Data science basics

Data classes

Prof. Dr. Jan Kirenz

The following content is based on Mine Çetinkaya-Rundel's excellent book Data Science in a Box

Data classes

Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- Vectors are like Lego building blocks
- We stick them together to build more complicated constructs, e.g. *representations of data*
- The **class** attribute relates to the S3 class of an object which determines its behaviour
  - You don't need to worry about what S3 classes really mean, but you can read more about it here if you're curious
- Examples: factors, dates, and data frames

Factors

R uses factors to handle categorical variables, variables that have a fixed and known set of possible values

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
## [1] BS  MS  PhD MS
## Levels: BS MS PhD
```

```
typeof(x)
```

```
## [1] "integer"
```

```
class(x)
```

```
## [1] "factor"
```

More on factors

We can think of factors like:

- character (level labels) and an
- integer (level numbers) glued together

```
glimpse(x)
```

```
##  Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

```
as.integer(x)
```

```
## [1] 1 2 3 2
```

Dates

```
y <- as.Date("2020-01-01")
y
```

```
## [1] "2020-01-01"
```

```
typeof(y)
```

```
## [1] "double"
```

```
class(y)
```

```
## [1] "Date"
```

More on dates

We can think of dates like an:

- integer (the number of days since the origin, 1 Jan 1970) and an
- integer (the origin) glued together

```
as.integer(y)
```

```
## [1] 18262
```

```
as.integer(y) / 365 # roughly 50 yrs
```

```
## [1] 50.03288
```

Data frames and Tibbles

We can think of data frames and tibbles like vectors of equal length glued together

```
df <- data.frame(x = 1:2, y = 3:4)
df
```

```
##   x y
## 1 1 3
## 2 2 4
```

```
tb <- tibble(x = 1:2, y = 3:4)
tb
```

```
## # A tibble: 2 x 2
##       x     y
##   <int> <int>
## 1     1     3
## 2     2     4
```

Data frames and Tibbles

```
typeof(df)
```

```
## [1] "list"
```

```
typeof(tb)
```

```
## [1] "list"
```

```
class(df)
```

```
## [1] "data.frame"
```

```
class(tb)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Lists

Lists are a generic vector container vectors of any type can go in them

```
l <- list(
  x = 1:4,
  y = c("hi", "hello", "jello"),
  z = c(TRUE, FALSE)
)
l
```

```
## $x
## [1] 1 2 3 4
##
## $y
## [1] "hi"    "hello" "jello"
##
## $z
## [1]  TRUE FALSE
```

Lists vs data frames and tibbles

- A data frame or tibble is a special list containing vectors of equal length
- When we use the `pull()` function, we extract a vector from the data frame or tibble

```
df
```

```
##   x y
## 1 1 3
## 2 2 4
```

```
df %>%
  pull(y)
```

```
## [1] 3 4
```

Working with factors

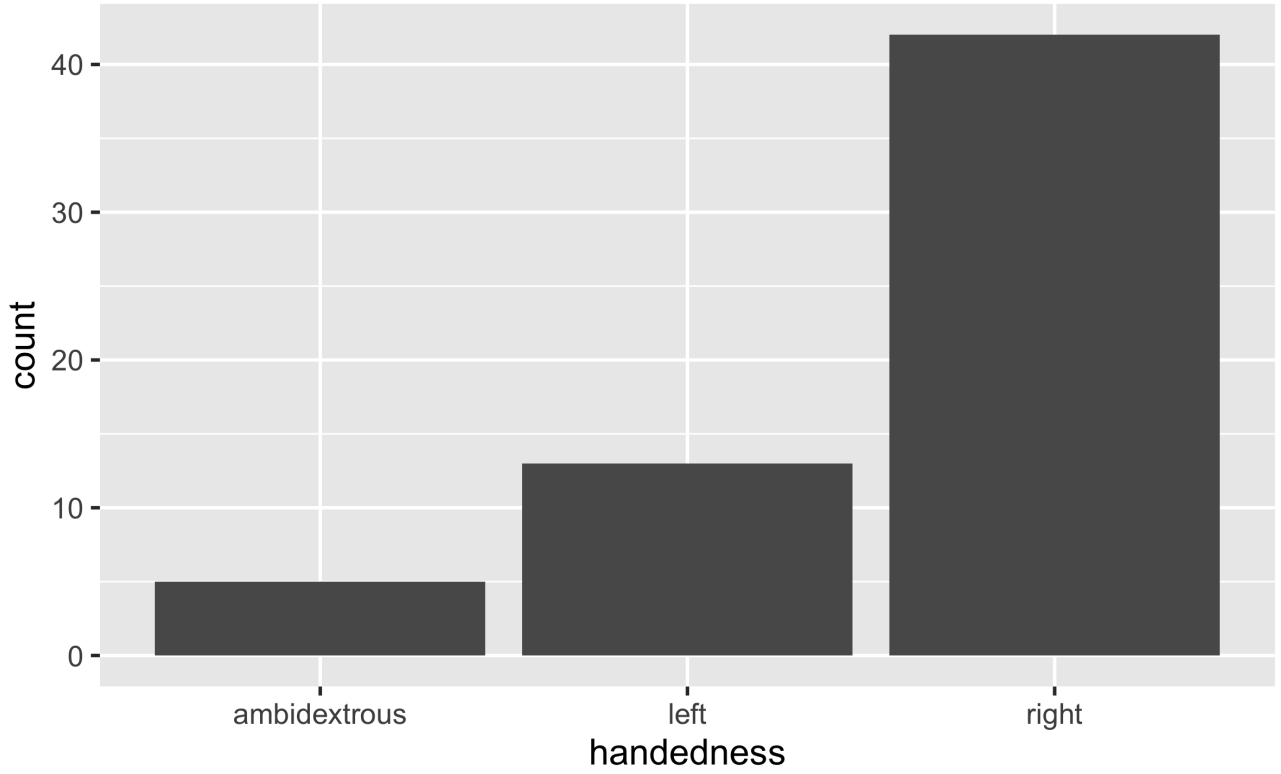
Read data in as character strings

```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name           <chr> "Bernice Warren", "Woodrow Stone", "Wil…
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1",…
## $ handedness     <chr> "left", "left", "left", "left", "left",…
```

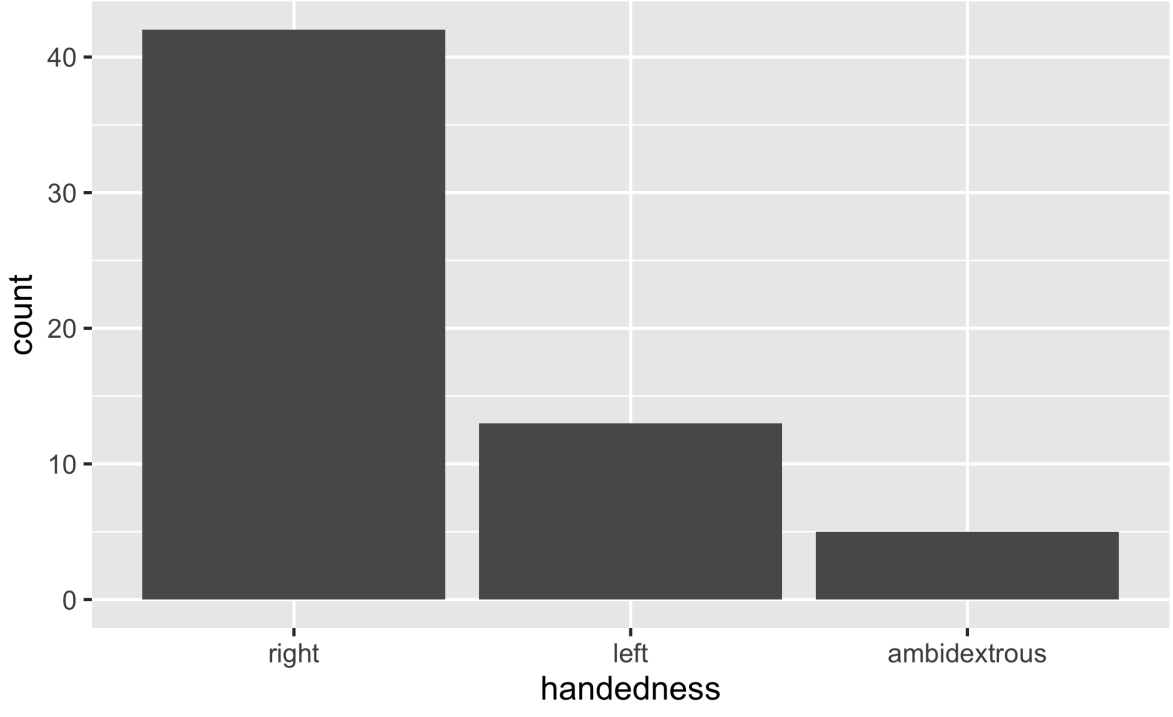
But coerce when plotting

```
ggplot(cat_lovers, mapping = aes(x = handedness)) +
  geom_bar()
```

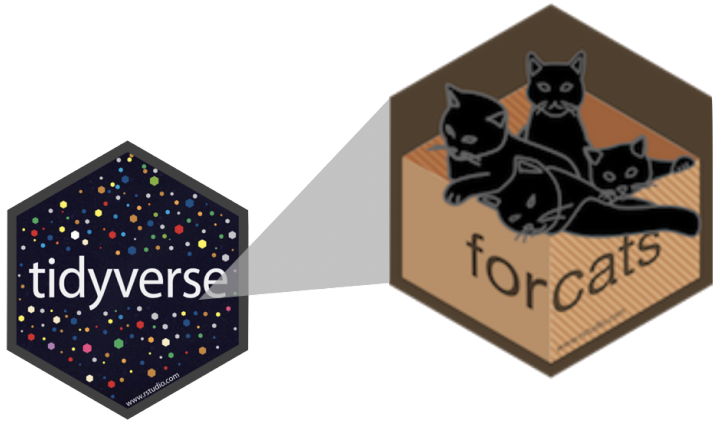


Use forcats to manipulate factors

```
cat_lovers %>%
  mutate(handedness = fct_infreq(handedness)) %>%
  ggplot(mapping = aes(x = handedness)) +
  geom_bar()
```



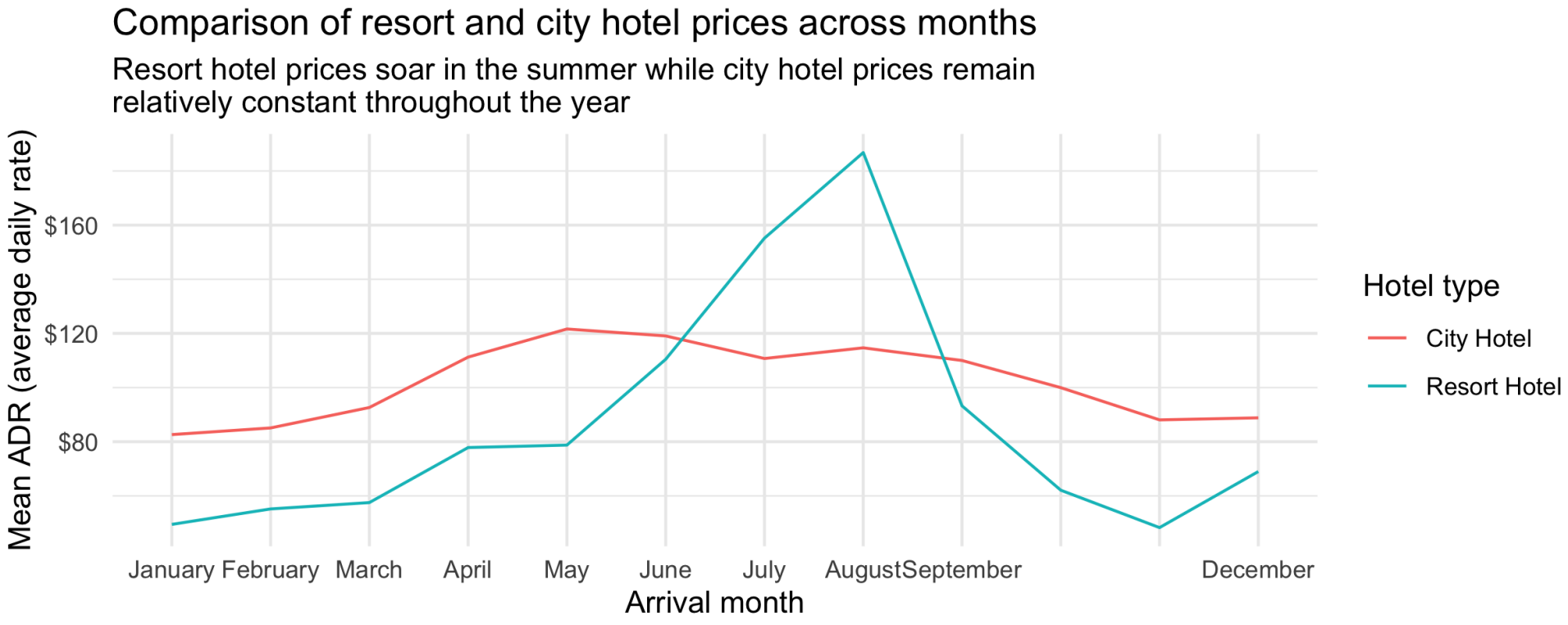
Forcats



- Factors are useful when you have true **categorical** data and you want to override the ordering of character vectors to improve display
- They are also useful in **modeling** scenarios
- The **forcats** package provides a suite of useful tools that solve common problems with factors

Your turn!

- RStudio Cloud > `AE 05 – Hotels + Data types` > `hotels-forcats.Rmd` > knit
- Recreate the x-axis of the following plot.
- **Stertch goal:** Recreate the y-axis.



Working with dates

Make a date

- **lubridate** is the tidyverse-friendly package that makes dealing with dates a little easier
- It's not one of the *core* tidyverse packages, hence it's installed with `install.packages("tidyverse)` but it's not loaded with it, and needs to be explicitly loaded with `library(lubridate)`

we're just going to scratch the surface of working with dates in R here...

Calculate and visualise the number of bookings on any given arrival date.

```
hotels %>%
  select(starts_with("arrival_"))
```

```
## # A tibble: 119,390 x 4
##   arrival_date_ye… arrival_date_mo… arrival_date_we…
##              <dbl> <chr>                        <dbl>
## 1             2015 July                            27
## 2             2015 July                            27
## 3             2015 July                            27
## 4             2015 July                            27
## 5             2015 July                            27
## 6             2015 July                            27
## # … with 119,384 more rows, and 1 more variable:
## #   arrival_date_day_of_month <dbl>
```

Step 1. Construct dates

```
library(glue)

hotels %>%
  mutate(
    arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_m
    ) %>%
  relocate(arrival_date)
```

```
## # A tibble: 119,390 x 33
##   arrival_date hotel is_canceled lead_time arrival_date_ye…
##   <glue>       <chr>       <dbl>     <dbl>             <dbl>
## 1 2015 July 1  Reso…           0       342              2015
## 2 2015 July 1  Reso…           0       737              2015
## 3 2015 July 1  Reso…           0         7              2015
## 4 2015 July 1  Reso…           0        13              2015
...
```

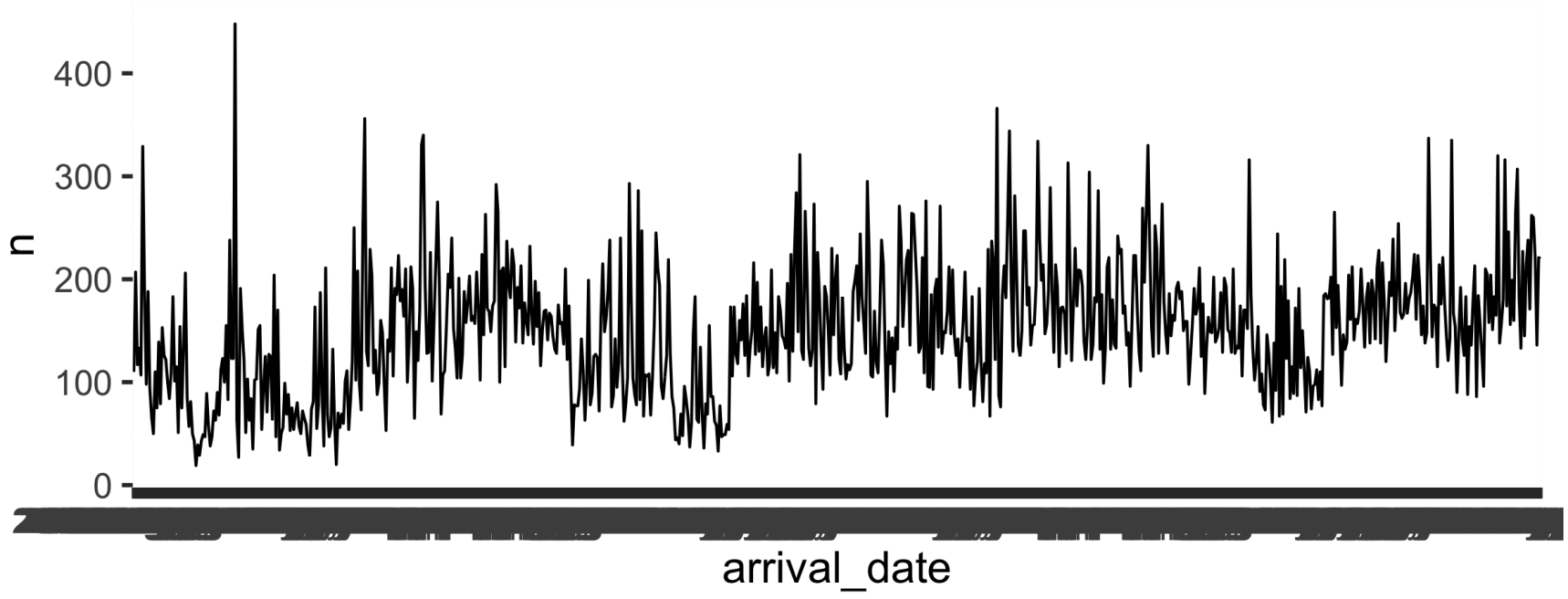
Step 2. Count bookings per date

```
hotels %>%
  mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day
  count(arrival_date)
```

```
## # A tibble: 793 x 2
##   arrival_date       n
##   <glue>         <int>
## 1 2015 August 1    110
## 2 2015 August 10   207
## 3 2015 August 11   117
## 4 2015 August 12   133
## 5 2015 August 13   107
## 6 2015 August 14   329
## # … with 787 more rows
```

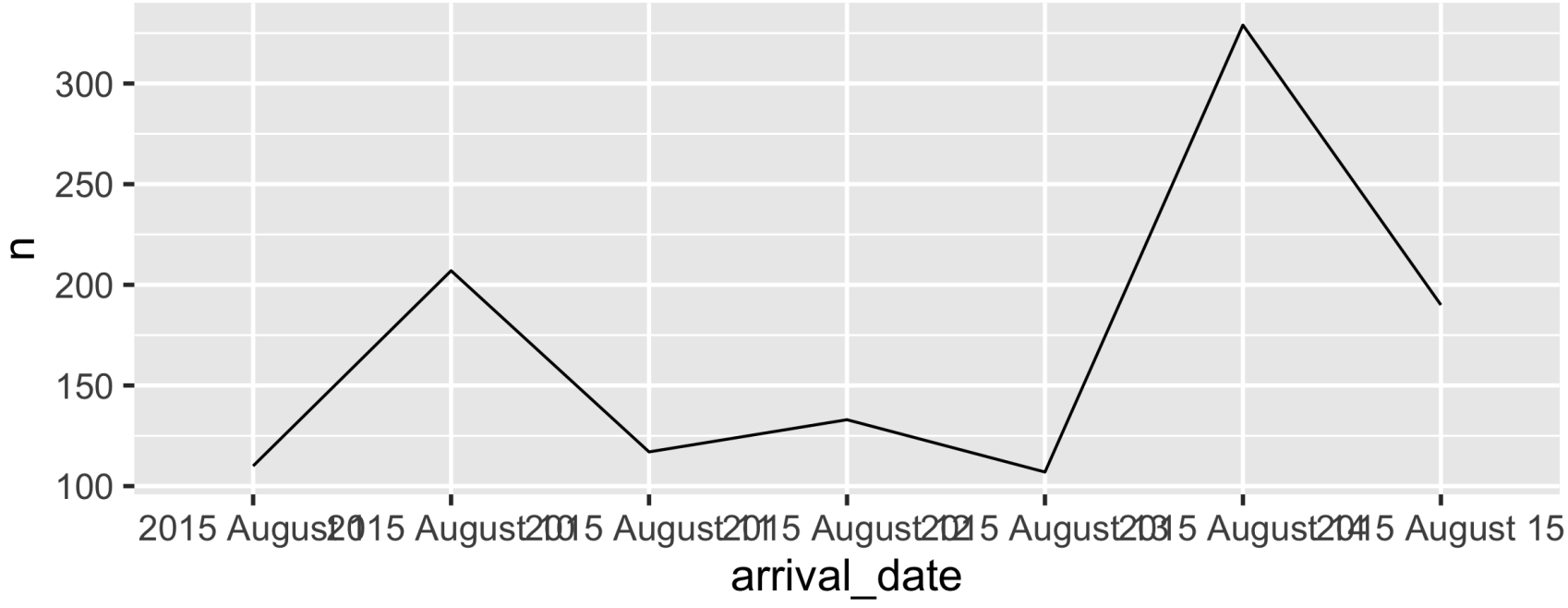
Step 3. Visualise bookings per date

```
hotels %>%
  mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day
  count(arrival_date) %>%
  ggplot(aes(x = arrival_date, y = n, group = 1)) +
  geom_line()
```



zooming in a bit...

Why does the plot start with August when we know our data start in July? And why does 10 August come after 1 August?



Step 1. *REVISED* Construct dates "as dates"

```
library(lubridate)

hotels %>%
  mutate(
    arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_
    ) %>%
  relocate(arrival_date)
```

```
## # A tibble: 119,390 x 33
##   arrival_date hotel is_canceled lead_time arrival_date_ye…
##   <date>       <chr>       <dbl>     <dbl>             <dbl>
## 1 2015-07-01   Reso…           0       342              2015
## 2 2015-07-01   Reso…           0       737              2015
## 3 2015-07-01   Reso…           0         7              2015
## 4 2015-07-01   Reso…           0        13              2015
...
```

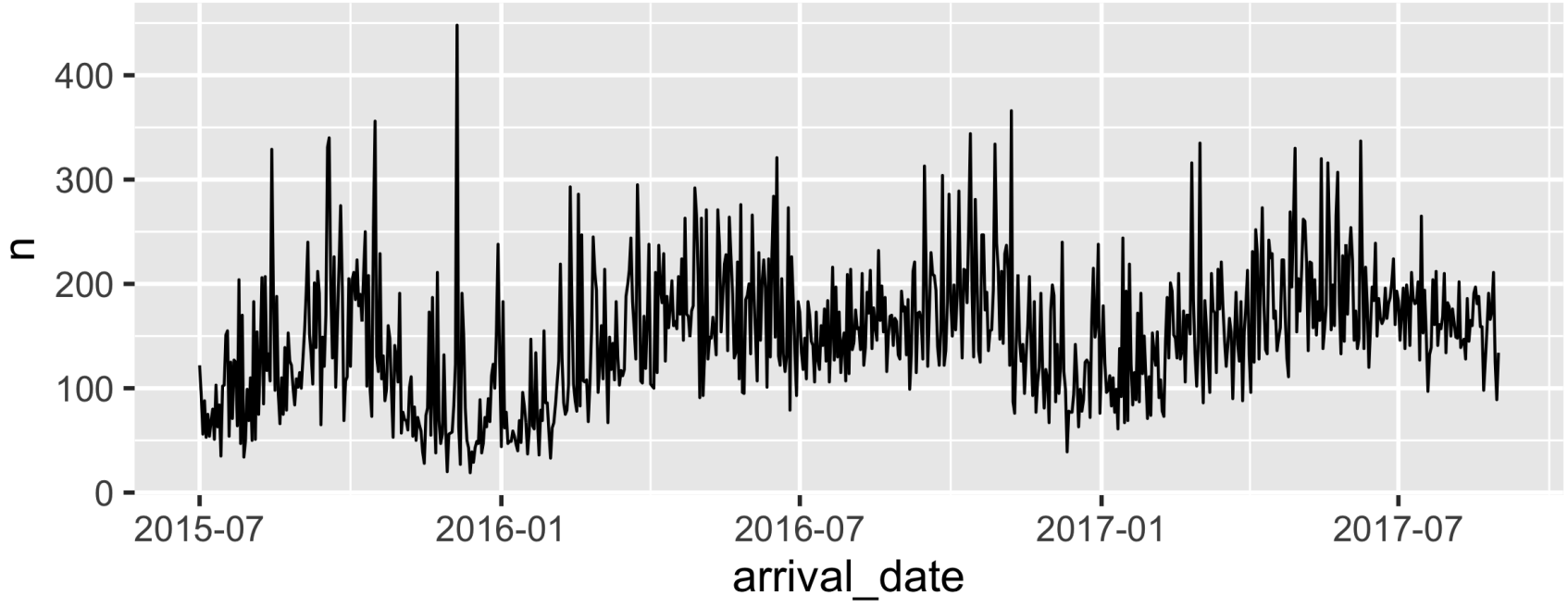
Step 2. Count bookings per date

```
hotels %>%
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date
  count(arrival_date)
```

```
## # A tibble: 793 x 2
##   arrival_date     n
##   <date>       <int>
## 1 2015-07-01     122
## 2 2015-07-02      93
## 3 2015-07-03      56
## 4 2015-07-04      88
## 5 2015-07-05      53
## 6 2015-07-06      75
## # … with 787 more rows
```

Step 3a. Visualise bookings per date

```
hotels %>%
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date
  count(arrival_date) %>%
  ggplot(aes(x = arrival_date, y = n, group = 1)) +
  geom_line()
```



Step 3b. Visualise using a smooth curve

```
hotels %>%
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date
  count(arrival_date) %>%
  ggplot(aes(x = arrival_date, y = n, group = 1)) +
  geom_smooth()
```