Data Visualisation Using R

Lecture-4

Suman Rakshit

School of EECMS, Curtin University

February 9, 2024

Outline

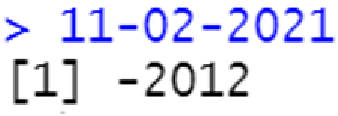
1. Dates: different convention of representing
2. Correct way to represent Dates
3. Dates in `R`
4. Datetime object in `R`
5. Datetime handling in `lubridate`
6. Creating time-series plots
7. Visualisation of spatial data

How do we represent Date in R?

If you write 11th February of 2021 as

$$11 - 02 - 2021$$

in R, you will get



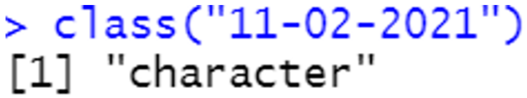
```
> 11-02-2021
[1] -2012
```

How do we represent Date in R?

If you put quotes around $11 - 02 - 2021$, it may work:

$$“11 - 02 - 2021”$$

But check the class of this object:



```
> class("11-02-2021")
[1] "character"
```

Different convention of representing dates

- Added confusion arises due to different conventions of representing the same date.
- In Australia, the date 11th February of 2021 may be represented either by $11 - 02 - 2021$ or by $11/02/2021$, or by $11/2/21$, or simply '11th Feb, 2021'. We like to follow the Day-Month-Year format.
- In USA, on the other hand, the same date may be written as $02 - 11 - 2021$, or as $2/11/2021$ or $2/11/21$, or 'Feb 11, 2021'. In USA they follow the Month-Day-Year format.

Different convention of representing dates

- Added confusion arises due to different conventions of representing the same date.
- In Australia, the date 11th February of 2021 may be represented either by $11-02-2021$ or by $11/02/2021$, or by $11/2/21$, or simply '11th Feb, 2021'. We like to follow the Day-Month-Year format.
- In USA, on the other hand, the same date may be written as $02-11-2021$, or as $2/11/2021$ or $2/11/21$, or 'Feb 11, 2021'. In USA they follow the Month-Day-Year format.

Curtin University

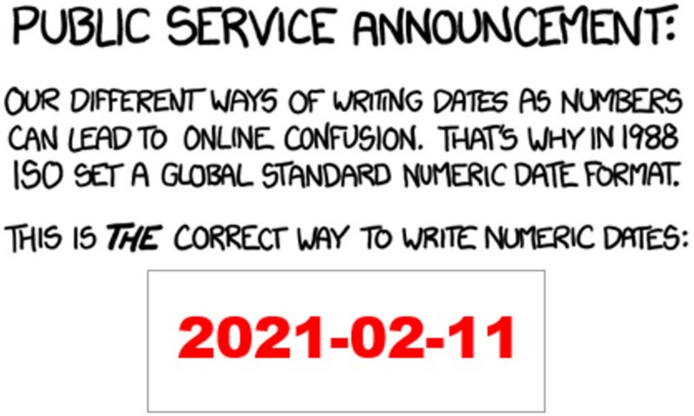
Different convention of representing dates

- Added confusion arises due to different conventions of representing the same date.
- In Australia, the date 11th February of 2021 may be represented either by $11 - 02 - 2021$ or by $11/02/2021$, or by $11/2/21$, or simply '11th Feb, 2021'. We like to follow the Day-Month-Year format.
- In USA, on the other hand, the same date may be written as $02 - 11 - 2021$, or as $2/11/2021$ or $2/11/21$, or 'Feb 11, 2021'. In USA they follow the Month-Day-Year format.

How to represent 11th Feb 2021

Global standard – ISO 8601

It turns out that there is a global standard of presenting dates (introduced in 1988), called ISO 8601 standard.


PUBLIC SERVICE ANNOUNCEMENT:
OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.
THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:
2021-02-11

Curtin University

ISO 8601 Date properties

- ISO 8601 states that the three components of a date should be written in the decreasing order of time units, i.e., first the year, then the month, and finally, the day.
- Each time component should have a fixed number of digits – Year should be 4 digits, Month should be 2 digits, and Day should be 2 digits.
- Because of the last point, the single digit days and months should be padded with a leading zero.
- You do not need a separator, but if you use a separator, it has to be a dash. So, the 11th Feb of 2021 will be written in ISO 8601 as

$$2021 - 02 - 11$$

ISO 8601 Date properties

- ISO 8601 states that the three components of a date should be written in the decreasing order of time units, i.e., first the year, then the month, and finally, the day.
- Each time component should have a fixed number of digits – Year should be 4 digits, Month should be 2 digits, and Day should be 2 digits.
- Because of the last point, the single digit days and months should be padded with a leading zero.
- You do not need a separator, but if you use a separator, it has to be a dash. So, the 11th Feb of 2021 will be written in ISO 8601 as

$$2021-02-11$$

ISO 8601 Date properties

- ISO 8601 states that the three components of a date should be written in the decreasing order of time units, i.e., first the year, then the month, and finally, the day.
- Each time component should have a fixed number of digits – Year should be 4 digits, Month should be 2 digits, and Day should be 2 digits.
- Because of the last point, the single digit days and months should be padded with a leading zero.
- You do not need a separator, but if you use a separator, it has to be a dash. So, the 11th Feb of 2021 will be written in ISO 8601 as

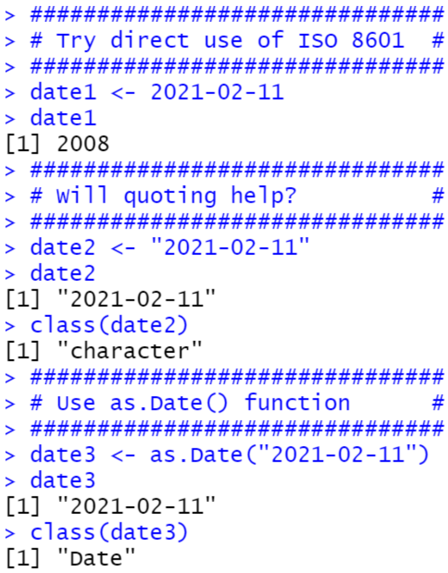
$$2021-02-11$$

ISO 8601 Date properties

- ISO 8601 states that the three components of a date should be written in the decreasing order of time units, i.e., first the year, then the month, and finally, the day.
- Each time component should have a fixed number of digits – Year should be 4 digits, Month should be 2 digits, and Day should be 2 digits.
- Because of the last point, the single digit days and months should be padded with a leading zero.
- You do not need a separator, but if you use a separator, it has to be a dash. So, the 11th Feb of 2021 will be written in ISO 8601 as

$$2021 - 02 - 11$$

Use as.Date() to create a Date object



```
> ################################
> # Try direct use of ISO 8601  #
> ################################
> date1 <- 2021-02-11
> date1
[1] 2008
> ################################
> # Will quoting help?           #
> ################################
> date2 <- "2021-02-11"
> date2
[1] "2021-02-11"
> class(date2)
[1] "character"
> ################################
> # Use as.Date() function       #
> ################################
> date3 <- as.Date("2021-02-11")
> date3
[1] "2021-02-11"
> class(date3)
[1] "Date"
```

Quiz: which one is correct ISO8601 format?

`as.Date()` will only accept ISO-8601 formatted dates.

So, which one is the correct ISO 8601 date format for 16th August of 2021?

1. "16-8-2021"
2. "2021-16-08"
3. "2021-08-16"
4. "2021-8-16"

Curtin University

Mathematical operations with Dates

- Behind the scenes, Dates are stored as number of days since 1970-01-01.
- As a result, we can perform standard mathematical comparisons and computations.
- We can ask if one date comes after another date: `as.Date("2021-08-16")>as.Date("2021-08-01")` The answer will be `TRUE`.
- We can add days: `as.Date("2021-08-16")+ 1` gives the answer `"2021-08-17"`.
- We can find the difference between dates: `as.Date("2022-08-16") − as.Date("2021-08-16")` gives the answer `365 days`.

Mathematical operations with Dates

- Behind the scenes, Dates are stored as number of days since 1970-01-01.
- As a result, we can perform standard mathematical comparisons and computations.
- We can ask if one date comes after another date: `as.Date("2021-08-16")>as.Date("2021-08-01")` The answer will be `TRUE`.
- We can add days: `as.Date("2021-08-16")+ 1` gives the answer `"2021-08-17"`.
- We can find the difference between dates: `as.Date("2022-08-16") − as.Date("2021-08-16")` gives the answer `365 days`.

Curtin University

Mathematical operations with Dates

- Behind the scenes, Dates are stored as number of days since 1970-01-01.
- As a result, we can perform standard mathematical comparisons and computations.
- We can ask if one date comes after another date: `as.Date("2021-08-16")>as.Date("2021-08-01")` The answer will be `TRUE`.
- We can add days: `as.Date("2021-08-16")+ 1` gives the answer `"2021-08-17"`.
- We can find the difference between dates: `as.Date("2022-08-16") − as.Date("2021-08-16")` gives the answer `365 days`.

Mathematical operations with Dates

- Behind the scenes, Dates are stored as number of days since 1970-01-01.
- As a result, we can perform standard mathematical comparisons and computations.
- We can ask if one date comes after another date: `as.Date("2021-08-16")>as.Date("2021-08-01")` The answer will be `TRUE`.
- We can add days: `as.Date("2021-08-16")+ 1` gives the answer `"2021-08-17"`.
- We can find the difference between dates: `as.Date("2022-08-16") − as.Date("2021-08-16")` gives the answer `365 days`.

Curtin University

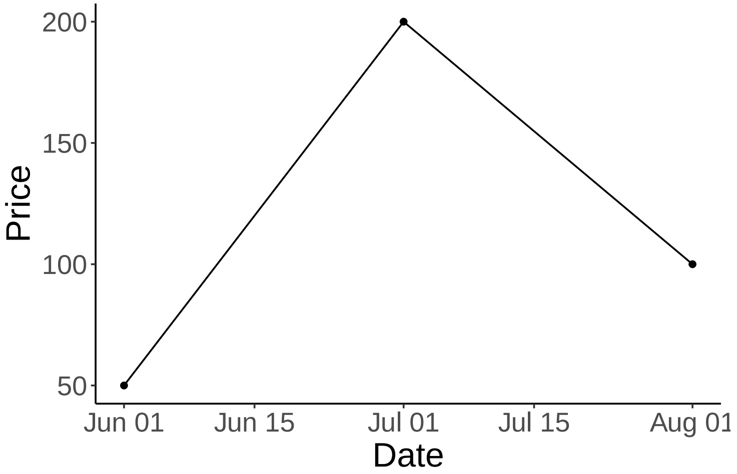
Mathematical operations with Dates

- Behind the scenes, Dates are stored as number of days since 1970-01-01.
- As a result, we can perform standard mathematical comparisons and computations.
- We can ask if one date comes after another date: `as.Date("2021-08-16")>as.Date("2021-08-01")` The answer will be `TRUE`.
- We can add days: `as.Date("2021-08-16")+ 1` gives the answer `"2021-08-17"`.
- We can find the difference between dates: `as.Date("2022-08-16") − as.Date("2021-08-16")` gives the answer `365 days`.

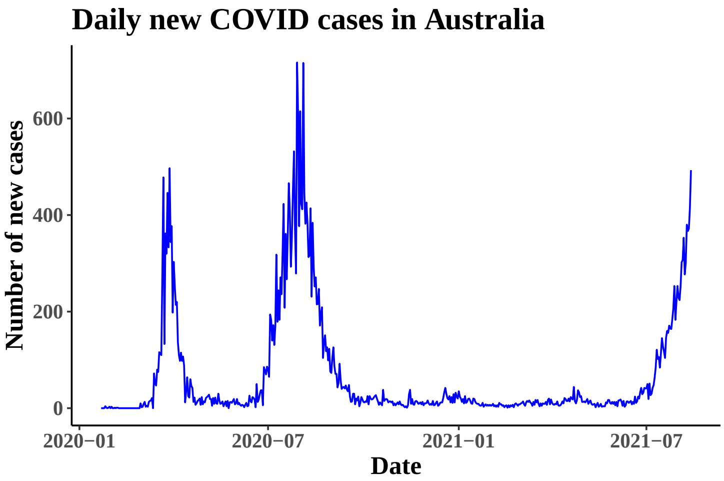
RCode: Plotting with Date objects

```r
# Create three Dates
Date <- c(as.Date("2021-06-01"),
          as.Date("2021-07-01"),
          as.Date("2021-08-01"))
# Create a time-series
Price <- c(50, 200, 100)
# Create the time-series dataset
data <- data.frame(Date, Price)
# Plot the time-series
ggplot(data, aes(Date, Price)) +
  geom_line() + geom_point() +
  theme_classic()
```

Plot of the time-series



Plot of real-life time-series



Let's talk about Time

1. ISO 8601 convention is to put time units again in the decreasing order: **HH:MM:SS**.

2. Each time unit should have fixed number of digits:
   - Hours: 00–24
   - Minutes: 00–59
   - Seconds: 00–60 (60 only for leap seconds)

3. Can use no or : as separator.

Datetimes objects in R

- Datetimes can be stored using two objects
  1. `POSIXlt` – list with named components
  2. `POSIXct` – seconds since 1970-01-01 00:00:00
- `POSIXct` is better suited to be stored in a Data Frame column.
- `as.POSIXct()` turns a ISO 8601 datetime string into a `POSIXct object`.

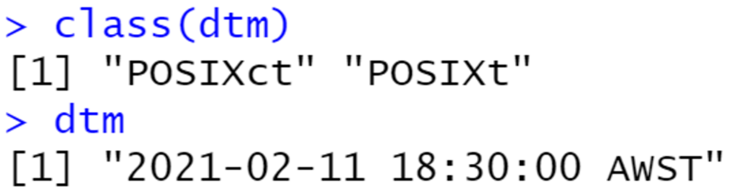
Quiz: which one is valid ISO 8601 format?

Which one is the valid ISO-8601 Datetime representation for 6:30 pm of 11th February of 2021.

1. "2021-02-11 06:30:00"
2. "2021-02-11 06:00:30"
3. "2021-02-11 18:00:30"
4. "2021-02-11 18:30:00"

Example of using `as.POSIXct()`

```r
# Define 6:30 pm of 11th Feb, 2021 as a
    string
dtmString <- "2021-02-11 18:30:00"
# Create the Datetime object
dtm <- as.POSIXct(dtmString)
```



```
> class(dtm)
[1] "POSIXct" "POSIXt"
> dtm
[1] "2021-02-11 18:30:00 AWST"
```

Let's talk about Timezones

- In ISO 8601 convention, if no timezone is specified, the local timezone is assumed:
  - "2021-02-11T18:30:00" – 6:30 pm Local Time on 11th Feb, 2021
- If you add a "Z" at the end of Datetime specification, then a UTC timezone is assumed:
  - "2021-02-11T18:30:00Z" – 6:30 pm UTC (coordinated universal time) on 11th Feb, 2021
- In ISO 8601 convention, any other timezone is defined as the offset from the UTC timezone:
  - "2021-02-11T18:30:00+08:00" – 6:30 pm in Perth

Let's talk about Timezones

- In ISO 8601 convention, if no timezone is specified, the local timezone is assumed:
  - "2021-02-11T18:30:00" – 6:30 pm Local Time on 11th Feb, 2021
- If you add a "Z" at the end of Datetime specification, then a UTC timezone is assumed:
  - "2021-02-11T18:30:00Z" – 6:30 pm UTC (coordinated universal time) on 11th Feb, 2021
- In ISO 8601 convention, any other timezone is defined as the offset from the UTC timezone:
  - "2021-02-11T18:30:00+08:00" – 6:30 pm in Perth

Curtin University

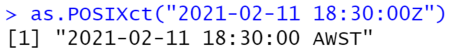
Let's talk about Timezones

- In ISO 8601 convention, if no timezone is specified, the local timezone is assumed:
  - "2021-02-11T18:30:00" – 6:30 pm Local Time on 11th Feb, 2021
- If you add a "Z" at the end of Datetime specification, then a UTC timezone is assumed:
  - "2021-02-11T18:30:00Z" – 6:30 pm UTC (coordinated universal time) on 11th Feb, 2021
- In ISO 8601 convention, any other timezone is defined as the offset from the UTC timezone:
  - "2021-02-11T18:30:00+08:00" – 6:30 pm in Perth

Curtin University

Timezone specification in as.POSIXct()

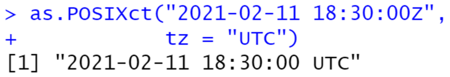
Unfortunately, as.POSIXct does not recognise ISO-8601 timezone specifications.



```
> as.POSIXct("2021-02-11 18:30:00Z")
[1] "2021-02-11 18:30:00 AWST"
```

Figure 1: UTC specification shows local timezone – AWST stands for Australian Western Standard Time (UTC+8).

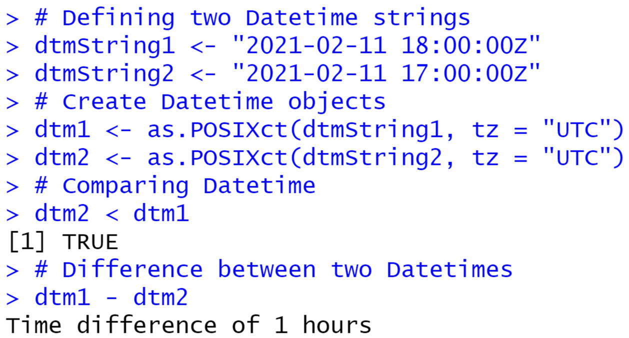
To specify a time zone, you have to access the tz parameter in the function as.POSIXct:



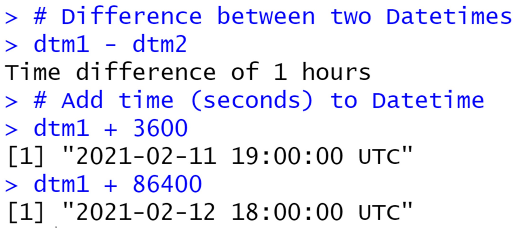
```
> as.POSIXct("2021-02-11 18:30:00Z",
+            tz = "UTC")
[1] "2021-02-11 18:30:00 UTC"
```

Figure 2: tz = "UTC sets the UTC timezone.

Operations with Datetime objects



```
> # Defining two Datetime strings
> dtmString1 <- "2021-02-11 18:00:00Z"
> dtmString2 <- "2021-02-11 17:00:00Z"
> # Create Datetime objects
> dtm1 <- as.POSIXct(dtmString1, tz = "UTC")
> dtm2 <- as.POSIXct(dtmString2, tz = "UTC")
> # Comparing Datetime
> dtm2 < dtm1
[1] TRUE
> # Difference between two Datetimes
> dtm1 - dtm2
Time difference of 1 hours
```

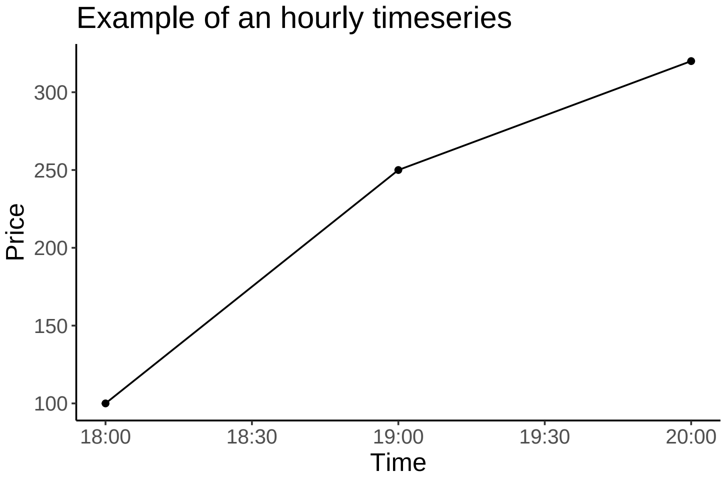


```
> # Difference between two Datetimes
> dtm1 - dtm2
Time difference of 1 hours
> # Add time (seconds) to Datetime
> dtm1 + 3600
[1] "2021-02-11 19:00:00 UTC"
> dtm1 + 86400
[1] "2021-02-12 18:00:00 UTC"
```

RCode: plotting datetime objects

```
# Create three consecutive hours
Time <- c(as.POSIXct("2021-02-11
   18:00:00"),
 as.POSIXct("2021-02-11 19:00:00"),
 as.POSIXct("2021-02-11 20:00:00"))
# Create price data
Price <- c(100, 250, 320)
# Create hourly timeseries
data <- data.frame(Time, Price)
# Plot hourly timeseries
ggplot(data, aes(Time, Price)) +
   geom_line() +  geom_point() +
   theme_classic() +
ggtitle("Example of an hourly timeseries
   ")
```

Plot of hourly data



lubridate makes handling Datetime easy

- lubridate makes it as easy as possible to work with date and time in R.
- lubridate is part of the tidyverse – which means
  - it works well with built-in datetime objects;
  - it works well with other tidyverse packages;
  - function names are very intuitive and easy to work with.
- lubridate allows consistent behaviour regardless of the underlying datetime object;
- You just need to learn one lubridate function to achieve a given task for any datetime object – be it stored in a Date object, or in a POSIXct object, or in any other timeseries objects such as xts or zoo.

lubridate makes handling Datetime easy

- lubridate makes it as easy as possible to work with date and time in R.
- lubridate is part of the tidyverse – which means
  - it works well with built-in datetime objects;
  - it works well with other tidyverse packages;
  - function names are very intuitive and easy to work with.
- lubridate allows consistent behaviour regardless of the underlying datetime object;
- You just need to learn one lubridate function to achieve a given task for any datetime object – be it stored in a Date object, or in a POSIXct object, or in any other timeseries objects such as xts or zoo.

lubridate makes handling Datetime easy

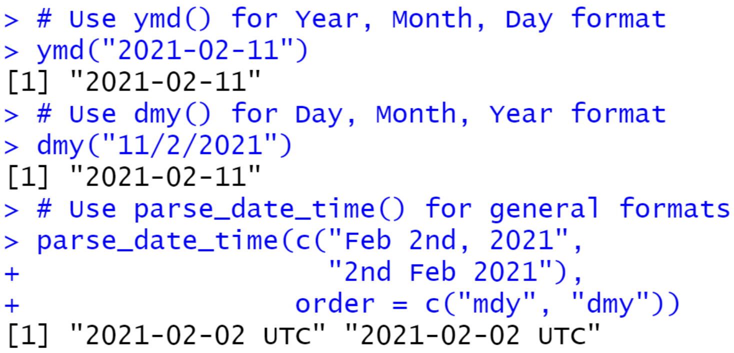
- lubridate makes it as easy as possible to work with date and time in R.
- lubridate is part of the tidyverse – which means
  - it works well with built-in datetime objects;
  - it works well with other tidyverse packages;
  - function names are very intuitive and easy to work with.
- lubridate allows consistent behaviour regardless of the underlying datetime object;
- You just need to learn one lubridate function to achieve a given task for any datetime object – be it stored in a Date object, or in a POSIXct object, or in any other timeseries objects such as xts or zoo.

Curtin University

lubridate makes handling Datetime easy

- lubridate makes it as easy as possible to work with date and time in R.
- lubridate is part of the tidyverse – which means
  - it works well with built-in datetime objects;
  - it works well with other tidyverse packages;
  - function names are very intuitive and easy to work with.
- lubridate allows consistent behaviour regardless of the underlying datetime object;
- You just need to learn one lubridate function to achieve a given task for any datetime object – be it stored in a Date object, or in a POSIXct object, or in any other timeseries objects such as xts or zoo.

Key Date functions in lubridate



```
> # Use ymd() for Year, Month, Day format
> ymd("2021-02-11")
[1] "2021-02-11"
> # Use dmy() for Day, Month, Year format
> dmy("11/2/2021")
[1] "2021-02-11"
> # Use parse_date_time() for general formats
> parse_date_time(c("Feb 2nd, 2021",
+                   "2nd Feb 2021"),
+                 order = c("mdy", "dmy"))
[1] "2021-02-02 UTC" "2021-02-02 UTC"
```

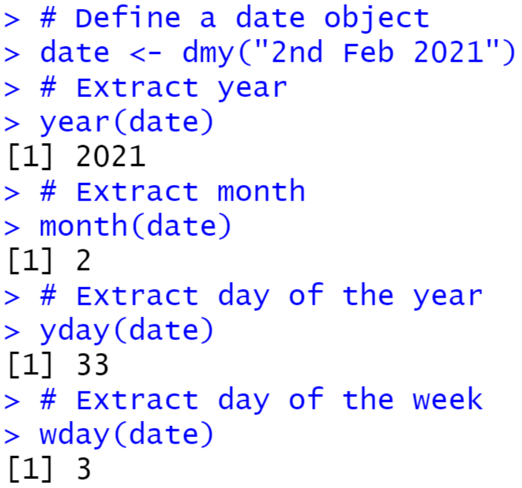
Figure 3: ymd(), mdy(), and parse_date_time() are versatile functions for converting almost any legal Date specification into the universally accepted ISO 8601 format.

Curtin University

RCode: key Date functions in lubridate

```
# Use ymd() for Year, Month, Day format
ymd("2021-02-11")
# Use dmy() for Day, Month, Year format
dmy("11/2/2021")
# Use parse_date_time() for general
   formats
parse_date_time(c("Feb 2nd, 2021",
                  "2nd Feb 2021"),
                order = c("mdy", "dmy"))
```

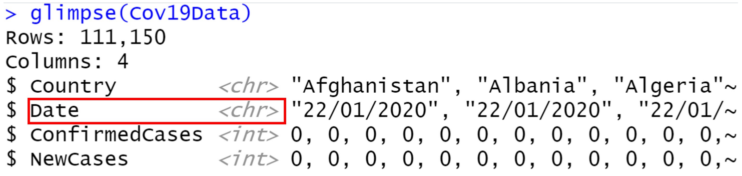
lubridate functions for extracting features



```
> # Define a date object
> date <- dmy("2nd Feb 2021")
> # Extract year
> year(date)
[1] 2021
> # Extract month
> month(date)
[1] 2
> # Extract day of the year
> yday(date)
[1] 33
> # Extract day of the week
> wday(date)
[1] 3
```

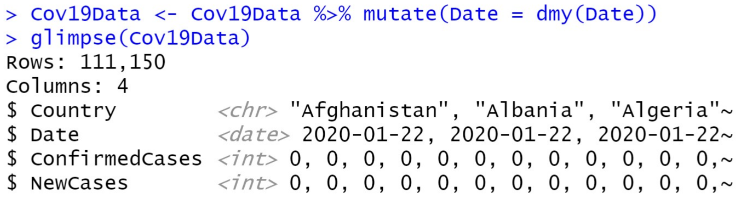
Figure 4: `year()`, `month()`, `yday()`, and `wday()` are great functions to extract useful date related features.

Example: Covid-19 data summary



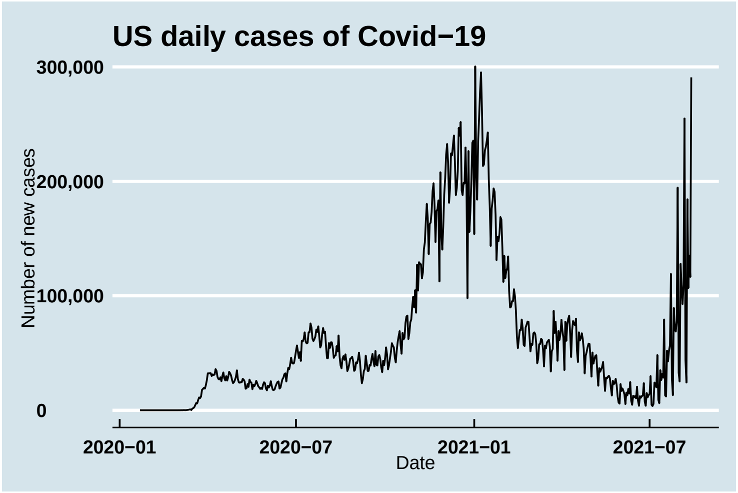
```
> glimpse(Cov19Data)
Rows: 111,150
Columns: 4
$ Country        <chr> "Afghanistan", "Albania", "Algeria"~
$ Date           <chr> "22/01/2020", "22/01/2020", "22/01/~
$ ConfirmedCases <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ NewCases       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
```

We can turn the Date variable into a vector of `Date` elements using the `dmy()` function.



```
> Cov19Data <- Cov19Data %>% mutate(Date = dmy(Date))
> glimpse(Cov19Data)
Rows: 111,150
Columns: 4
$ Country        <chr> "Afghanistan", "Albania", "Algeria"~
$ Date           <date> 2020-01-22, 2020-01-22, 2020-01-22~
$ ConfirmedCases <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ NewCases       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
```

Timeseries plot using `geom_line()`

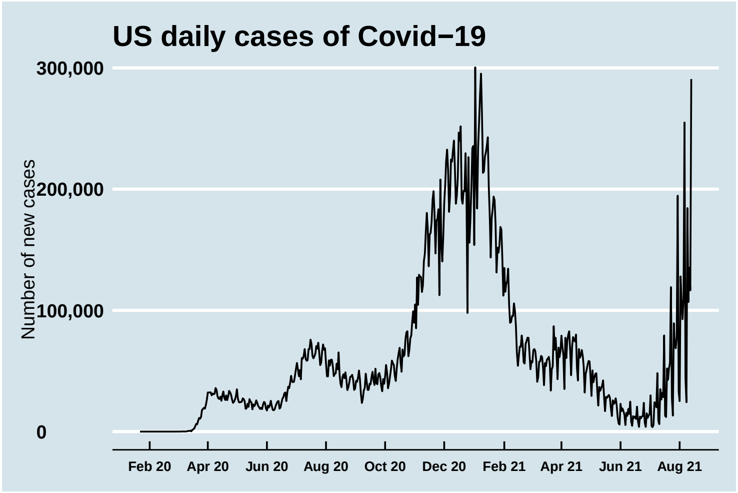


Curtin University

RCode: timeseries plot using geom_line()

```
# Filter USA data
Cov19USA <- Cov19Data %>%
            filter(Country == "US")
# Timeseries plot of Covid cases
ggplot(Cov19USA , aes(Date, NewCases)) +
 geom_line() +
 theme_economist() +
 scale_y_continuous(labels = scales::
    comma) +
 theme(axis.text = element_text(face="
    bold")) +
 labs(y = "Number of new cases")+
 ggtitle("US daily cases of Covid-19")
```

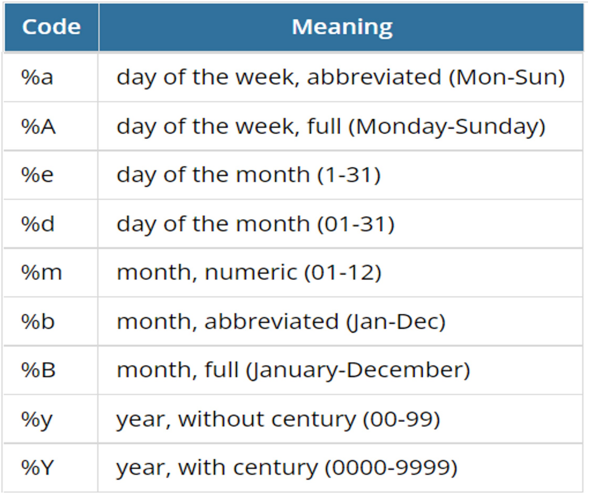
Better labelling of Dates



RCode: better labeling via `scale_x_date()`

```
covidUSA_baseplt +
# Show Month and Year with 2 month
   breaks
scale_x_date(date_breaks = "2 month",
             date_labels = "%b %y")
```

List of Date labels



Code	Meaning
%a	day of the week, abbreviated (Mon-Sun)
%A	day of the week, full (Monday-Sunday)
%e	day of the month (1-31)
%d	day of the month (01-31)
%m	month, numeric (01-12)
%b	month, abbreviated (Jan-Dec)
%B	month, full (January-December)
%y	year, without century (00-99)
%Y	year, with century (0000-9999)

Figure 5: For `date_labels` argument in `scale_*_date()`

Another labelling example of dates

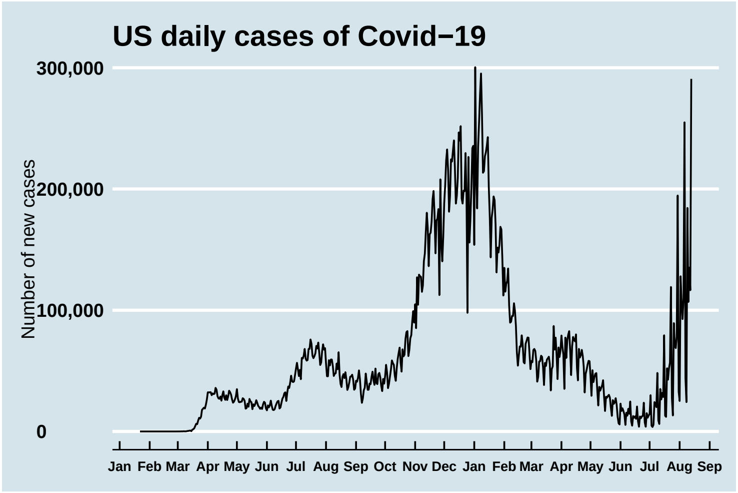


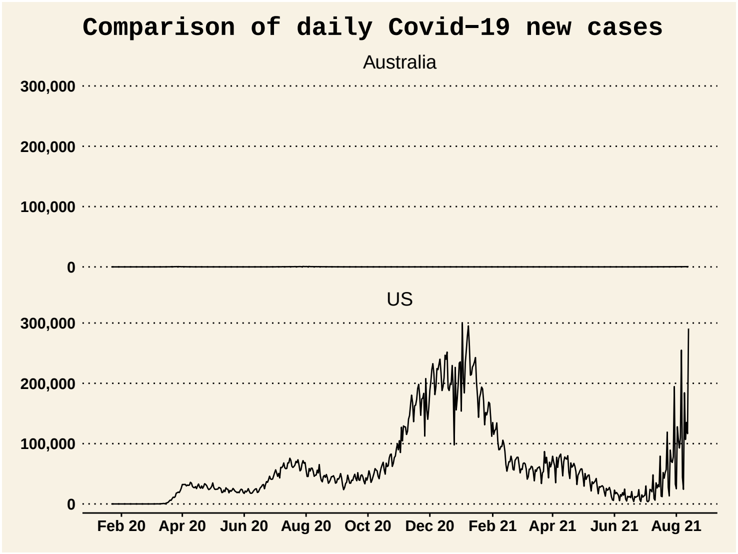
Figure 6: We have introduced monthly labels on the date axis.

RCode: month label using scale_x_date()

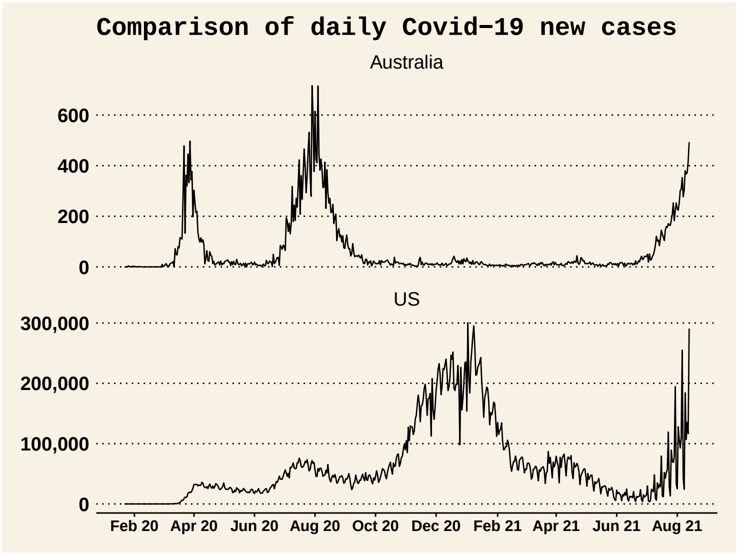
```
covidUSA_baseplt +
# Show Month labels for each month
scale_x_date(date_breaks = "1 month",
             date_labels = "%b")
```

Curtin University

Comparing US and Australia Covid-19 data



Use `scales = "free_y"` in `facet_wrap()`

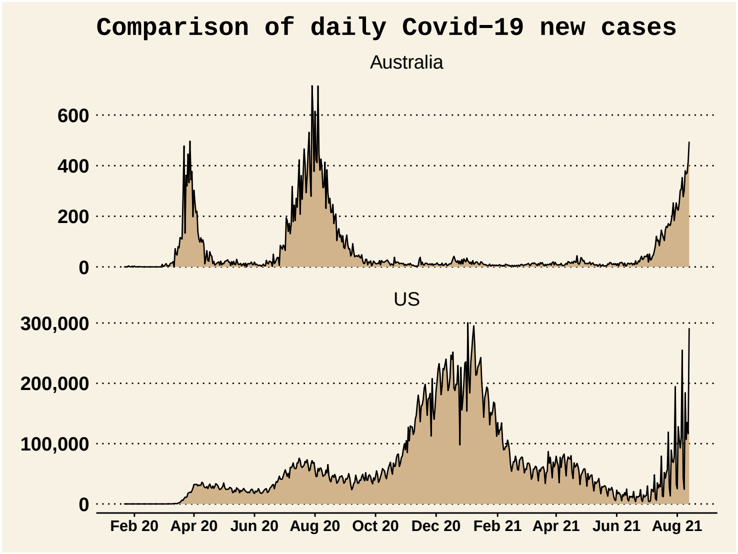


RCode:scales = "free_y" in facet_wrap()

```
####################################
# Default facet_wrap() uses common  #
# scale for both x and y variables  #
####################################
facet_wrap(~ Country,
             nrow = 2)
```

```
####################################
# Use different scales for USA and  #
# Australian daily Covid cases      #
####################################
facet_wrap(~ Country,
             nrow = 2,
             scales = "free_y")
```

Use geom_area() for advanced plotting



Visualisation of Spatial Data

Making maps in R

- To draw a map, we need mainly two things:
  1. The map – polygons constituting a geographical region. For example, the world map with polygons given for each country or the map of USA with polygons given for 51 states in the USA.
  2. The variable – colors or mark to fill in the polygons based on the variable values.
- Typically for a real-life spatial data problem, you should have the shapefile corresponding to your map. Then, the variable values may come in a separate file, and you may have to merge the map and the variables so that you have a proper mapping of variables onto polygons constituting the map/study region.
- Once you map variables onto the polygons, you can create colorful maps known as Choropleths.

Curtin University

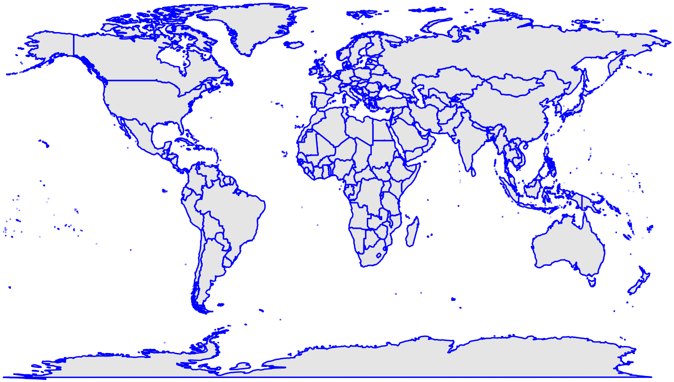
Making maps in R

- To draw a map, we need mainly two things:
  1. The map – polygons constituting a geographical region. For example, the world map with polygons given for each country or the map of USA with polygons given for 51 states in the USA.
  2. The variable – colors or mark to fill in the polygons based on the variable values.
- Typically for a real-life spatial data problem, you should have the shapefile corresponding to your map. Then, the variable values may come in a separate file, and you may have to merge the map and the variables so that you have a proper mapping of variables onto polygons constituting the map/study region.
- Once you map variables onto the polygons, you can create colorful maps known as Choropleths.

Making maps in R

- To draw a map, we need mainly two things:
  1. The map – polygons constituting a geographical region. For example, the world map with polygons given for each country or the map of USA with polygons given for 51 states in the USA.
  2. The variable – colors or mark to fill in the polygons based on the variable values.
- Typically for a real-life spatial data problem, you should have the shapefile corresponding to your map. Then, the variable values may come in a separate file, and you may have to merge the map and the variables so that you have a proper mapping of variables onto polygons constituting the map/study region.
- Once you map variables onto the polygons, you can create colorful maps known as Choropleths.

Plotting the World map

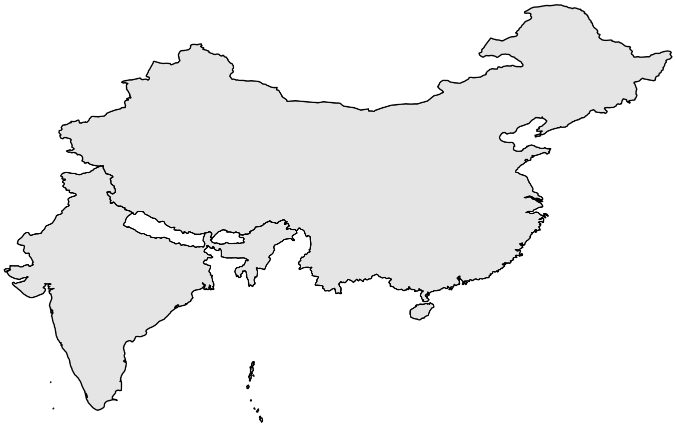


Curtin University

RCode: plotting the World map

```r
library(maps)
# Read in the World map
WorldMap <- map_data("world")
# Plot WorldMap using geom_polygon()
ggplot(WorldMap, aes(long, lat)) +
  geom_polygon(aes(group = group),
               fill="gray90",
               color = "blue") +
  theme_void()
```

Plotting filtered data – India and China

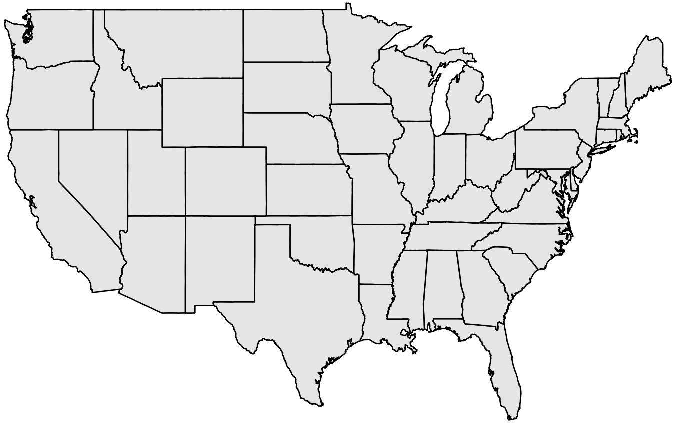


RCode: plotting a subset of the World map

```
# Get the map for India and China
indoChina <- WorldMap %>%
  filter(region %in% c("India",
                       "China"))

# Plot Indo-China region using geom_
   polygon()
ggplot(indoChina, aes(long, lat)) +
  geom_polygon(aes(group = group),
               fill = "gray90",
               color = "black") +
  theme_void()
```

Map of USA states



RCode: plotting the map of USA states

```r
library(maps)
# Read in the USA map
usaMap <- map_data("state")
# Plot the USA map using geom_polygon()
ggplot(usaMap, aes(long, lat)) +
  geom_polygon(aes(group = group),
               fill="gray90",
               color = "black") +
  theme_void()
```

RCode: create data to fill in map polygons

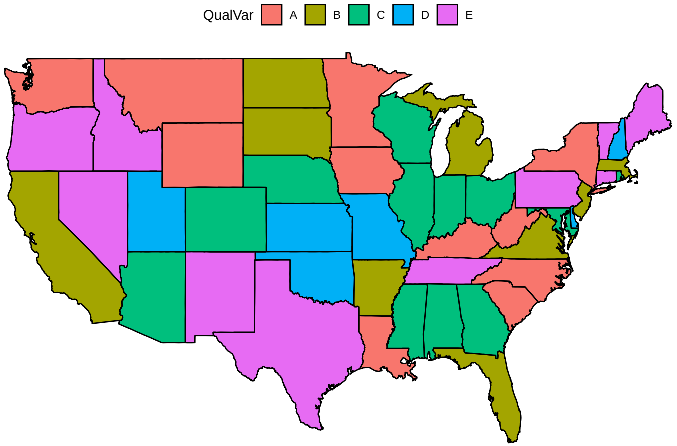
```
# Get all USA states
states <- unique(usaMap$region)
# Create fake data for all states
set.seed(123)
qualVar <- sample(LETTERS[1:5], 49,
                  replace=TRUE)
set.seed(3011)
quantVar <- runif(49, 0, 25)
# Create the data frame
dataForUSMap <- data.frame(
         region = states,
         QualVar = qualVar,
         QuantVar = quantVar)
```

Curtin University

RCode: merge variable data with USA map

```
####################################
# Merge the data frame dataForUSMap #
# with the USA state map data       #
####################################
usaMapMerged <- usaMap %>%
                left_join(dataForUSMap,
                          by = "region")
#######################################
# Choropleth using Qualitative variable#
#######################################
ggplot(usaMapMerged, aes(long, lat,
                    group = group,
                    fill = QualVar)) +
  geom_polygon(color = "black") +
  theme_void()
```

US map: choropleth using qualitative data



Choropleth with Spectral color palette

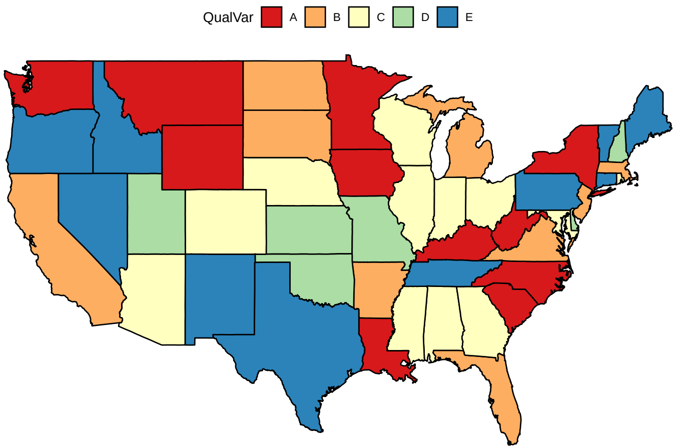
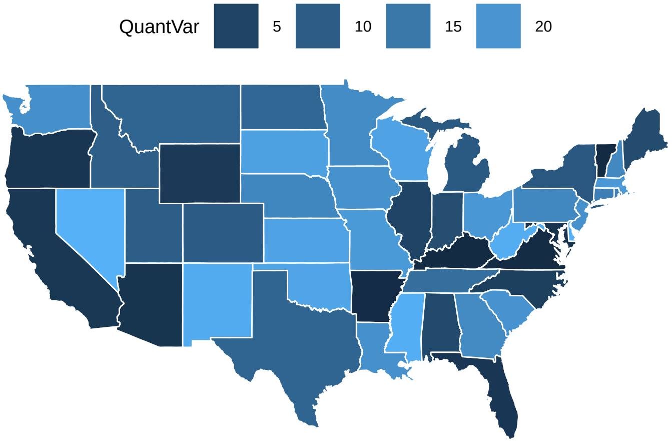


Figure 7: Added `scale_fill_brewer(palette = "Spectral")` with the last plot.

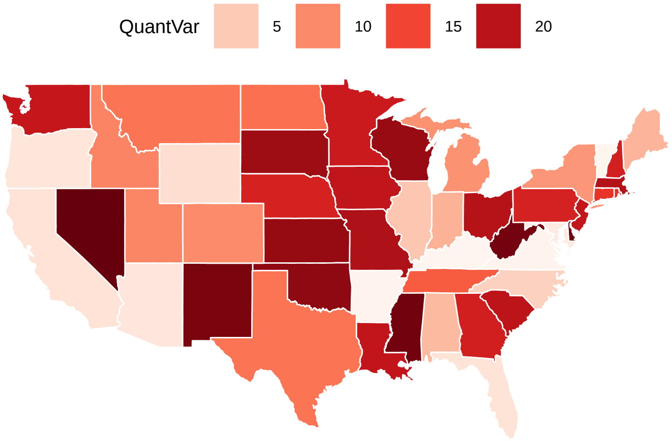
Choropleth using quantitative variable

```
######################################
#Choropleth using Quantitative variable#
######################################
ggplot(usaMapMerged, aes(long, lat,
                    group = group,
                    fill = QuantVar)) +
  geom_polygon(color = "black") +
  theme_void()
```

US map: choropleth using quantitative data

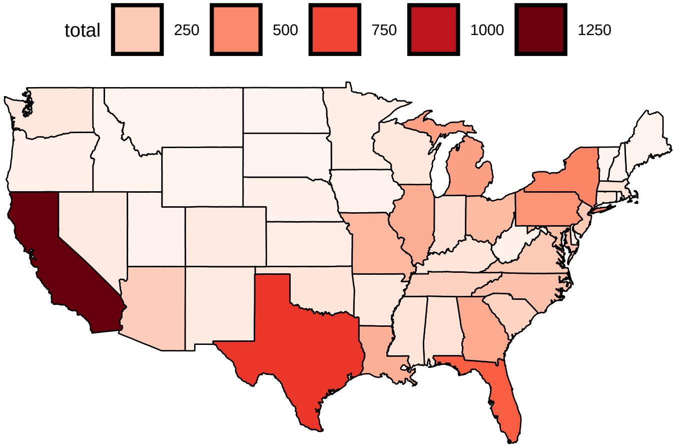


Improved color by `scale_fill_gradientn()`



Curtin University

Real-life data example: `murders` data


total
250
500
750
1000
1250