Explainable artificial intelligence model to predict mortality in patients with suspected acute coronary syndrome

MIMIC III ANALYSIS

Edwin Kagereki

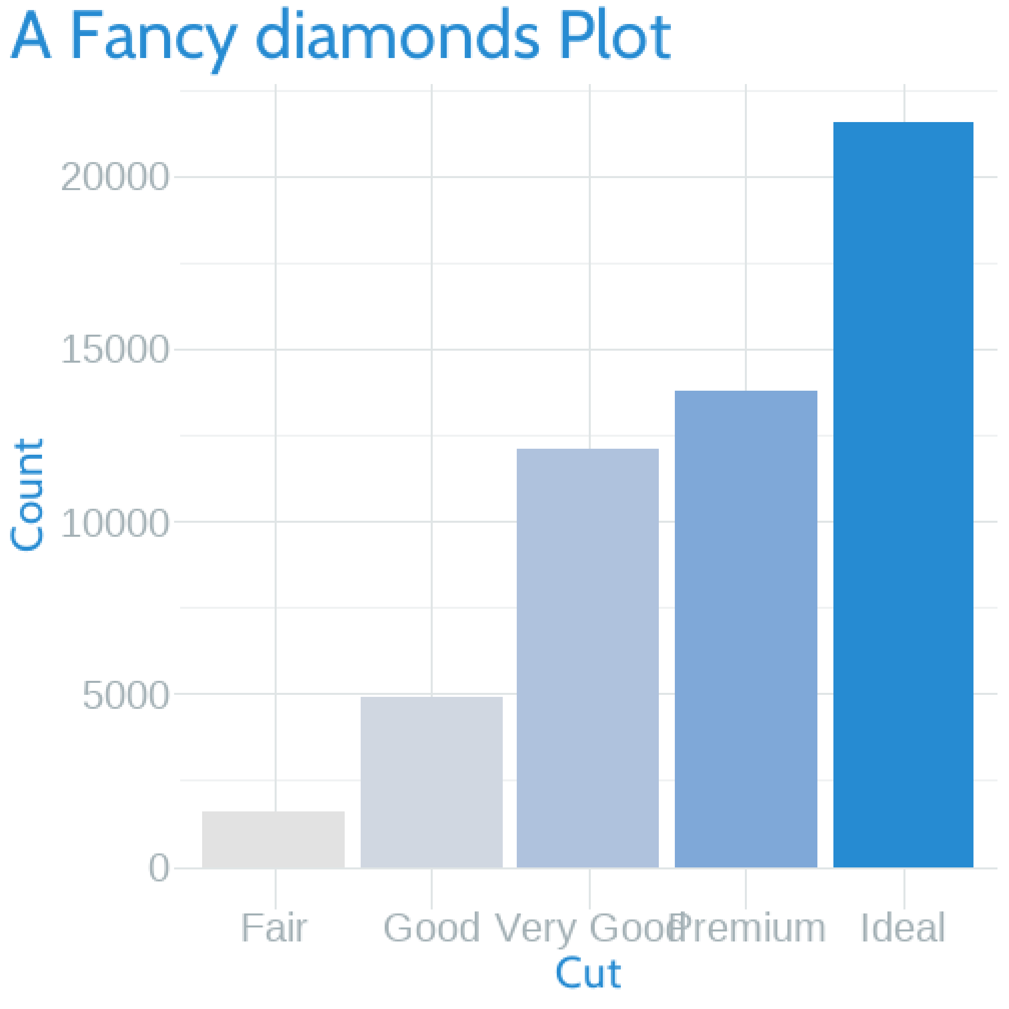
2021/09/20 (Presented: 2021-11-23)

Outline

- Business Understanding
- Data Understanding
- Data Preparation
- Modeling
- Evaluation
- Deployment

Business Understanding

- Cardiovascular diseases (CVDs) are the leading cause of global mortality.
- Prediction of prognosis model for patients with suspected ACS is important in critical care medicine.


A Fancy diamonds Plot
Count
20000
15000
10000
5000
0
Fair
Good
Very Good
Premium
Ideal
Cut

Data Understanding

- MIMIC-III is a large, freely-available database comprising deidentified healthrelated data associated with 46,520 patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012.

- Researchers seeking to use the database must:

1. Become a credentialed user on PhysioNet. This involves completion of a training course in human subjects research.
2. Sign the data use agreement. Adherence to the terms of the DUA is paramount.

- All data was collected downloaded to PostgreSQL database, and SQL queries were thereafter applied.
- Although the database includes 26 tables, only the ADMISSIONS,PATIENTS,SERVICES,DIAGNOSIS_ICD,MICROBIOLOGYEVENTS, PRESCRIPTIONS,PROCEDUREEVENTS_MV,D_ITEMS,D_LABITEMS*

[*] This was enriched with the Loinc tables

Objectives

The aim of the project was:

- This project aims at developing a risk-prediction tool for ACS, focusing on clinical end point of all-cause mortality after one hour of treatment(golden hour) using machine learning classification algorithm.
- Explanation of the predicted outcome based on the care pathway using process mining. This AHA guideline(ACLS 2020) will was used as the gold standard for ACS workflow*.

[*] ACLS guideline

Data preparation

- Data exploration.
- Derived attributes.
- Encoding of categorical variables.
- Data standardization.
- EVent log generation.

Data access

- After connecting to the PostgreSQL database, SQL queries were made using the R dplyr interface.

```r
con <- DBI::dbConnect(RMySQL::MySQL(),
  host = "databaseEndpoint",user = "Kagereki",password = "Pass")
pt<-tbl(con, "ADMISSIONS") %>%
  select(SUBJECT_ID, GENDER, DOB,EXPIRE_FLAG)
admin <- tbl(con, "ADMISSIONS")
admin<-merge(x = admin, y = pt, by = "SUBJECT_ID", all.x = TRUE) %>%
  mutate(AGE = round(as.numeric(difftime(ADMITTIME,DOB, units = "days")/365),0)) %>%
  mutate(LOS = as.numeric(difftime(DISCHTIME,ADMITTIME, units = "hours"))) %>%
  mutate(AGE = ifelse(AGE<300, AGE, AGE-211)) %>%
  mutate(endGoldenHour = ADMITTIME + minutes(60)) %>%
  group_by(SUBJECT_ID) %>%
  mutate(admissionCycle = 1:n()) %>%
  group_by(SUBJECT_ID) %>%
  arrange(DISCHTIME) %>%
  mutate(nAdmissions = n_distinct(HADM_ID)) %>%
  mutate(deadBefore = as.numeric(DEATHTIME-endGoldenHour)) %>%
  filter(deadBefore>=0) %>%
  mutate(DIAGNOSIS2 = tolower(DIAGNOSIS),dayOfYear = yday(ADMITTIME),Month = month(ADMITTIME))
  mutate(week = week(ADMITTIME),weekday = wday(ADMITTIME)) %>%
  mutate(year = year(ADMITTIME),hour = hour(ADMITTIME))
```

Data exploration.

Distribution of patients over time

Derived Attributes

Some considerations were done

Age

1. Computed by subtracting the *DOB* from the *ADMITTIME*.
2. Values above 300 was adjusted by subtracting 211.

Datetime Features

1. Day of the year;
2. Week of the year;
3. Month;
4. Year
5. Hour of the day;

Categorical variable encoding

Some considerations were done

Variables with high cardinality

1. Start with a boilerplate HTML file;
2. Plain Markdown;
3. Write JavaScript to autoplay slides;
4. Manually configure MathJax;
5. Highlight code with `*`;
6. Edit Markdown source and refresh browser to see updated slides;

Variables with less cardinality but need to preserve the variance

1. Start with an R Markdown document;
2. R Markdown (can embed R/other code chunks);
3. Provide an option `autoplay`;
4. MathJax just works;[*]
5. Highlight code with `{{}}`;
6. The RStudio addin "Infinite Moon Reader" automatically refreshes slides on changes;

[*] Not really. See next page.

Modeling

The following candidate models were chosen with the vanila methods

- Logistic Regression
- Random Forest,
- XGBoost (extreme gradient boosted trees),
- K-nearest neighbor
- Artificial Neural network(ANN)

Now we can use our validation set (k folds) to estimate the performance of our models using the fit_resamples() function to fit the models on each of the folds and store the results.

- The metrics used were

Chosen model

The best model was:

Model tuning

Determining the most appropriate cut-off value

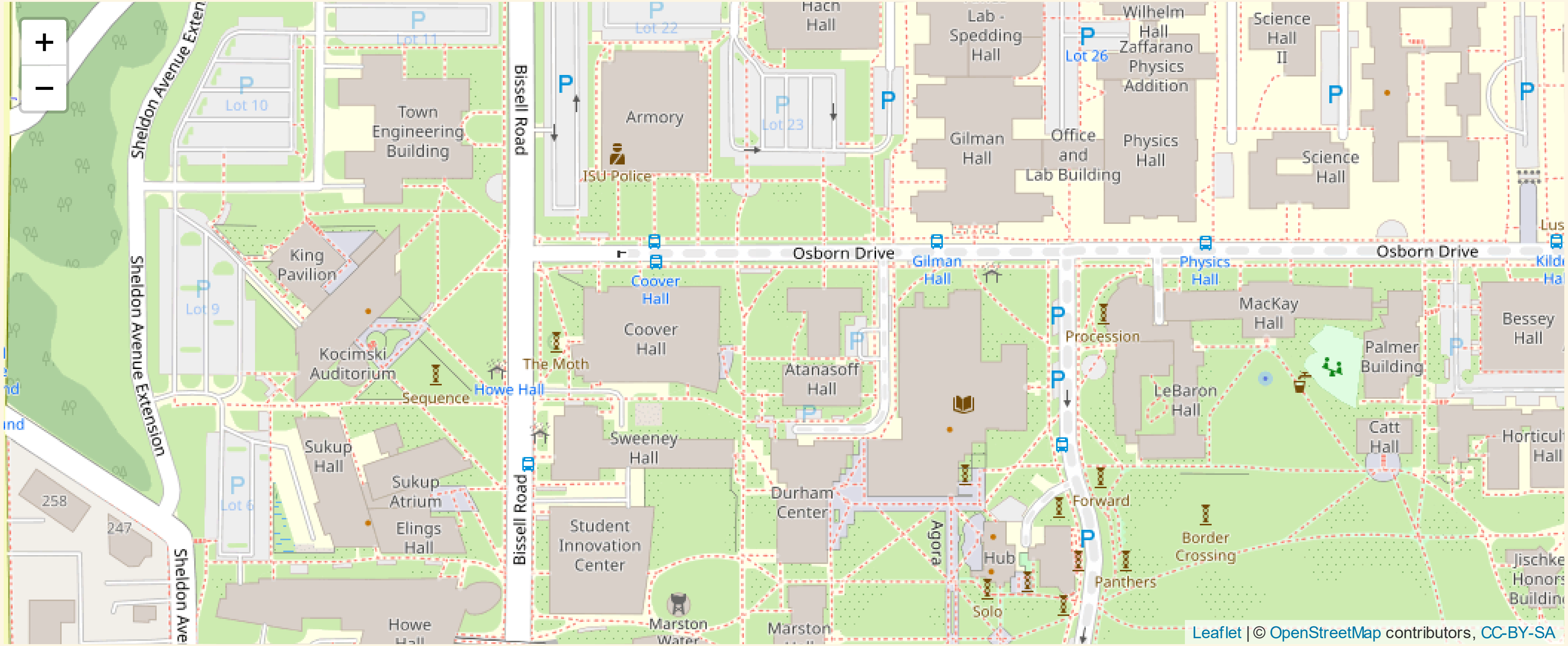
HTML Widgets

I have not thoroughly tested HTML widgets against **xaringan**. Some may work well, and some may not. It is a little tricky.

Similarly, the Shiny mode (`runtime: shiny`) does not work. I might get these issues fixed in the future, but these are not of high priority to me. I never turn my presentation into a Shiny app. When I need to demonstrate more complicated examples, I just launch them separately. It is convenient to share slides with other people when they are plain HTML/JS applications.

See the next page for two HTML widgets.

```r
library(leaflet)
leaflet() %>% addTiles() %>% setView(-93.65, 42.0285, zoom = 17)
```



```r
DT::datatable(
  head(iris, 10),
  fillContainer = FALSE, options = list(pageLength = 8)
)
```

Show 8 entries Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa

Showing 1 to 8 of 10 entries Previous 1 2 Next

Some Tips

- Do not forget to try the `yolo` option of `xaringan::moon_reader`.

```yaml
output:
  xaringan::moon_reader:
    yolo: true
```

Some Tips

- Slides can be automatically played if you set the `autoplay` option under `nature`, e.g. go to the next slide every 30 seconds in a lightning talk:

```
output:
  xaringan::moon_reader:
    nature:
      autoplay: 30000
```

- If you want to restart the play after it reaches the last slide, you may set the sub-option `loop` to TRUE, e.g.,

```
output:
  xaringan::moon_reader:
    nature:
      autoplay:
        interval: 30000
        loop: true
```

Some Tips

- A countdown timer can be added to every page of the slides using the `countdown` option under `nature`, e.g. if you want to spend one minute on every page when you give the talk, you can set:

  ```
  output:
    xaringan::moon_reader:
      nature:
        countdown: 60000
  ```

  Then you will see a timer counting down from `01:00`, to `00:59`, `00:58`, ... When the time is out, the timer will continue but the time turns red.

Some Tips

- The title slide is created automatically by **xaringan**, but it is just another remark.js slide added before your other slides.

  The title slide is set to `class: center, middle, inverse, title-slide` by default. You can change the classes applied to the title slide with the `titleSlideClass` option of `nature` (`title-slide` is always applied).

  ```
  output:
    xaringan::moon_reader:
      nature:
        titleSlideClass: [top, left, inverse]
  ```

- If you'd like to create your own title slide, disable **xaringan**'s title slide with the `seal = FALSE` option of `moon_reader`.

  ```
  output:
    xaringan::moon_reader:
      seal: false
  ```

Some Tips

- There are several ways to build incremental slides. See this presentation for examples.

- The option `highlightLines: true` of `nature` will highlight code lines that start with `*`, or are wrapped in `{{ }}`, or have trailing comments `#<<`;

  ```
  output:
    xaringan::moon_reader:
      nature:
        highlightLines: true
  ```

  See examples on the next page.

Some Tips

An example using a leading `*`:

~~~
```r
if (TRUE) {
* message("Very important!")
}
```
~~~

Output:

```r
if (TRUE) {
  message("Very important!")
}
```

This is invalid R code, so it is a plain fenced code block that is not executed.

An example using `{{}}`:

~~~
```{r tidy=FALSE}
if (TRUE) {
{{ message("Very important!") }}
}
```
~~~

Output:

```r
if (TRUE) {
  message("Very important!")
}
```

```
## Very important!
```

It is valid R code so you can run it. Note that `{{}}` can wrap an R expression of multiple lines.

Some Tips

An example of using the trailing comment `#<<` to highlight lines:

~~~
```{r tidy=FALSE}
library(ggplot2)
ggplot(mtcars) +
  aes(mpg, disp) +
  geom_point() +   #<<
  geom_smooth()    #<<
```
~~~

Output:

```
library(ggplot2)
ggplot(mtcars) +
  aes(mpg, disp) +
  geom_point() +
  geom_smooth()
```

Some Tips

When you enable line-highlighting, you can also use the chunk option `highlight.output` to highlight specific lines of the text output from a code chunk. For example, `highlight.output = TRUE` means highlighting all lines, and `highlight.output = c(1, 3)` means highlighting the first and third line.

~~~
```{r, highlight.output=c(1, 3)}
head(iris)
```
~~~

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Question: what does `highlight.output = c(TRUE, FALSE)` mean? (Hint: think about R's recycling of vectors)

Some Tips

- To make slides work offline, you need to download a copy of remark.js in advance, because **xaringan** uses the online version by default (see the help page `?xaringan::moon_reader`).

- You can use `xaringan::summon_remark()` to download the latest or a specified version of remark.js. By default, it is downloaded to `libs/remark-latest.min.js`.

- Then change the `chakra` option in YAML to point to this file, e.g.

  ```
  output:
    xaringan::moon_reader:
      chakra: libs/remark-latest.min.js
  ```

- If you used Google fonts in slides (the default theme uses *Yanone Kaffeesatz*, *Droid Serif*, and *Source Code Pro*), they won't work offline unless you download or install them locally. The Heroku app google-webfonts-helper can help you download fonts and generate the necessary CSS.

Macros

- remark.js allows users to define custom macros (JS functions) that can be applied to Markdown text using the syntax `![:macroName arg1, arg2, ...]` or `![:macroName arg1, arg2, ...](this)`. For example, before remark.js initializes the slides, you can define a macro named `scale`:

  ```
  remark.macros.scale = function (percentage) {
    var url = this;
    return '<img src="' + url + '" style="width: ' + percentage + '" />';
  };
  ```

  Then the Markdown text

  ```
  ![:scale 50%](image.jpg)
  ```

  will be translated to

  ```
  <img src="image.jpg" style="width: 50%" />
  ```

Macros (continued)

- To insert macros in **xaringan** slides, you can use the option `beforeInit` under the option `nature`, e.g.,

  ```
  output:
    xaringan::moon_reader:
      nature:
        beforeInit: "macros.js"
  ```

  You save your remark.js macros in the file `macros.js`.

- The `beforeInit` option can be used to insert arbitrary JS code before `remark.create()`. Inserting macros is just one of its possible applications.

CSS

Among all options in `xaringan::moon_reader`, the most challenging but perhaps also the most rewarding one is `css`, because it allows you to customize the appearance of your slides using any CSS rules or hacks you know.

You can see the default CSS file here. You can completely replace it with your own CSS files, or define new rules to override the default. See the help page `?xaringan::moon_reader` for more information.

CSS

For example, suppose you want to change the font for code from the default "Source Code Pro" to "Ubuntu Mono". You can create a CSS file named, say, `ubuntu-mono.css`:

```css
@import url(https://fonts.googleapis.com/css?family=Ubuntu+Mono:400,700,400italic);

.remark-code, .remark-inline-code { font-family: 'Ubuntu Mono'; }
```

Then set the `css` option in the YAML metadata:

```yaml
output:
  xaringan::moon_reader:
    css: ["default", "ubuntu-mono.css"]
```

Here I assume `ubuntu-mono.css` is under the same directory as your Rmd.

See yihui/xaringan#83 for an example of using the Fira Code font, which supports ligatures in program code.

CSS (with Sass)

xaringan also supports Sass support via **rmarkdown**. Suppose you want to use the same color for different elements, e.g., first heading and bold text. You can create a `.scss` file, say `mytheme.scss`, using the sass syntax with variables:

```scss
$mycolor: #ff0000;
.remark-slide-content > h1 { color: $mycolor; }
.remark-slide-content strong { color: $mycolor; }
```

Then set the `css` option in the YAML metadata using this file placed under the same directory as your Rmd:

```yaml
output:
  xaringan::moon_reader:
    css: ["default", "mytheme.scss"]
```

This requires **rmarkdown** >= 2.8 and the **sass** package. You can learn more about **rmarkdown** and **sass** support in this blog post and in **sass** overview vignette.

Themes

Don't want to learn CSS? Okay, you can use some user-contributed themes. A theme typically consists of two CSS files `foo.css` and `foo-fonts.css`, where `foo` is the theme name. Below are some existing themes:

```r
names(xaringan::list_css())
```

```
##  [1] "chocolate-fonts"  "chocolate"        "default-fonts"   
##  [4] "default"          "duke-blue"        "fc-fonts"        
##  [7] "fc"               "glasgow_template" "hygge-duke"      
## [10] "hygge"            "ki-fonts"         "ki"              
## [13] "kunoichi"         "lucy-fonts"       "lucy"            
## [16] "metropolis-fonts" "metropolis"       "middlebury-fonts"
## [19] "middlebury"       "nhsr-fonts"       "nhsr"            
## [22] "ninjutsu"         "rladies-fonts"    "rladies"         
## [25] "robot-fonts"      "robot"            "rutgers-fonts"   
## [28] "rutgers"          "shinobi"          "tamu-fonts"      
## [31] "tamu"             "uio-fonts"        "uio"             
## [34] "uo-fonts"         "uo"               "uol-fonts"       
## [37] "uol"              "useR-fonts"       "useR"            
## [40] "uwm-fonts"        "uwm"
```

Themes

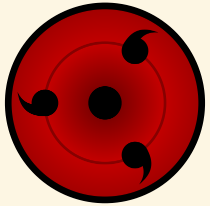
To use a theme, you can specify the `css` option as an array of CSS filenames (without the `.css` extensions), e.g.,

```
output:
  xaringan::moon_reader:
    css: [default, metropolis, metropolis-fonts]
```

If you want to contribute a theme to **xaringan**, please read this blog post.

Naruto

Sharingan



The R package name **xaringan** was derived[1] from **Sharingan**, a dōjutsu in the Japanese anime *Naruto* with two abilities:

- the "Eye of Insight"

- the "Eye of Hypnotism"

I think a presentation is basically a way to communicate insights to the audience, and a great presentation may even "hypnotize" the audience.[2,3]

[1] In Chinese, the pronounciation of *X* is *Sh* /ʃ/ (as in *shrimp*). Now you should have a better idea of how to pronounce my last name *Xie*.

[2] By comparison, bad presentations only put the audience to sleep.

[3] Personally I find that setting background images for slides is a killer feature of remark.js. It is an effective way to bring visual impact into your presentations.

Naruto terminology

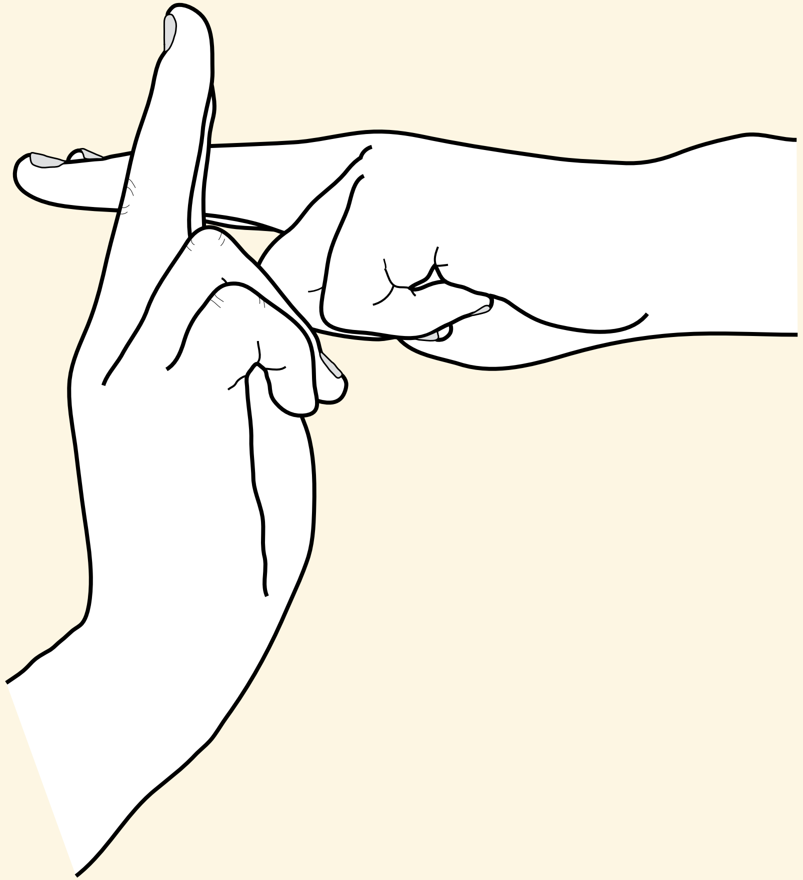
The **xaringan** package borrowed a few terms from Naruto, such as

- Sharingan (写輪眼; the package name)
- The moon reader (月読; an attractive R Markdown output format)
- Chakra (查克拉; the path to the remark.js library, which is the power to drive the presentation)
- Nature transformation (性質変化; transform the chakra by setting different options)
- The infinite moon reader (無限月読; start a local web server to continuously serve your slides)
- The summoning technique (download remark.js from the web)

You can click the links to know more about them if you want. The jutsu "Moon Reader" may seem a little evil, but that does not mean your slides are evil.

Hand seals (印)

Press h or ? to see the possible ninjutsu you can use in remark.js.



Thanks!

Slides created via the R package **xaringan**.

The chakra comes from remark.js, **knitr**, and R Markdown.