**4案例学习：ER损伤**

**4.1简介**

在前三章中，我向您介绍了一系列新概念。因此，为了帮助理解他们，我们现在将介绍一个更丰富的Shiny应用程序，该应用程序探索了一个有趣的数据集，并汇集了您迄今为止看到的许多概念。我们将首先在Shiny之外进行一点数据分析，然后将其转化为一个应用程序，从简单开始，然后逐步对更多细节分层展开。

在本章中，我们将使用vroom(用于快速文件读取)和tidyverse(用于一般数据分析)来帮助Shiny。

library(shiny)

library(vroom)

library(tidyverse)

**4.2数据**

我们将探索由消费品安全委员会收集的国家电子伤害监控系统(NEISS)的数据。这是一项长期研究，记录了在美国具有代表性的医院样本中看到的所有事故。这是一个值得探索的有趣数据集，因为每个人都已经熟悉该领域，每个样本都伴随着一个简短的描述，解释了事故是如何发生的。有关此数据集的更多信息，请访问：<https://github.com/hadley/neiss>。

在本章中，我将只关注2017年的数据。这使得数据足够小(~10 MB)，便于存储在git中(与本书的其余部分一起)，这意味着我们不需要考虑快速导入数据的复杂策略(我们将在本书后面讨论)。您可以在这里看到我用于抽取本章数据的代码<https://github.com/hadley/mastering-shiny/blob/master/neiss/data.R>。

如果您想将数据传输到自己的计算机上，请运行以下代码：

dir.create("neiss")

#> Warning in dir.create("neiss"): 'neiss' already exists

download <- function(name) {

url <- "https://github.com/hadley/mastering-shiny/raw/master/neiss/"

download.file(paste0(url, name), paste0("neiss/", name), quiet = TRUE)

}

download("injuries.tsv.gz")

download("population.tsv")

download("products.tsv")

我们将使用的主要数据集是injuries，其中包含约250000个观察值(样本)：

injuries <- vroom::vroom("neiss/injuries.tsv.gz")

injuries

#> # A tibble: 255,064 × 10

#> trmt\_date age sex race body\_part diag locat…¹ prod\_…² weight narra…³

#> <date> <dbl> <chr> <chr> <chr> <chr> <chr> <dbl> <dbl> <chr>

#> 1 2017-01-01 71 male white Upper Trunk Contus… Other … 1807 77.7 71YOM …

#> 2 2017-01-01 16 male white Lower Arm Burns,… Home 676 77.7 16YOM …

#> 3 2017-01-01 58 male white Upper Trunk Contus… Home 649 77.7 58 YOM…

#> 4 2017-01-01 21 male white Lower Trunk Strain… Home 4076 77.7 21 YOM…

#> 5 2017-01-01 54 male white Head Inter … Other … 1807 77.7 54 YOM…

#> 6 2017-01-01 21 male white Hand Fractu… Home 1884 77.7 21 YOM…

#> # … with 255,058 more rows, and abbreviated variable names ¹​location,

#> # ²​prod\_code, ³​narrative

#> # ℹ Use `print(n = ...)` to see more rows

每行代表一次事故，包含10个变量：

1、trmt\_date是患者在医院的就诊日期(不是事故发生时)。

2、age、sex、race提供了事故发生者的人口统计信息。

3、body\_part是身体上损伤的位置(如脚踝或耳朵)；location是事故发生的地点(如家或学校)。

4、diag提供了损伤(如骨折或撕裂伤)的基本诊断。

5、prod\_ code是与伤害相关的主要产品。

6、weight是统计权重，给出了如果将该数据集扩展到整个美国人口，将遭受这种伤害的估计人数。

7、narrative是一个关于事故发生的简短描述。

我们将它与另外两个data frame配对以获得额外的上下文：products让我们从产品代码中查找产品名称，population告诉我们2017年美国总人口的年龄和性别组合。

products <- vroom::vroom("neiss/products.tsv")

products

#> # A tibble: 38 × 2

#> prod\_code title

#> <dbl> <chr>

#> 1 464 knives, not elsewhere classified

#> 2 474 tableware and accessories

#> 3 604 desks, chests, bureaus or buffets

#> 4 611 bathtubs or showers

#> 5 649 toilets

#> 6 676 rugs or carpets, not specified

#> # … with 32 more rows

#> # ℹ Use `print(n = ...)` to see more rows

population <- vroom::vroom("neiss/population.tsv")

population

#> # A tibble: 170 × 3

#> age sex population

#> <dbl> <chr> <dbl>

#> 1 0 female 1924145

#> 2 0 male 2015150

#> 3 1 female 1943534

#> 4 1 male 2031718

#> 5 2 female 1965150

#> 6 2 male 2056625

#> # … with 164 more rows

#> # ℹ Use `print(n = ...)` to see more rows

**4.3探索**

在创建应用程序之前，让我们先浏览一下数据。我们将从一个有趣的故事开始，看看一个产品：649“厕所”。首先，我们将找出与本产品相关的损伤：

selected <- injuries %>% filter(prod\_code == 649)

nrow(selected)

#> [1] 2993

接下来，我们将对厕所相关损伤的位置、身体部位和诊断进行一些基本总结。请注意，我用weight 变量加权，因此计数可以解释为整个美国的总损伤估计数。

selected %>% count(location, wt = weight, sort = TRUE)

#> # A tibble: 6 × 2

#> location n

#> <chr> <dbl>

#> 1 Home 99603.

#> 2 Other Public Property 18663.

#> 3 Unknown 16267.

#> 4 School 659.

#> 5 Street Or Highway 16.2

#> 6 Sports Or Recreation Place 14.8

selected %>% count(body\_part, wt = weight, sort = TRUE)

#> # A tibble: 24 × 2

#> body\_part n

#> <chr> <dbl>

#> 1 Head 31370.

#> 2 Lower Trunk 26855.

#> 3 Face 13016.

#> 4 Upper Trunk 12508.

#> 5 Knee 6968.

#> 6 N.S./Unk 6741.

#> # … with 18 more rows

#> # ℹ Use `print(n = ...)` to see more rows

selected %>% count(diag, wt = weight, sort = TRUE)

#> # A tibble: 20 × 2

#> diag n

#> <chr> <dbl>

#> 1 Other Or Not Stated 32897.

#> 2 Contusion Or Abrasion 22493.

#> 3 Inter Organ Injury 21525.

#> 4 Fracture 21497.

#> 5 Laceration 18734.

#> 6 Strain, Sprain 7609.

#> # … with 14 more rows

#> # ℹ Use `print(n = ...)` to see more rows

正如你所料，涉及厕所的损伤最常发生在家里。涉及的最常见身体部位可能表明这些是跌倒(因为常规厕所使用通常不涉及头部和面部)，诊断说明之间似乎相当不同。

我们还可以探索年龄和性别的组合模式。我们这里有足够的数据，一张表并没有那么有用，所以我绘制了一个图，图4.1，这使模式更加明显。

summary <- selected %>%

count(age, sex, wt = weight)

summary

#> # A tibble: 208 × 3

#> age sex n

#> <dbl> <chr> <dbl>

#> 1 0 female 4.76

#> 2 0 male 14.3

#> 3 1 female 253.

#> 4 1 male 231.

#> 5 2 female 438.

#> 6 2 male 632.

#> # … with 202 more rows

#> # ℹ Use `print(n = ...)` to see more rows

summary %>%

ggplot(aes(age, n, colour = sex)) +

geom\_line() +

labs(y = "Estimated number of injuries")

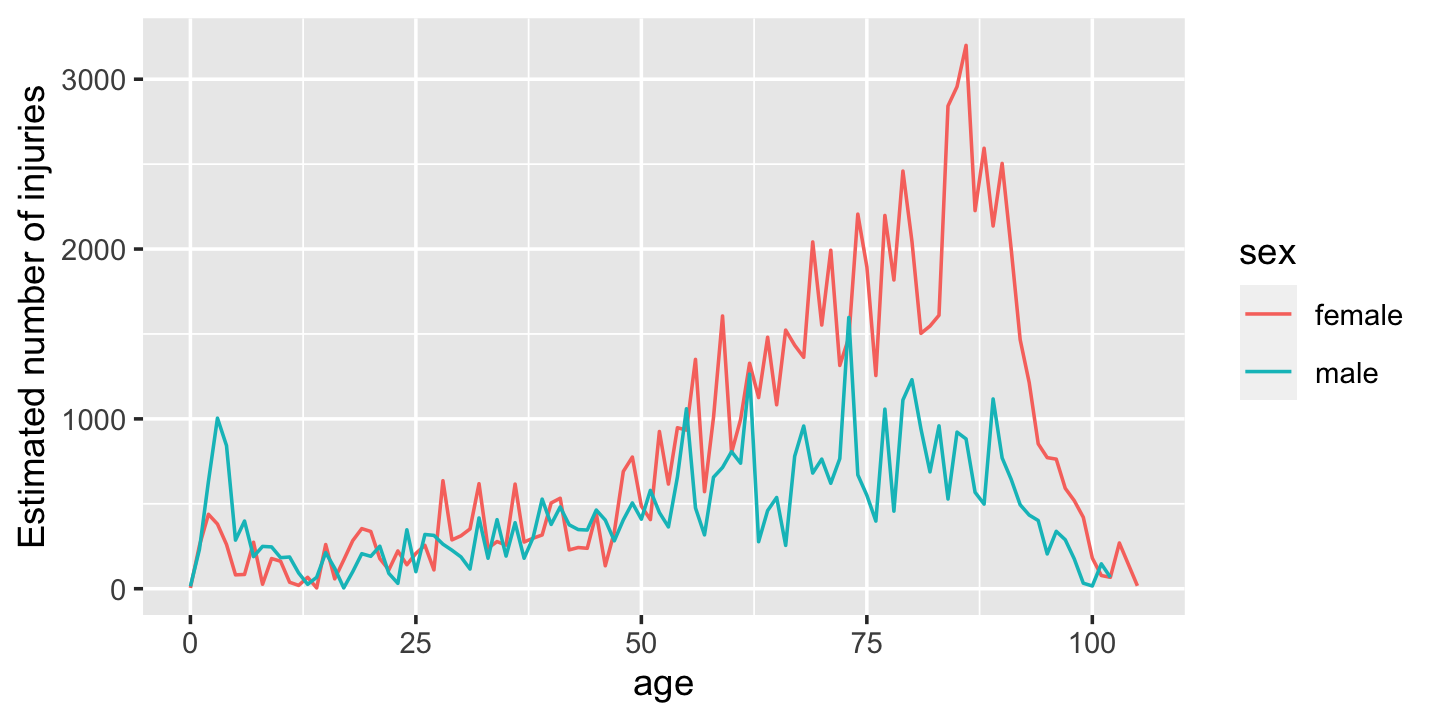


图4.1：按年龄和性别分组估计的厕所伤害人数

我们看到，男孩在3岁时达到峰值，然后在中年左右开始增加(尤其是女性)，80岁后逐渐下降。我怀疑这一峰值是因为男孩通常站着上厕所，而女性的增加是由于骨质疏松症(即，我怀疑女性和男性的受伤率相同，但更多女性最终进入急诊室，因为她们骨折的风险更高)。

解释这种模式的一个问题是，我们知道老年人比年轻人少，因此受伤的人数较少。我们可以通过将受伤人数与总人口进行比较并计算受伤率来控制这一点。这里我使用的是每10000人的受伤率。

summary <- selected %>%

count(age, sex, wt = weight) %>%

left\_join(population, by = c("age", "sex")) %>%

mutate(rate = n / population \* 1e4)

summary

#> # A tibble: 208 × 5

#> age sex n population rate

#> <dbl> <chr> <dbl> <dbl> <dbl>

#> 1 0 female 4.76 1924145 0.0247

#> 2 0 male 14.3 2015150 0.0708

#> 3 1 female 253. 1943534 1.30

#> 4 1 male 231. 2031718 1.14

#> 5 2 female 438. 1965150 2.23

#> 6 2 male 632. 2056625 3.07

#> # … with 202 more rows

#> # ℹ Use `print(n = ...)` to see more rows

图4.2描绘了50岁后的比率，得出了一个截然不同的趋势：男性和女性之间的差异要小得多，我们不再看到下降。这是因为女性往往比男性寿命更长，所以在老年人中，有更多的女性在厕所里受伤。

summary %>%

ggplot(aes(age, rate, colour = sex)) +

geom\_line(na.rm = TRUE) +

labs(y = "Injuries per 10,000 people")

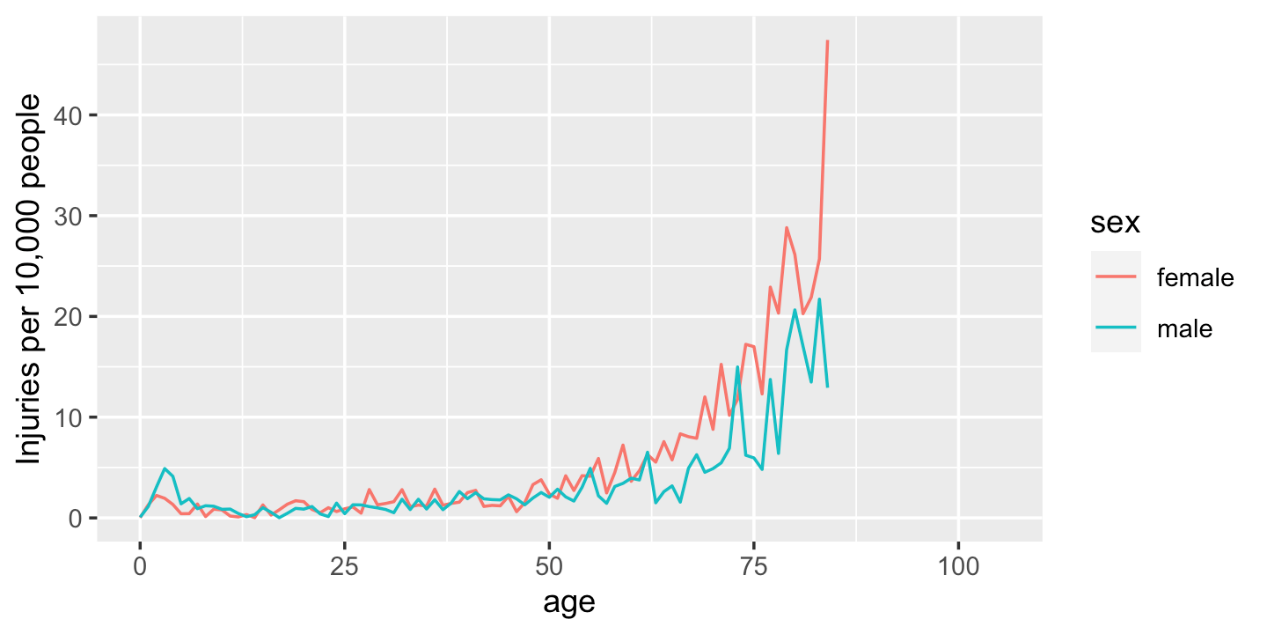


图4.2：按年龄和性别分组的每10000人估计受伤率

(请注意，该比率仅统计到80岁，因为我找不到80岁以上的人口数据。)

最后，我们可以看看一些描述。浏览这些是一种非正式的方式来检验我们的假设，并为进一步的探索提供新的想法。这里我随机抽取了10个样本：

selected %>%

sample\_n(10) %>%

pull(narrative)

#> [1] "6 YOM SLIPPED IN THE BATHROOM @ HOME HIT CHIN ON EDGE OF TOILET DX LACERATION CHIN"

#> [2] "91YOF EVAL OF LOW BK TENDERNESS, 1 HR PTA, PT SITTING ON TOILET, WENT TO STAND &FELL TO FLOOR,LANDED BUTT/LOW BK,NO LOC, DX LOW BK TENDERNESS"

#> [3] "89YOM SITTING ON THE TOILET AND RIGHT HIP PROSTHESIS POPPED OUT OF PLACE DISLOCATED HIP PROSTHESIS"

#> [4] "60YOF AVULSION TOENAIL- MOVE FR WC TO TOILET, FELL ONTO FOOT & TOE"

#> [5] "85YF GETTING UP FROM TOILET WHEN FOOT SLIPPED ONTO THE FLOOR&FELL>>HIP FX"

#> [6] "80YOM-DISLOCATION HIP AFTER FALLING FROM GETTING UP FROM POTTY CHAIR TRANSFERRING TO WHEELCHAIR DX: DISLOCATION"

#> [7] "4 YO M W/LAC TO FOREHEAD SLIPPED IN BATHROOM HIT ON TOILET FLUSH HANDLE"

#> [8] "84YOM UNWITNESSED FALL. HE WAS SITTING ON TOILET AND FOUND ON FLOOR. IMAGING +. DX: IVH OF BRAIN."

#> [9] "80YOM FELL OFF THE TOILET AND SUSTAINED A CONTUSION TO LEFT KNEE"

#> [10] "92YOM FELL OFF THE TOILET STRUCK FACE LACERATION TO FACE"

在对一个产品进行了探索之后，如果我们可以轻松地对其他产品进行探索，而不必重新输入代码，那将是非常好的。所以，让我们制作一个Shiny应用程序！

**4.4原型**

在构建一个复杂的应用程序时，我强烈建议从尽可能简单的开始，这样你就可以在开始做更复杂的事情之前确认基本的机制正确工作。在这里，我将从一个输入(产品代码)、三个表格和一个绘图开始。

在设计第一个原型时，挑战在于使其“尽可能简单”。在快速实现基本功能和规划应用程序的未来之间存在着紧张关系。任何一个极端都可能是不好的：如果你设计得太狭隘，你以后会花很多时间重新设计你的应用程序；如果设计过于严格，您将花费大量时间编写代码，而这些代码最终会被扔在地板上。为了保持平衡，我经常在编写代码之前用铅笔和纸画一些草图来快速探索UI和反应图。

在这里，我决定UI的一行用于输入(假设我可能会在这个应用程序完成之前添加更多输入)，一行用于所有三个表格(每个表格有4列，占屏幕12列宽度的1/3)，然后一行用于绘图：

prod\_codes <- setNames(products$prod\_code, products$title)

ui <- fluidPage(

fluidRow(

column(6,

selectInput("code", "Product", choices = prod\_codes)

)

),

fluidRow(

column(4, tableOutput("diag")),

column(4, tableOutput("body\_part")),

column(4, tableOutput("location"))

),

fluidRow(

column(12, plotOutput("age\_sex"))

)

)

我们还没有讨论fluidRow()和column()，但您应该能够从上下文中猜出它们的作用，我们将在第6.2.3节中再次讨论它们。还要注意在selectInput()选项中使用setNames()：这将在UI中显示产品名称，并将产品代码返回给服务器。

服务器函数相对简单。我首先将上一节中创建的选定变量和汇总变量转换为反应表达式。这是一个合理的一般模式：在数据分析中创建变量，将分析分解为步骤，并避免多次重新计算，而反应表达式在Shiny应用程序中起着相同的作用。

通常，在启动Shiny应用程序之前，花一点时间清理分析代码是个好主意，这样，在增加反应机制的额外复杂性之前，您可以在常规R代码中思考这些问题。

server <- function(input, output, session) {

selected <- reactive(injuries %>% filter(prod\_code == input$code))

output$diag <- renderTable(

selected() %>% count(diag, wt = weight, sort = TRUE)

)

output$body\_part <- renderTable(

selected() %>% count(body\_part, wt = weight, sort = TRUE)

)

output$location <- renderTable(

selected() %>% count(location, wt = weight, sort = TRUE)

)

summary <- reactive({

selected() %>%

count(age, sex, wt = weight) %>%

left\_join(population, by = c("age", "sex")) %>%

mutate(rate = n / population \* 1e4)

})

output$age\_sex <- renderPlot({

summary() %>%

ggplot(aes(age, n, colour = sex)) +

geom\_line() +

labs(y = "Estimated number of injuries")

}, res = 96)

}

请注意，这里并不一定要创建反应表达式summary，因为它仅由单个反应式消费者使用。但是，将计算和绘图分开是一种很好的做法，因为它使应用程序的流程更容易理解，并且在将来更容易扩展。

生成的应用程序的屏幕截图如图4.3所示，源码在https://github.com/hadley/mastering-shiny/tree/master/neiss/prototype.R，在线演示在：<https://hadley.shinyapps.io/ms-prototype/>。

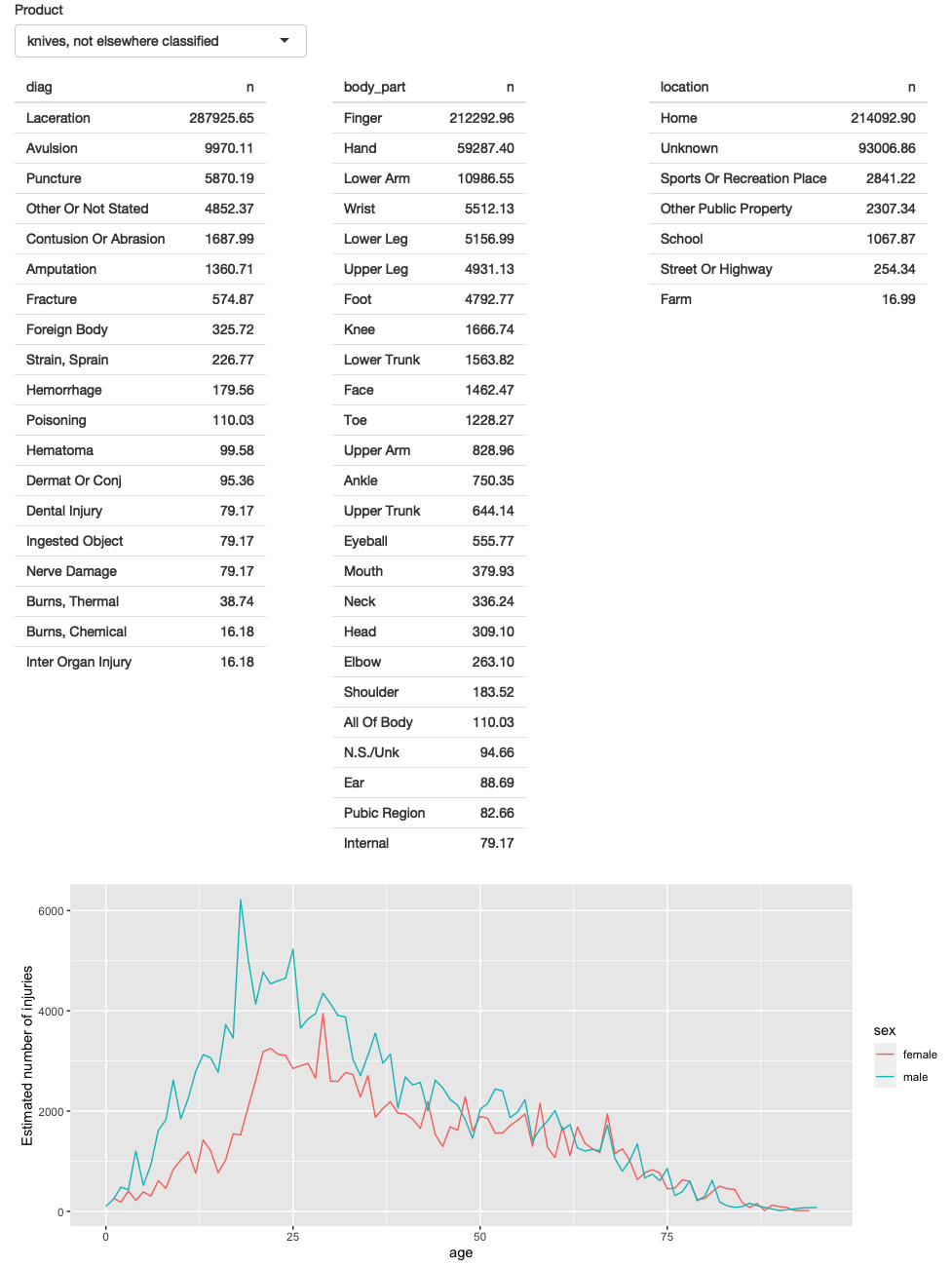


图4.3:NEISS探索应用程序的第一个原型

**4.5对齐表格**

现在我们已经有了基本的组件并正确的工作，我们可以逐步改进我们的应用程序。该应用程序的第一个问题是，它在表格中显示了大量信息，我们可能只想看最重要的。要解决这个问题，我们首先需要弄清楚如何截断表格。我选择了使用forcats函数的组合来实现这一点：我将变量转换为一个因子，按因子水平的频数排序，然后将所有因子的前5个水平合并在一起。

injuries %>%

mutate(diag = fct\_lump(fct\_infreq(diag), n = 5)) %>%

group\_by(diag) %>%

summarise(n = as.integer(sum(weight)))

#> # A tibble: 6 × 2

#> diag n

#> <fct> <int>

#> 1 Other Or Not Stated 1806436

#> 2 Fracture 1558961

#> 3 Laceration 1432407

#> 4 Strain, Sprain 1432556

#> 5 Contusion Or Abrasion 1451987

#> 6 Other 1929147

因为我知道怎么做，所以我写了一个小函数来自动处理任何变量。这里的细节并不重要，但我们将在第12章中再次讨论。您可以通过复制和粘贴来解决问题，因此如果代码看起来完全陌生，也不用担心。

count\_top <- function(df, var, n = 5) {

df %>%

mutate({{ var }} := fct\_lump(fct\_infreq({{ var }}), n = n)) %>%

group\_by({{ var }}) %>%

summarise(n = as.integer(sum(weight)))

}

然后我在服务器函数中使用它：

output$diag <- renderTable(count\_top(selected(), diag), width = "100%")

output$body\_part <- renderTable(count\_top(selected(), body\_part), width = "100%")

output$location <- renderTable(count\_top(selected(), location), width = "100%")

我做了另一个改变来改善应用程序的美观性：我强制所有表格占据最大宽度(即填充它们占用的列)。这使得输出更加美观，因为它减少了表格宽度的偶然变化。

生成的应用程序的屏幕截图如图4.4所示，源码在https://github.com/hadley/mastering-shiny/tree/master/neiss/polish-tables.R，在线演示网址：<https://hadley.shinyapps.io/ms-polish-tables>。

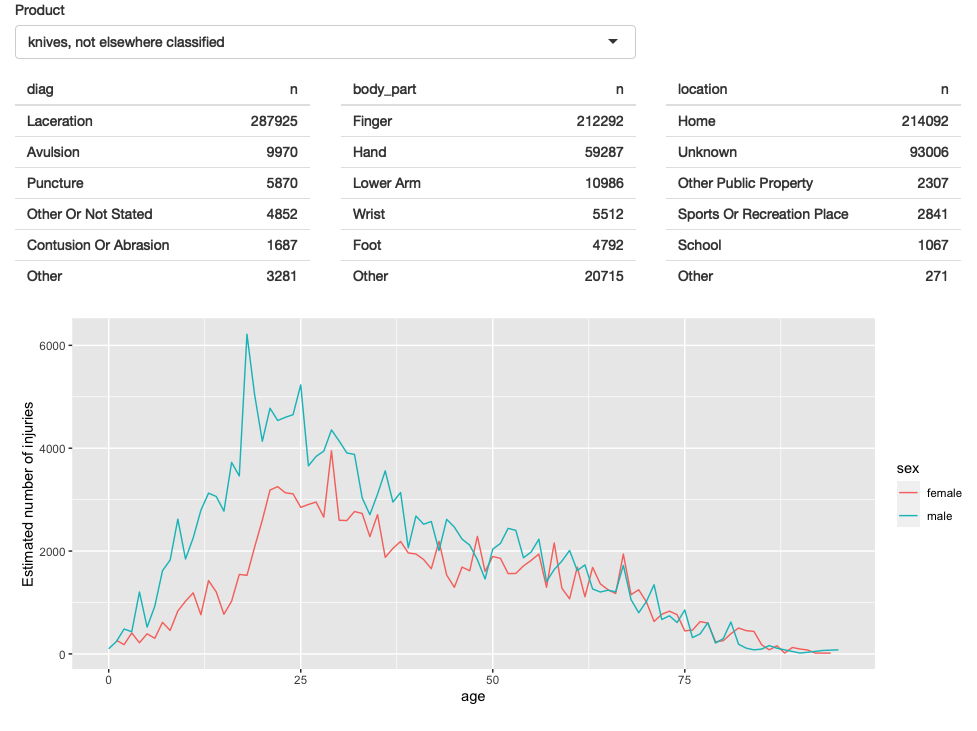


图4.4：应用程序的第二次迭代通过仅显示汇总表中最频繁的行来改进显示

**4.6受伤率与计数**

到目前为止，我们只显示了一个图表，但我们想让用户在可视化受伤人数或人口标准化的受伤率之间做出选择。首先，我向UI添加了一个控件。在这里，我选择使用selectInput()，因为它使两个选项都清楚，而且将来添加新选项也很容易：

fluidRow(

column(8,

selectInput("code", "Product",

choices = setNames(products$prod\_code, products$title),

width = "100%"

)

),

column(2, selectInput("y", "Y axis", c("rate", "count")))

),

(我默认为“受伤率”，因为我认为这样更安全；你不需要了解人口分布就能正确解释该图。)

然后，在生成绘图时，我根据该输入进行条件设置：

output$age\_sex <- renderPlot({

if (input$y == "count") {

summary() %>%

ggplot(aes(age, n, colour = sex)) +

geom\_line() +

labs(y = "Estimated number of injuries")

} else {

summary() %>%

ggplot(aes(age, rate, colour = sex)) +

geom\_line(na.rm = TRUE) +

labs(y = "Injuries per 10,000 people")

}

}, res = 96)

生成的应用程序的屏幕截图如图4.5所示，源码在<https://github.com/hadley/mastering-shiny/tree/master/neiss/rate-vs-count.R>，在线演示在：<https://hadley.shinyapps.io/ms-rate-vs-count>。

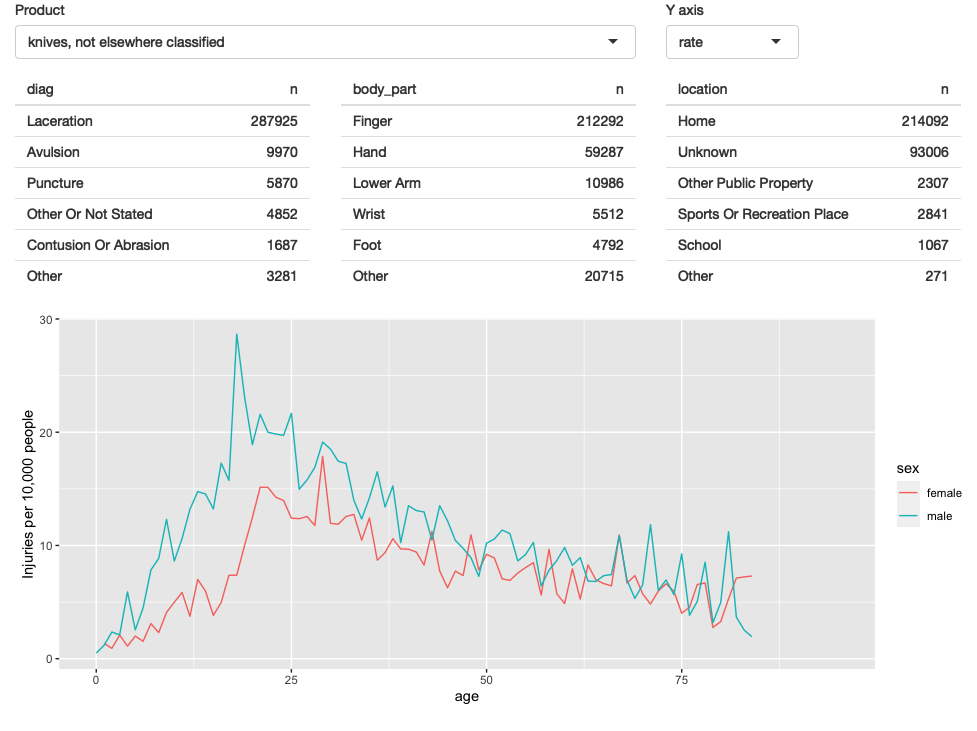


图4.5：在本次迭代中，我们为用户提供了在y轴上显示计数或人口标准化的受伤率之间切换的能力。

**4.7描述**

最后，我想提供一些方法来获取事故的描述，因为它们非常有趣，而且它们提供了一种非正式的方式来交叉检查你在查看分布图时提出的假设。在前面的R代码中，我一次对多个描述进行过采样，但在一个可以交互探索的应用程序中没有理由这样做。

解决方案分为两部分。首先，我们在UI底部添加新行。我使用一个动作按钮来触发一个新的事故描述，并将描述放在textOutput()中：

fluidRow(

column(2, actionButton("story", "Tell me a story")),

column(10, textOutput("narrative"))

)

然后，我使用eventReactive()创建一个仅在单击按钮或底层数据更改时更新的反应表达式。

narrative\_sample <- eventReactive(

list(input$story, selected()),

selected() %>% pull(narrative) %>% sample(1)

)

output$narrative <- renderText(narrative\_sample())

生成的应用程序的屏幕截图如图4.6所示，源码在<https://github.com/hadley/mastering-shiny/tree/master/neiss/narrative.R>，在线演示在：<https://hadley.shinyapps.io/ms-narrative>。

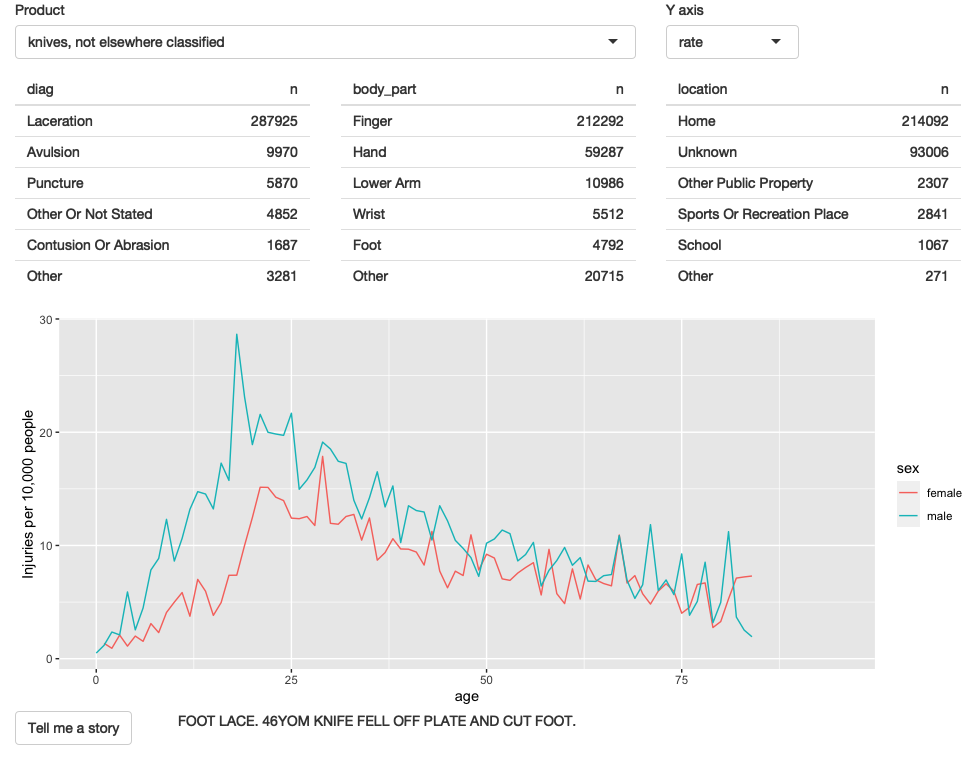


图4.6：最终的迭代增加了从选定行中随机提取事故描述的能力

**4.8练习**

1、绘制每个应用程序的反应图。

2、如果在减少汇总表的代码时调转fct\_inferq()和fct\_ lump()的顺序，会发生什么？

3、添加一个输入控件，以允许用户决定在汇总表中显示多少行。

4、提供一种通过向前和向后按钮全面地逐条浏览每个事故描述的方法。

5、高级：使事故描述列表“循环”，以便从最后一个前进到第一个。

**4.9小结**

现在，您已经掌握了Shiny应用程序的基本知识，下面的七章将为您提供一些重要的技术的百宝袋。阅读下一章关于工作流的内容后，我建议您略读其余章节，以便更好地了解它们所涵盖的内容，然后在应用程序中使用需要的技术时再深入了解。