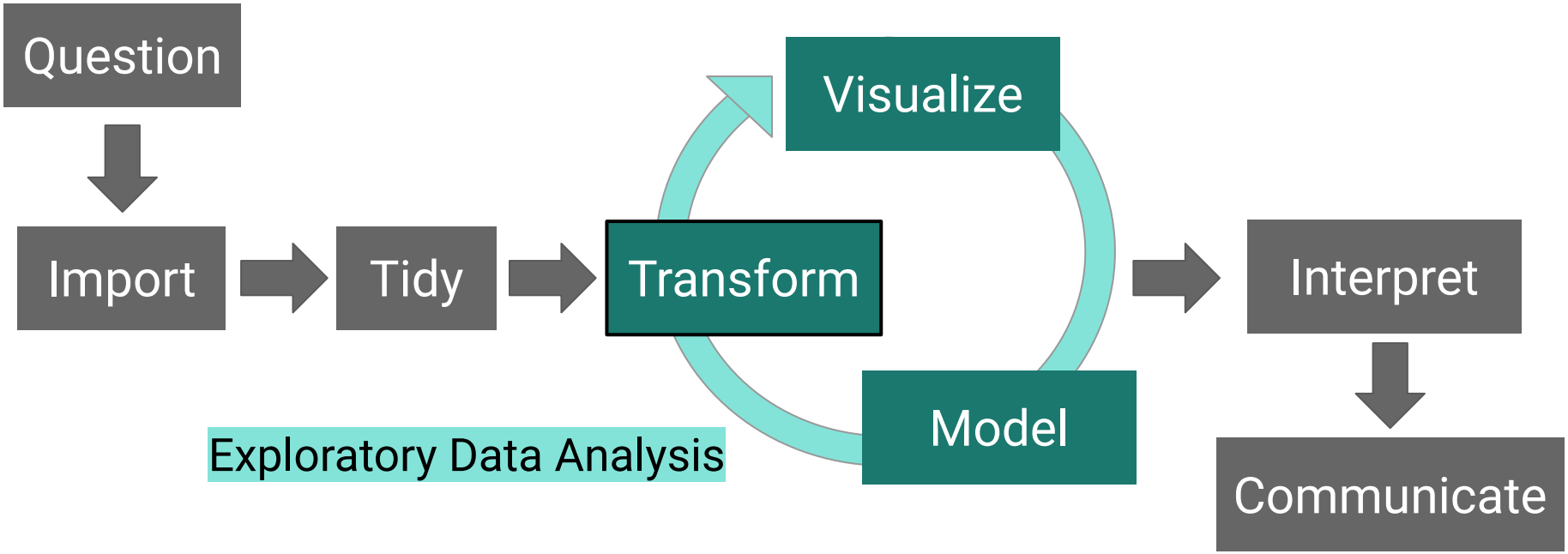
Factors and Dates

Lecture 9

Objective

- to handle categorical variables using **forcats**
- to handle time series data using **lubridate**

Motivation



Modified from https://r4ds.had.co.nz/explore-intro.html#explore-intro

forcats package

Tidyverse

Packages

forcats

forcats provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values. Go to docs...

https://www.tidyverse.org/packages/

Categorical variables

- Factors are used to work with **categorical variables**, which have a fixed and known set of possible values
- e.g. sex (male or female); months (Jan-Dec)

Creating factors

- A factor is a vector with **levels** attribute that contains mapping between integer values and categorical values.

```
> patient
```

	patient_ID	sex	age_year	weight_kg	height_cm
1	P001	female	1	9.1	73
2	P002	female	4	16.4	96
3	P003	female	2	10.5	85
4	P004	male	3	13.2	95
5	P005	male	4	15.9	104

```
> factor(patient$sex, levels=c("male", "female")
```

```
[1] female female female male   male
Levels: male female
```

Creating factors

```
> quarterly <− c("Apr", "Oct", "Jan", "Jul")
> sort(quarterly)
```

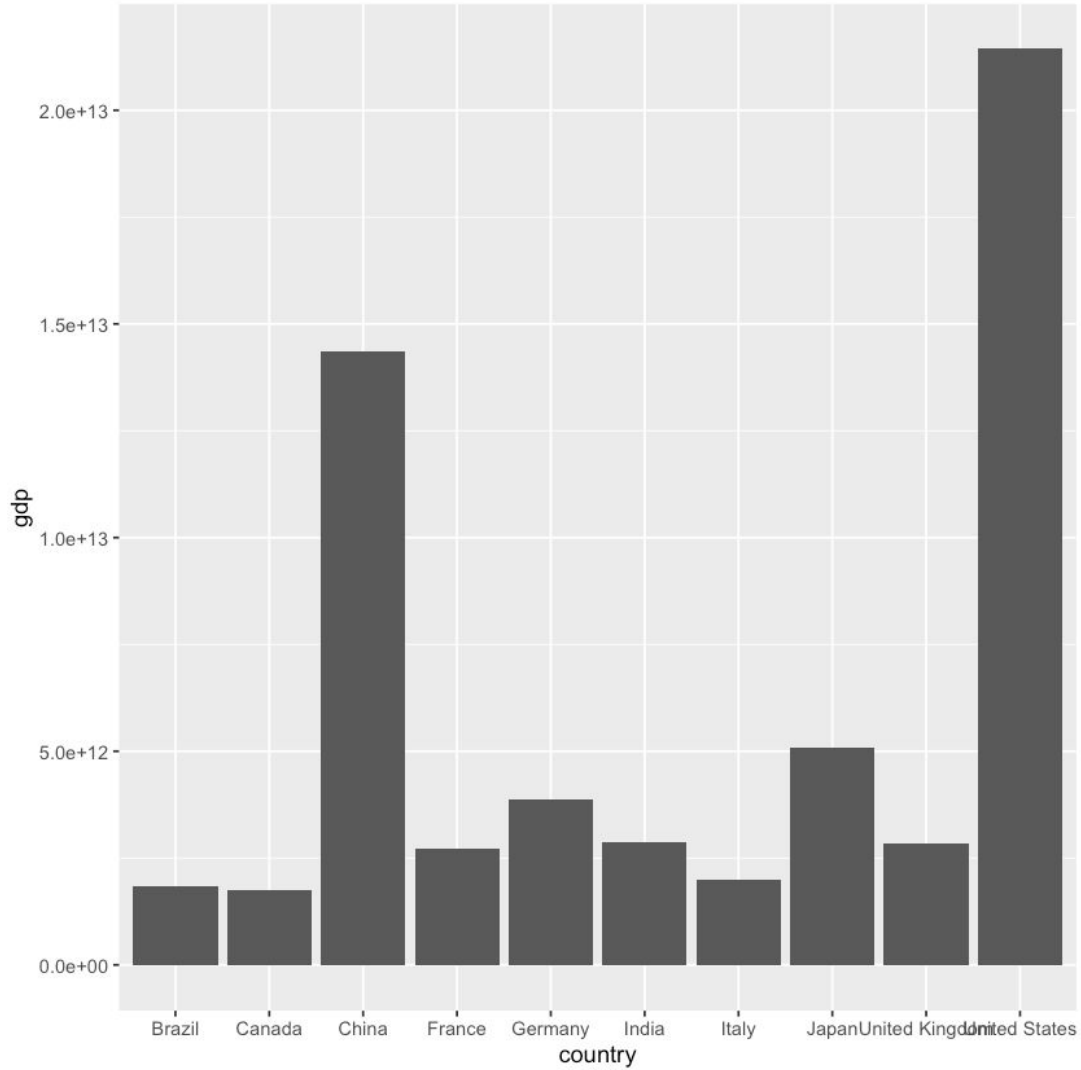
```
[1] "Apr" "Jan" "Jul" "Oct"
```

```
> month_levels <− c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
> quarterly <− factor(quarterly, levels = month_levels)
> sort(quarterly)
```

```
[1] Jan Apr Jul Oct
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

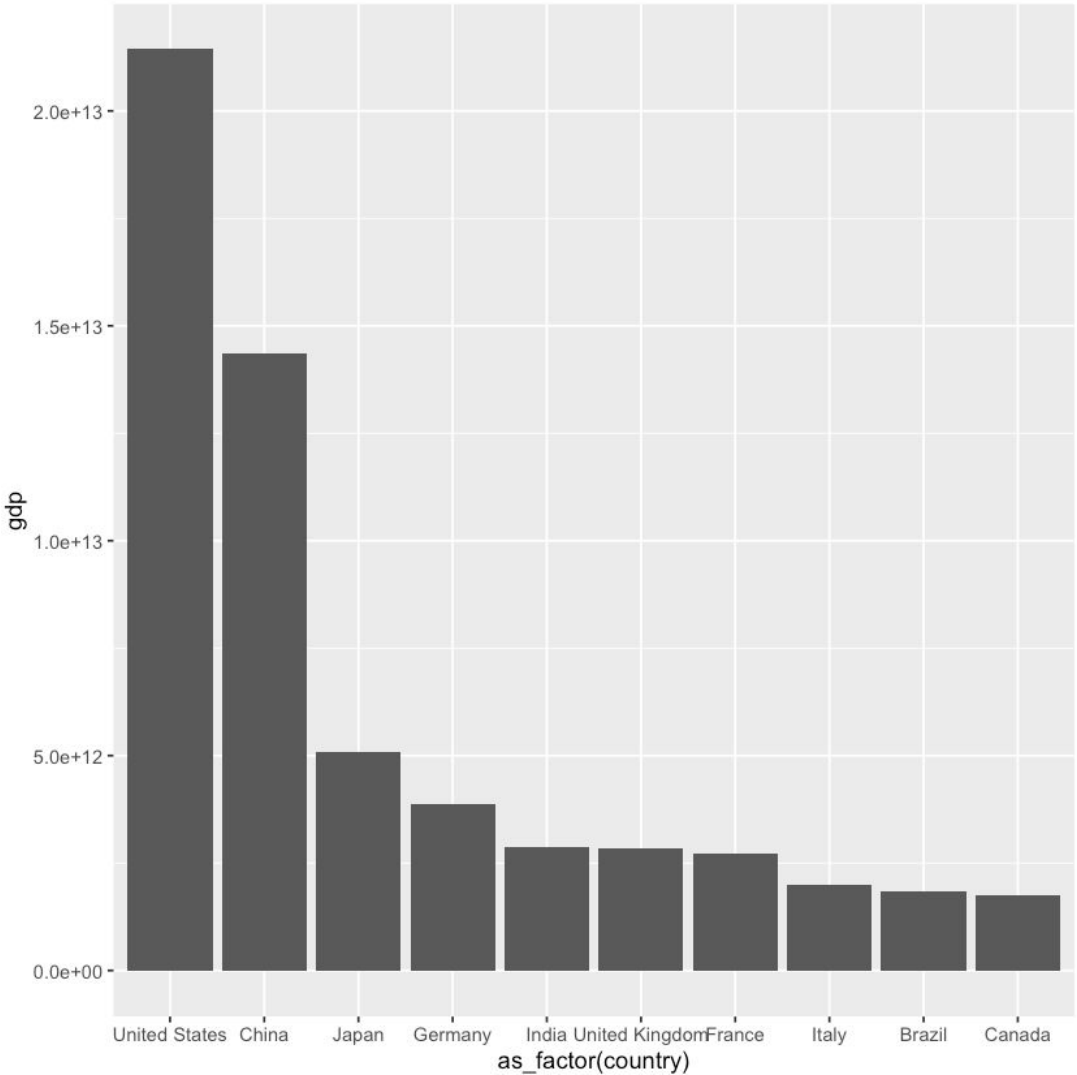
Modifying factor order

- It is often useful to change the order of the factor levels in a visualisation
- It is difficult to interpret this plot because there is no order or pattern



Modifying factor order

- This can be improved by converting a variable as factor using as_factor( )
- Or reordering the levels of using fct_reorder( )



Time series data

Handling dates and times are challenging due to:

- Leap years
- Time zones (UTC, EST, etc.)
- Various date formats (yyyy-mm-dd, dd/mm/yy, etc.)
- Various time formats (hh:mm:ss, hh:mm AM/PM)
- Daylight saving time

lubridate

CRAN 1.7.9.2 | R-CMD-check failing | codecov 76% | downloads 604K/month

devel 1.7.4.9000

Overview

Date-time data can be frustrating to work with in R. R commands for date-times are generally unintuitive and change depending on the type of date-time object being used. Moreover, the methods we use with date-times must be robust to time zones, leap days, daylight savings times, and other time related quirks, and R lacks these capabilities in some situations. Lubridate makes it easier to do the things R does with date-times and possible to do the things R does not.

https://github.com/tidyverse/lubridate

Creating date/times

3 types of date-time data:

- Date <date>
- Time <time>
- Date-time <dttm>

```
> today( )
```

```
[1] "2021-02-22"
```

```
> now( )
```

```
[1] "2021-02-22 10:39:59 JST"
```

Date/time from strings

- Date/time data often comes as strings.
- The order in which year, month, and day appear in your dates must be identified.

```
> ymd("2021/02/22")
```

[1] "2021-02-22"

```
> dmy("22-Feb-2021")
```

[1] "2021-02-22"

```
> mdy("February 22nd, 2021")
```

[1] "2021-02-22"

Date/time from strings

- Add underscore “_” + “hms” or “hm” to append time data

```
> ymd_hms("2021/02/22 10:39:59")
```

```
[1] "2021-02-22 10:39:59 UTC"
```

```
> ymd_hm("20210222 10:39")
```

```
[1] "2021-02-22 10:39:00 UTC"
```

```
> mdy("February 22nd, 2021")
```

```
[1] "2021-02-22"
```

date/time from individual components

- An object can be created from individual components of the date-time spread across multiple columns.

```
> flights
```

	year <int>	month <int>	day <int>	hour <dbl>	minute <dbl>
1	2013	1	1	5	15
2	2013	1	1	5	29
3	2013	1	1	5	40
4	2013	1	1	5	45
5	2013	1	1	6	0
6	2013	1	1	5	58 ...

date/time from individual components

- Use **make_date( )** for dates or **make_datetime( )** for date-times to create a date/time from individual components.

```
> flights %>%
  select(year, month, day, hour, minute) %>%
  mutate(departure = make_datetime(year, month, day, hour, minute))
```

	year	month	day	hour	minute	departure
	<int>	<int>	<int>	<dbl>	<dbl>	<dttm>
1	2013	1	1	5	15	2013-01-01 05:15:00
2	2013	1	1	5	29	2013-01-01 05:29:00
3	2013	1	1	5	40	2013-01-01 05:40:00
4	2013	1	1	5	45	2013-01-01 05:45:00
5	2013	1	1	6	0	2013-01-01 06:00:00
6	2013	1	1	5	58	2013-01-01 05:58:00

date/time from other types

- Use **as_datetime( )** and **as_date( )** functions to switch between a date-time and a date formats.

```
> as_date("2021/02/22 10:39:59")
```

[1] "2021-02-22"

```
> as_datetime("2021/02/22")
```

[1] "2021-02-22 UTC"

Date-time components

- Parse individual parts of the date with the accessor functions year(), month(), mday() (day of the month), yday() (day of the year), wday() (day of the week), hour(), minute(), and second()

```
> my_date <- ymd_hms("2021/02/22 10:39:59")
```

```
> year(my_date)
```

```
[1] 2021
```

```
> wday(my_date, label = TRUE)
```

```
[1] Mon
```

```
> week(my_date)
```

```
[1] 8
```

Setting components

- Use each accessor function to change the components of a date/time object.

```
> year(my_date) <− 1918
```

```
[1] "1918-02-22 10:39:59 UTC"
```

```
> day(my_date) <− 5
```

```
[1] "1918-02-05 10:39:59 UTC"
```

```
> hour(my_date) <− hour(my_date) + 1
```

```
[1] "1918-02-05 11:39:59 UTC"
```

Time spans

Three important classes that represent time spans:

- **durations**, which represent an exact number of seconds.
- **periods**, which represent human units like weeks and months.
- **intervals**, which represent a starting and ending point.

Durations

- A **difftime** object is returned when subtracting two dates

```
> my_date
```

```
[1] "1918-02-05 11:39:59 UTC"
```

```
> now() - my_date
```

```
Time difference of 37637.58 days
```

- Use **as.duration( )** function to return values in seconds, which consistent

```
> as.duration(now() - my_date)
```

```
[1] "3251714047.73497s (~103.04 years)"
```

Durations

```
> dseconds(60)
```

```
[1] "60s (~1 minutes)"
```

```
> dminutes(120)
```

```
[1] "7200s (~2 hours)"
```

```
> dweeks(1)
```

```
[1] "604800s (~1 weeks)"
```

```
> dyears(1) + dweeks(1) + ddays(12)
```

```
[1] "33199200s (~1.05 years)"
```

Periods

- Periods are time spans but not anchored in seconds
- Human-friendly representation, like days and months.

Periods

```
> seconds(60)
```

```
[1] "60S"
```

```
> minutes(120)
```

```
[1] "120M 0S"
```

```
> weeks(1)
```

```
[1] "7d 0H 0M 0S"
```

```
> dyears(1) + dweeks(1) + ddays(12)
```

```
[1] "1y 0m 19d 0H 0M 0S"
```

Intervals

- An **interval** is an accurate measurement between time point A and time point B.

```
> today <- ymd_hms("2021-02-22 11:05:16", tz = "Japan")
```

```
[1] "2021-02-22 11:05:16 JST"
```

```
> yesterday <- ymd_hms("2021-02-21 11:05:16", tz = "America/New_York")
```

```
[1] "2021-02-21 11:05:16 EST"
```

```
> interval(yesterday, today)
> yesterday %--% today          # same output
```

```
[1] 2021-02-21 11:05:16 EST--2021-02-21 21:05:16 EST
```

Take-away message

- **forcats** package is useful for handling categorical variables especially in plotting
- **lubridate** package is useful for handling time-series data