. Each of the 42 rows in this dataset represents a test rocket launch which had one or two very brave dogs on board.

Your goal is to reshape this dataset so that for each launch, each dog has a row.

The challenge is that in the column headers (`name_1`, `name_2`, `gender_1`, and `gender_2`), the part before the `_` separator can point to two different variables (name and gender), while the

second part always points to the dog ID (1st or 2nd dog).

Instructions `100 XP`

- As the first argument to pivot_longer(), pass the columns to pivot (`name_1`, `name_2`, `gender_1`, and `gender_2`).
- Complete the names_to argument so that the first part of the column headers are reused.
- Make sure NA values are dropped since not all rockets had two dogs.

ex_013.R

```
space_dogs_df %>%
  pivot_longer(
    # Add the columns to pivot
    c(
      "name_1",
      "name_2",
      "gender_1",
      "gender_2"
    ),
    names_sep = "_",
    # Complete the names_to argument to re-use the first part of the column headers
    names_to = c(".value",  "dog_id"),
    # Make sure NA values are dropped
    values_drop_na = TRUE
  )
```

2.7 WHO obesity vs. life expectancy

You've been given a sample of WHO data (`who_df`) with obesity percentages and life expectancy data per country, year, and sex. You want to visually inspect the correlation between obesity and life expectancy.

However, the data is very messy with four variables hidden in the column names. Each column name is made up of three parts separated by underscores: Values for the year, followed by those for `sex`, and then values for either `pct.obese` or `life.exp`. Since the third part of the column name string holds two variables you'll need to use the special "`.value`" value in the `names_to` argument.

You'll pivot the data into a tidy format and create the `scatterplot`.

Load the `ggplot2` package.

Instructions 100 XP

ex_014.R

```
who_df %>%
  # Put each variable in its own column
  pivot_longer(
    -country,
    names_to = c("year", "sex", ".value"),
    names_sep = "_",
    names_transform = list("year" = as.integer)
  ) %>%
  # Create a plot with life expectancy over obesity
  ggplot(
    aes(
      x = pct.obese,
      y = life.exp,
      color=sex
    )
  ) +
  geom_point()
```

2.8 Unconting ovservations

You've found the job of your dreams providing technical support for a dog breed beauty contest. The jury members want a spreadsheet with the breed and id of each participating dog so that they can add the scores later on. You've only been given the number of participants per dog breed (`dog_df`) so you decide to use your tidyr skills to create the desired result.

Instructions 100 XP

- Inspect the data in the console.
- Uncount the data so that per breed, each dog gets a row and an ID. The ID should go in the `dog_id` column.

ex_015.R

```
dog_df %>%
  # Create one row for each participant and add the id
  uncount(n_participants, .id = "dog_id")
```

2.9 Soviet space dogs, the flight perspective

Remember the USSR space dogs dataset[1]? You changed it to a long format so that for every dog in every rocket launch, there was a row. Suppose you're given this tidy dataset and are asked to answer the question, "In what percentage of flights were both dogs of the same gender?"

You'll reshape and investigate space_dogs_df to find the answer.

The dplyr package has been pre-loaded for you.

Instructions 100 XP

ex_016.R

```
space_dogs_df %>%
  # Pivot the data to a wider format
  pivot_wider(
    names_from = dog_id,
    values_from = gender,
    names_prefix = "gender_"
  ) %>%
  # Drop rows with NA values
  drop_na() %>%
  # Create a Boolean column on whether both dogs have the same gender
  mutate(
    same_gender= ifelse(gender_1==gender_2,
    TRUE,
    FALSE
    )
  ) %>%
  summarize(pct_same_gender = mean(same_gender))
```

2.10 Planet temperature & distance to the Sun

The intensity of light radiated by a light source follows an inverse square relationship with the distance it has traveled. https://en.wikipedia.org/wiki/Inverse-square_law You wonder if you could observe this trend in the temperature of the planets in our Solar System given

[1]Compiled by Duncan Geere.

their distance to the Sun. You'll use the `planet_df` dataset from the devstronomy project to investigate this.

Instructions `100 XP`

- Use the `pivot_wider()` function to extract column names from the `metric` column and values from the `value` column.

ex_017.R

```
planet_df %>%
  # Give each planet variable its own column
  pivot_wider(
    names_from = metric,
    values_from = value
  )

  planet_df %>%
  # Give each planet variable its own column
  pivot_wider(
    names_from = metric,
    values_from = value
  )
```

2.11 Transporting plantet data

You're again working on a planet dataset derived from the devstronomy project. This time, you're interested in the correlation between the diameter of a planet and the number of moons circling it.

However, the dataset (`planet_df`) has a row for each variable and a column for each planet (`observation`). You'll transpose this data in two steps and then create a plot to inspect the correlation.

Load ggplot2 package.

Instructions `100 XP`

- Pivot the data so that planet names are put in a column named `planet`.
- Pivot the data so that each variable in the `metric` column gets its own column.

- Use the `ggplot()` function to create a plot with the `number_of_moons` over `diameter`.

ex_018.R

```
planet_df %>%
  # Pivot all columns except metric to long format
  pivot_longer(
    -metric,
    names_to = "planet"
  )

planet_df %>%
  # Pivot all columns except metric to long format
  pivot_longer(-metric, names_to = "planet") %>%
  # Put each metric in its own column
  pivot_wider(names_from = metric, values_from = value) %>%
  # Plot the number of moons vs. planet diameter
  ggplot(aes(x=diameter, y=number_of_moons)) +
  geom_point(aes(size = diameter)) +
  geom_text(aes(label = planet), vjust = -1) +
  labs(x = "Diameter (km)", y = "Number of moons") +
  theme(legend.position = "none")
```

3 Expanding Data

Values can often be missing in your data, and sometimes entire observations are absent too. In this chapter, you'll learn how to complete your dataset with these missing observations. You'll add observations with zero values to counted data, expand time series to a full sequence of intervals, and more!

3.1 Letters of the genetic code

The basic building blocks of **RNA** are four molecules described by a single letter each: adenine (**A**), cytosine (**C**), guanine (**G**), and uracil (**U**). The information carried by an RNA strand can be represented as a long sequence of these four letters. To read this code, one has to divide this chain into sequences of three letters each (e.g.** GCU, ACG**, …). These three letter sequences are known as codons. The concept is illustrated in the image below.

Instructions `100 XP`

- Create a tibble with three columns called `letter1`, `letter2`, and `letter3` that holds all possible combinations of the vector letters using `expand_grid()`.
- Use the `unite()` function from chapter one to merge these three columns into a single column named codon. Use an empty string as the separator.

ex_019.R

```
letters <- c("A", "C", "G", "U")
# Create a tibble with all possible 3 way combinations
codon_df <- expand_grid(
    leter1 = letters,
    leter2 = letters,
    leter3 = letters
)
codon_df

letters <- c("A", "C", "G", "U")
# Create a tibble with all possible 3 way combinations
```

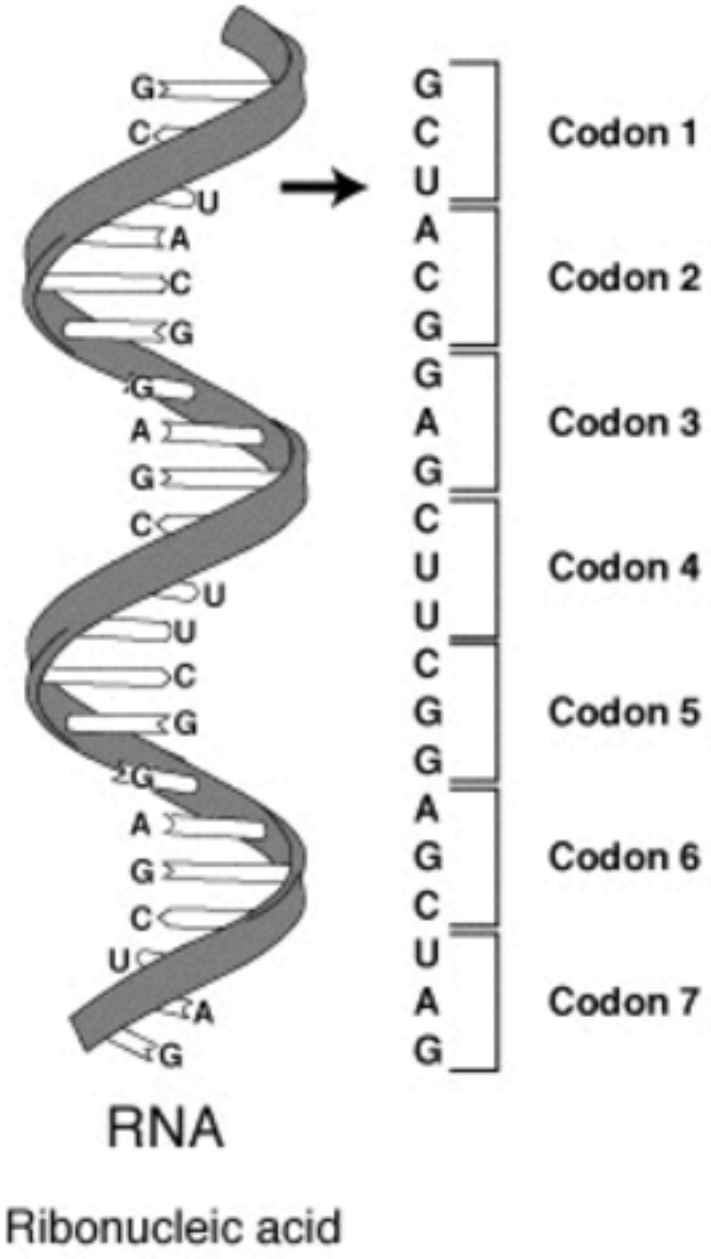

Figure 3.1: Helix rna model

```r
codon_df <- expand_grid(
  letter1 = letters,
  letter2 = letters,
  letter3 = letters
)
#
codon_df %>%
  # Unite these three columns into a "codon" column
  unite("codon",  c(letter1, letter2, letter3),
    sep = ''
  )
```

3.2 When did humans replace dogs in space

You already know that in the early days of spaceflight, the USSR was testing rockets with dogs. You now wonder when exactly humans started replacing dogs on space flight missions. You've been given a dataset `space_df` with the number of both dogs (compiled by Duncan Geere) and humans in space per year from 1951 till 1970 (collected from Wikipedia).

Your goal is to create a plot that shows you the number of individuals sent into space per species. Before you can create this plot, you'll first have to introduce zero values for missing combinations of `year` and `species`.

Load dplyr and ggplot2 packages.

Instructions `100 XP`

- Create `full_df`, a tibble with all unique combinations of the variables year (from 1951 to 1970) and `species` (`"Human"` and `"Dog"`).
- Perform a `right_join()` between `space_df` and `full_df` on the `year` and species columns.
- Use the ggplot() function to create a line plot of n_in_space over year, colored by species.
- Use the replace_na() function to overwrite NA values in the n_in_space column with zeros.

ex_020.R

```r
# Create a tibble with all combinations of years and species
full_df <- expand_grid(
  year = 1951:1970,
```

```
  species = c("Human", "Dog")
)

space_df %>%
  # Join with full_df so that missing values are introduced
  right_join(full_df, by = c("year", "species")) %>%
  # Create a line plot with n_in_space over year, color by species
  ggplot(
    aes(
      x = n_in_space,
      y = year,
      group = species,
      color = species
    )
  ) +
  geom_line()
# Create a tibble with all combinations of years and species
full_df <- expand_grid(
  year = 1951:1970,
  species = c("Human", "Dog")
)

space_df %>%
  # Join with full_df so that missing values are introduced
  right_join(full_df, by = c("year", "species")) %>%
  # Overwrite NA values for n_in_space with 0L
  replace_na(list(n_in_space = 0L)) %>%
  # Create a line plot with n_in_space over year, color by species
  ggplot(aes(x = year, y = n_in_space, color = species)) +
  geom_line()
```

3.3 Finding missing observations

You're an inspector at a nuclear plant and have to validate whether every reactor has received its daily safety check over the course of a full year. The safety check logs are in `reactor_df`, a data frame with columns date, reactor, and check.

Two vectors, `dates` and `reactors`, with all dates of the year and reactors at the plant respectively have been created for you. You'll use the combination of the `expand_grid()` and `anti_join()` functions to find dates where particular reactors were not checked.

Load `dplyr` package.

Instructions `100 XP`

- Use the `expand_grid()` function to create a tibble holding all combinations of the variables date and reactor. Use the dates and reactors vectors created for you.
- Perform an anti-join between `full_df` and `reactor_df` on the date and reactor columns.

ex_021.R

```
# Create a tibble with all combinations of dates and reactors
full_df <- expand_grid(
  date = dates,
  reactor = reactors
)

# Find the reactor - date combinations not present in reactor_df
full_df %>%
  anti_join(reactor_df, by=c("date", "reactor"))
```

3.4 Completing the Solar System

You have been given a data frame (`planet_df`) from the devstronomy project with the number of moons per planet in our Solar System. However, Mercury and Venus, the two moonless planets, are absent. You want to expand this dataset using the `complete()` function and a vector `planets` that contains all eight planet's names.

Instructions `100 XP`

- Complete the `planet` variable using the `planets` vector.
- Replace `NA` values in the `n_moons` variable with `0L` values.

ex_022.R

```
planets = c(
    "Mercury",
    "Venus",
    "Earth",
    "Mars",
```