**第十二章 用R markdown生成深度分析报告**

在数据分析工作中，当人们探索数据、分析数据后，了解掌握了数据的内在规律，接着建立各种模型来概括，包括应用各种回归、聚类等AI机器学习算法，以及深度学习算法等，然后在应用系统中封装部署这些模型，发布应用，用于分类及预测等，比如下面这些例子中的[Shiny App](http://jeanye.cn)。

还有一些情况，深度分析一般要生成一份专业的分析报告，呈交给相关的部门和人员，其中一些报告还是例行的或周期性的，比如月报和季报。这种情况下，通过抽象出通用的报告模板，应用于不同的时间或地区，就比较有用：通过传入不同的时间或地区等参数就可以自动生成不同的报告，这就是自动化。对于高价值的深度分析报告来说，还有一些潜在的质量要求：可重复，可验证，可分享。不可重复不能验证的分析，怎能证明它的可靠与可信？这在科研中是尤其重要的。深度分析报告的这些特性决定了通过脚本来生成（编著）是一个比较好的解决方案。近年来该领域发展了markdown编著系统，用于把大段的论述说明文字，与程序源码及运行结果的图表合并在一起，自动生成图文并茂的分析报告。这和在程序中写少量的注解，然后另外写文档说明，然后把程序、说明和运行结果再人工编辑组合的传统工作方式截然不同。不同的应用场景需要不同的工具和生态。

目前比较流行的，Python生态下有Jupyter Notebook，R语言生态下有R markdowm，R markdown里也支持使用Python语言，它们都基于共同的markdown语法。其实在网络上到处都可以见到合并代码、说明和输出的markdown语言，比如stackoverflow，要提交一个问题寻求帮助，就要用markdown语法把问题的说明，复现用例代码及代码输出合并到一个帖子里。

因为本书用Shiny来封装发布各个模型，配合用R markdown来生成深度分析报告是比较合适的，况且还可以直接用Python代码呢。Python也好，R也好，都是工具，重要的是业务和模型，喜欢用什么都可以，自己用着顺手就好了。应用在一种语言上实现了，移植到另一种语言上也是不难的。世界上有中英俄西法德意等很多种语言，大家和谐共存不好么？事实上谁也无法抹去谁吧。君子求同存异，和而不同。

有几种R markdown深度分析报告的应用场景：

1、交互式 Rmd，提供交互式网页界面，嵌入Shiny App，以及普通的R代码块及输出。动态交互式报告，这种情况直接提供一个URL去访问，比如Shiny Server sample-apps中的[Rmd实例](http://jeanye.cn:3838/sample-apps/rmd/)，它不提供PDF或Word报告下载，不过也可以另存为HTML报告并打印。

2、非交互式Rmd，嵌入普通的R代码块及输出，后台运行，渲染输出PDF或Word报告提供下载，便于打印与传阅，这就是普通的Rmd。比如《Mastering Shiny》一书中的[下载报告](https://mastering-shiny.org/action-transfer.html" \l "downloading-reports)例子，通过Shiny App的交互式界面设定参数，并传递给Rmd报告执行，产生的也是动态交互式报告。

3、非交互式Rmd，嵌入Shiny App以及普通的R代码块及输出，后台运行，渲染输出PDF或Word报告提供下载。这是前两种情况的组合，需要在后台运行Rmd渲染，并且在这个后台运行的Rmd中，后台调用其它Shiny App，传入参数，不显示界面，后台截图。这种情况要复杂一点，需要用到《Mastering Shiny》一书[《Testing》](https://mastering-shiny.org/scaling-testing.html)一章的技术，后台去运行Shiny App。

本章在前面章节开发模型的基础上，探讨一下前两种场景下R markdown深度分析报告的开发，篇幅所限，第三种场景就不深入展开了。

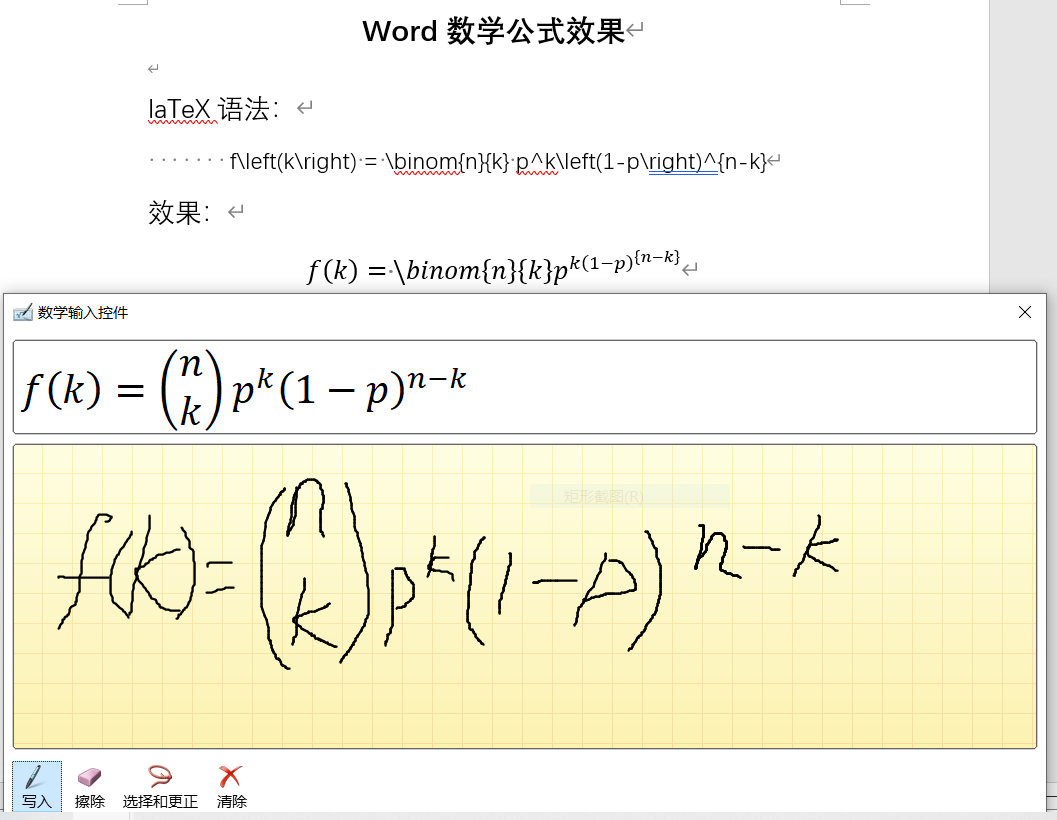
第一节 R markdown运行环境安装配置

R markdown报告是先写好 R markdown脚本，然后渲染输出成HTML、Word、PDF、epub等多种格式。像Java程序一样，写一次，跨平台，多格式输出。

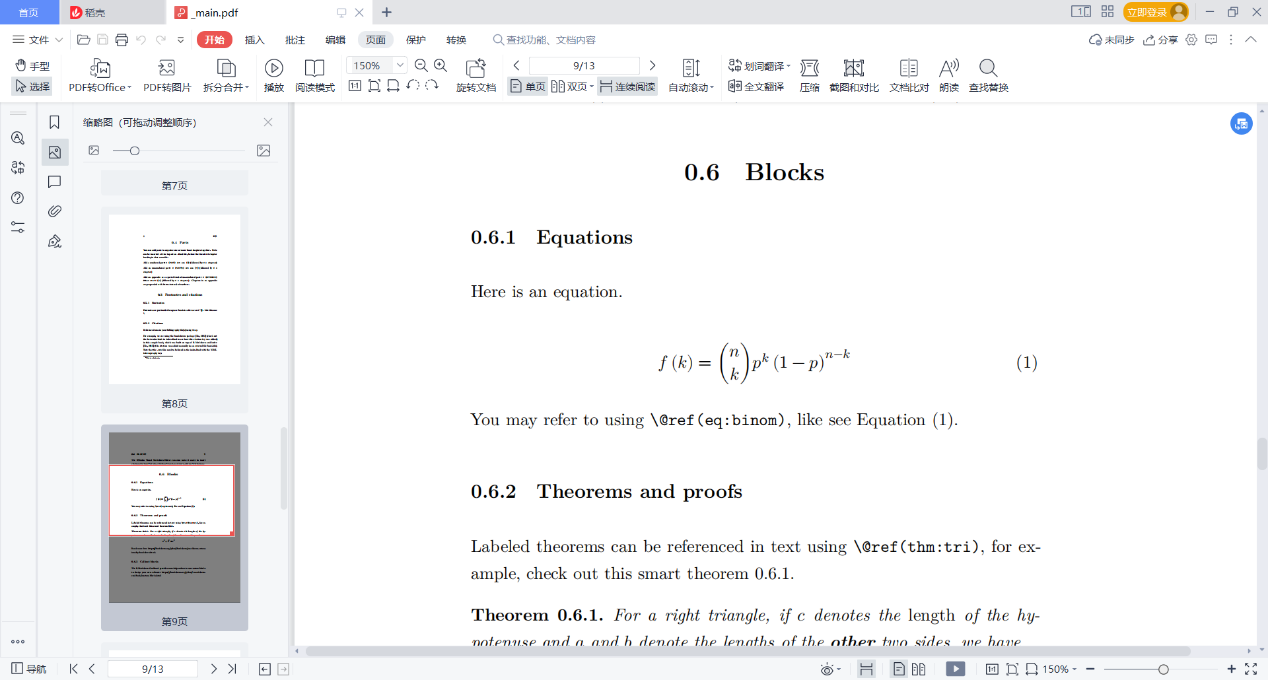
R markdown报告中文的处理，在渲染成PDF时，要做一些安装配置。PDF格式的优势是印刷质量高，尤其适用于有大量数学公式、化学式等的科技书籍和文章。数学公式主要是通过LaTeX语法来定义，在PDF中是原生的，Word中也是用LaTeX语法来输入和表示。Word与PDF的不同在于，Word是即时编译渲染，所见即所得；PDF是先写好脚本，再整个脚本编译渲染，R markdown -> LaTeX -> TeX脚本编译转换，再渲染成PDF。

1、渲染输出效果

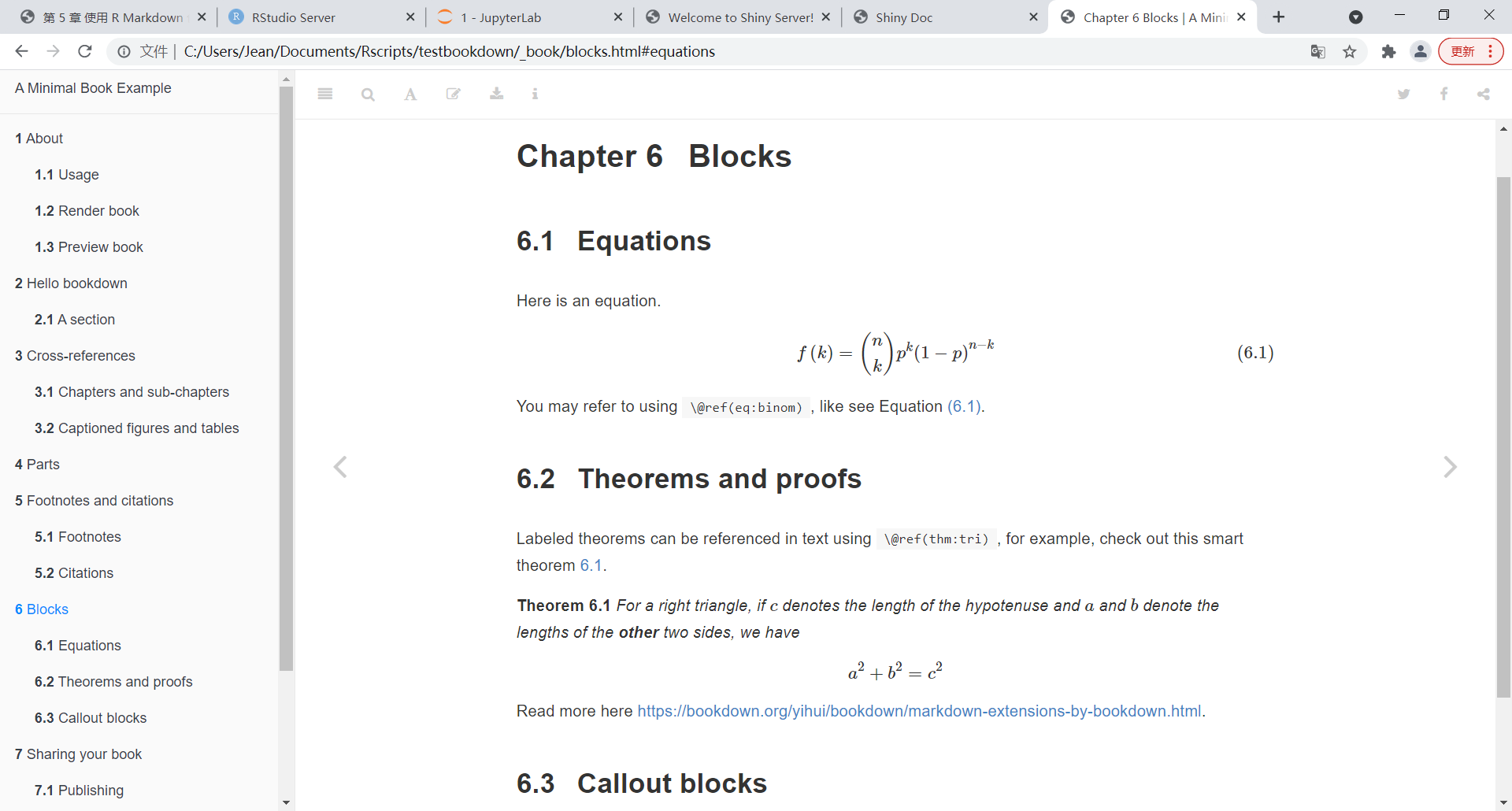
1）Word。Word提供了LaTeX语法输入和墨迹（鼠标手写）两种途径来输入公式，还有一些预定义的公式，所见即所得。



2）PDF。PDF是写好整个R markdown脚本后再编译渲染，公式渲染效果较好，无缝跨越Windows、Linux、Mac等各平台。目前大部分科技书籍与文章都是以PDF格式付印或发行。



3）HTML。渲染输出成HTML网页的效果，通过[MathJax JavaScript库](https://www.mathjax.org/)实现浏览器对LaTeX公式渲染的支持。



2、安装R 软件包

要想使用 R Markdown 输出 PDF 格式的文档，必须事先安装 [LaTeX](https://www.latex-project.org/get/)。LaTeX是[TeX](https://www.tug.org/texlive/)的封装，主要的设计目标是分离内容与格式，以便作者能专注于内容创作而非版式设计，LaTeX 使用 TeX作为它的排版（格式）引擎，R markdown通过[tinytex](https://yihui.org/tinytex/)包使用LaTeX，渲染PDF时把Rmd脚本编译为TeX脚本并执行，所以需要先安装配置它们。LaTeX 有很多发行版，如 MiKTeX，MacTeX， TeX Live 等，R Markdown 的用户可以使用 [TinyTeX](https://github.com/yihui/tinytex-releases/releases)。在Rstudio中运行下面的安装语句：

# 安装 tinytex

install.packages(c(“rmarkdown”, “bookdown”, “tinytex”))

# 安装 TinyTex 套件

tinytex::install\_tinytex()

LaTeX 宏包（也被称为“package”）是 LaTeX 的组件，数目非常多，可以用来扩展 LaTeX 的功能。tinytex::install\_tinytex() 默认只安装了必须的一些组件，在实际使用过程中经常会出现缺少组件的错误。

如果知道宏包的名称，那么可以直接使用下面的命令安装。这里要注意的是，对于国内的用户来说，通常需要设定一下 LaTeX 镜像的位置。下面的例子中，使用了清华大学的 TeX Live 镜像。

# 安装需要的 LaTeX 组件

tinytex::tlmgr\_repo(url = "https://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/tlnet")

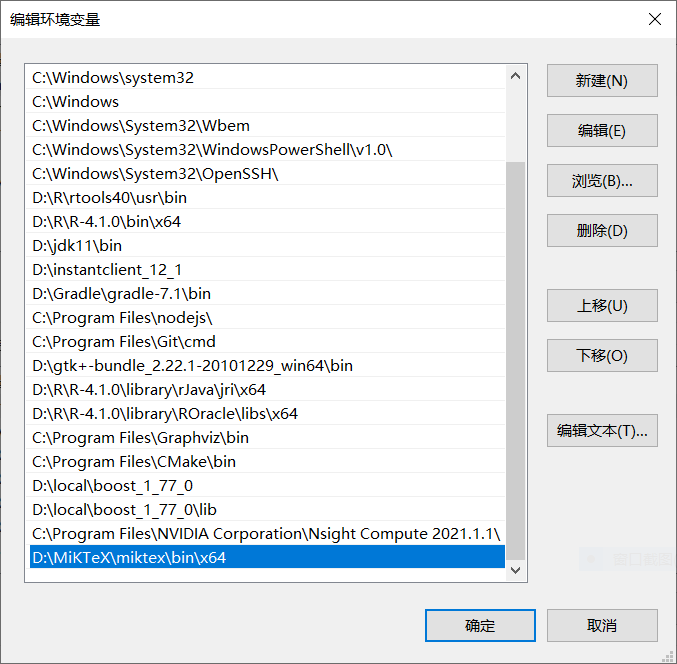
tinytex::tlmgr\_install(c('pgf', 'preview', 'xcolor'))

注意TinyTeX 与 tinytex 并不是一个东西。前者是一个 LaTeX 发行版，后者是一个用来安装和维护前者的 R 语言软件包。不过在本章中，在Windows平台将使用MiKTeX，Linux平台上将使用TeX Live。前者安装宏包比较方便，后者已经比较全了。

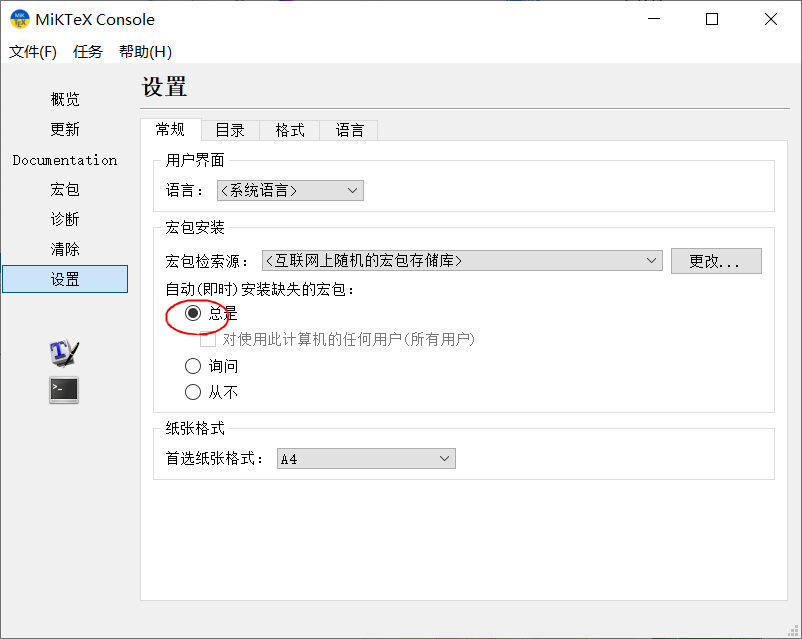
3、Windows上安装配置MiKTeX

MiKTeX是Windows上比较流行的TeX发行版，安装配置简单，可以从这里[下载](https://miktex.org/download)，安装说明参阅该[文档](https://miktex.org/howto/install-miktex)，按向导执行即可。它的安装文件与安装目录大概需要1G。需要注意安装后的两个设置：

1）在系统环境变量path中加入xelatex.exe等可执行文件的安装路径，后面将使用xelatex引擎来处理中文。



2）打开MiKTeX Console，更改宏包安装选项为“总是”自动安装缺失的宏包，否则使用支持中文的ctex宏包时找不到，会出错提示"GUI framework cannot be initialized"，参阅该[Issue文档](https://github.com/seebk/LaTeXText/issues/31)。



3)测试与中文处理

A、测试MiKTeX

新建一个utf8编码文本文件test.tex如下：

\documentclass{article}

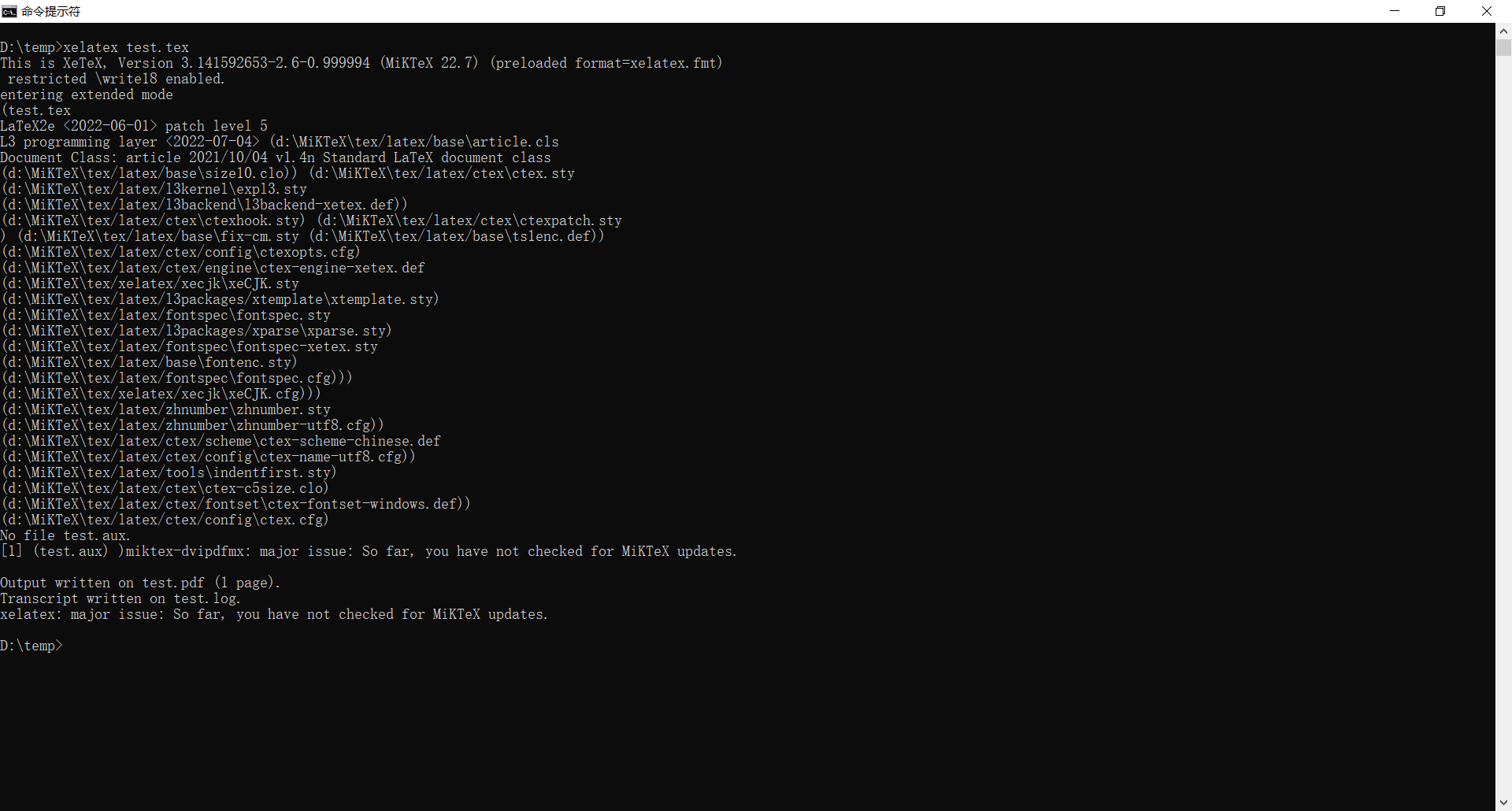
\usepackage{ctex}

\begin{document}

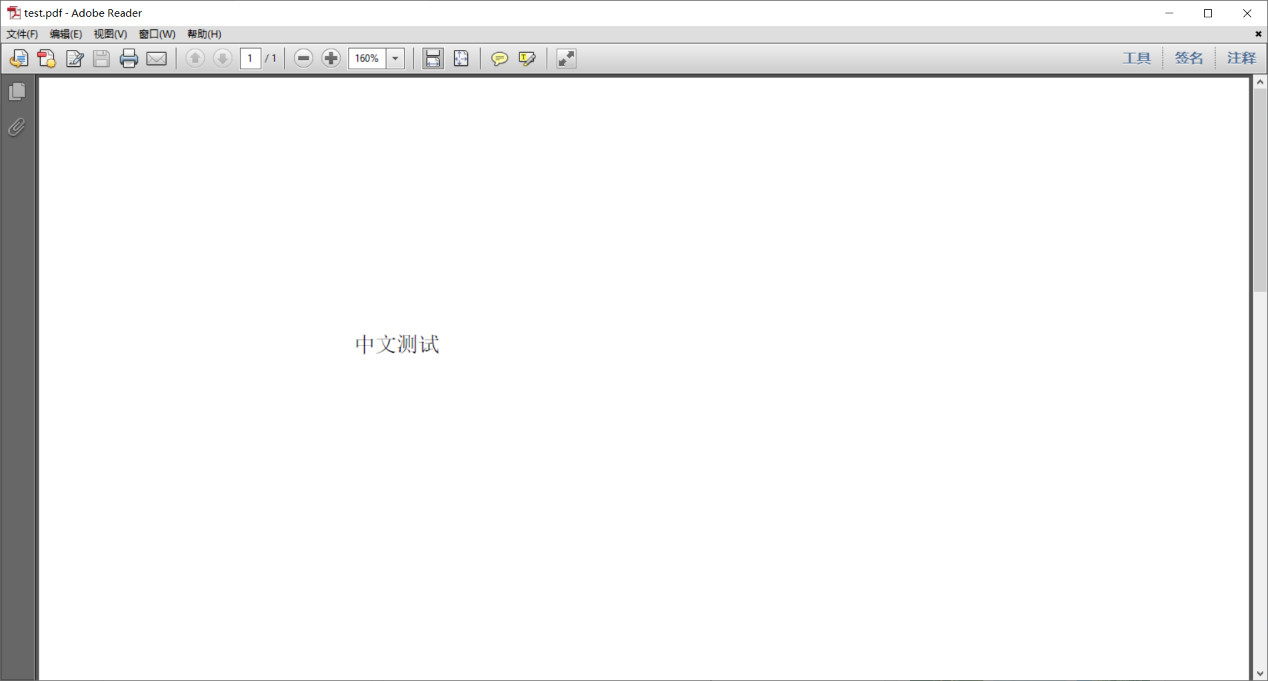
中文测试

\end{document}

B、在命令行窗口中运行xelatex test.tex生成PDF，第一次运行时需要的时间要长一点，因为要连线下载ctex相关的包：



C、查看生成的PDF，测试通过，安装成功。看到上面最后输出：Output written on test.pdf (1 page)就表示成功生成了PDF文件。



D、测试rmarkdown。在Rstudio中建立一个utf8编码的testPDF.Rmd文件，然后用rmarkdown::render()函数直接渲染它，如图所示。注意测试用例的写法与上面例子的不同，在Rmd脚本头部的YAML定义中，使用ctex宏包来处理中文，然后用xelatex引擎处理，它比pdflatex引擎更适合处理中文等unicode字符集。具体可以参阅该[文档](https://github.com/yufree/democode/tree/master/cjk)，那里有各种中日韩字符处理的例子。

---

title: "中文"

author: "Yufree"

date: "2016年9月19日"

header-includes:

- \usepackage{ctex}

output:

pdf\_document:

latex\_engine: xelatex

---

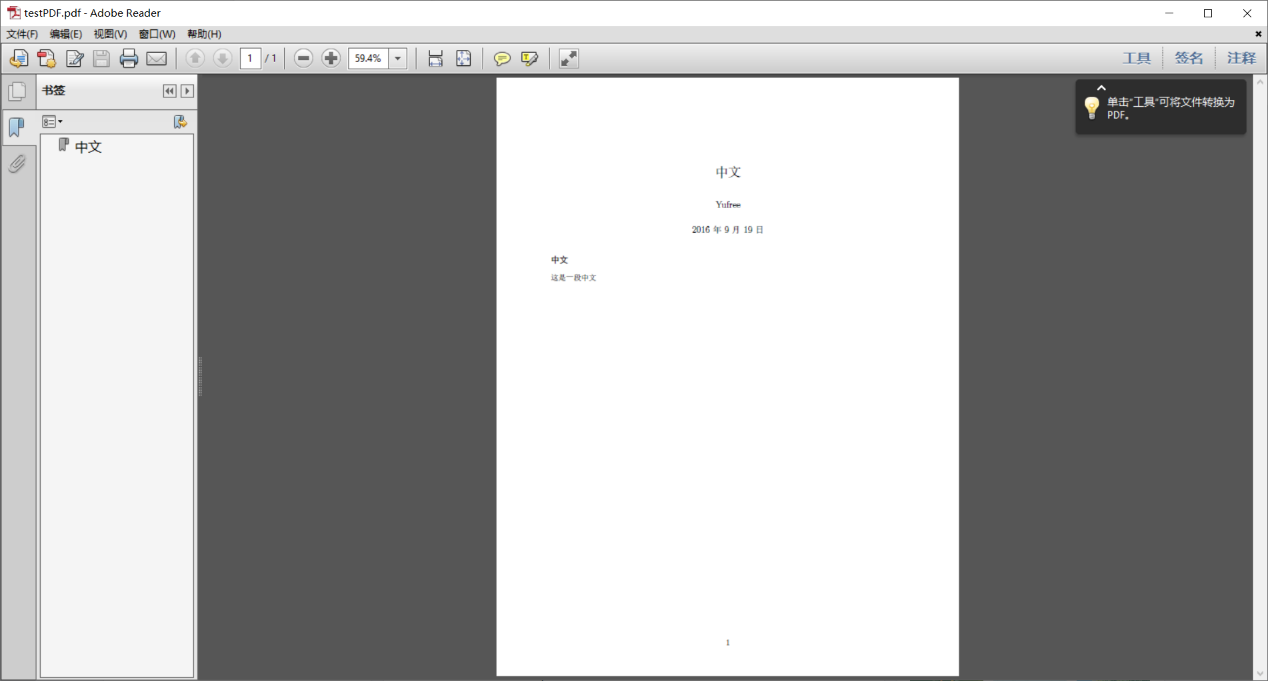
## 中文

这是一段中文

在Rstudio Console中执行下面的语句，渲染生成PDF输出。

rmarkdown::render(“testPDF.RMD”,“pdf\_document”)

E、查看输出的pdf文件，测试通过。



4、Linux上安装配置TeX Live

Linux平台上比较流行的TeX发行版是[texlive](https://www.tug.org/texlive/)，我在云端虚拟主机上装的是CentOS 7.6，它所带的texlive版本比较旧，没有处理中文的ctex等宏包（参阅[文档](http://t.zoukankan.com/raisok-p-12094705.html)），因此下载最新的texlive-2022镜像安装完整的版本，参阅[文档](https://zhuanlan.zhihu.com/p/297891593)及[清华大学最新镜像地址](https://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/Images/)。镜像文件大约4.3G，安装另需6.8G。

1）安装texlive

如果安装了老版本的texlive，要先删除干净，参阅[文档](http://t.zoukankan.com/raisok-p-12094705.html)与[文档2](https://github.com/rstudio/docker-r-session-complete/issues/64)(把Rstudio自带的pandoc加入搜索路径，以便命令行状态下可用)。

删除旧版：

# yum erase texlive\\*.\*

下载：

# cd /root

# wget https://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/Images/texlive2022-20220321.iso

挂载：

# mount -o loop texlive2022-20220321.iso /mnt/

安装：

# cd /mnt

# ./install-tl

输出环境变量：

# vi /etc/profile

# Added for texlive

export MANPATH=${MANPATH}:/usr/local/texlive/2022/texmf-dist/doc/man

export INFOPATH=${INFOPATH}:/usr/local/texlive/2022/texmf-dist/doc/info

export PATH=${PATH}:/usr/local/texlive/2022/bin/x86\_64-linux:/usr/lib/rstudio-server/bin/pandoc

重启系统或source激活环境变量。

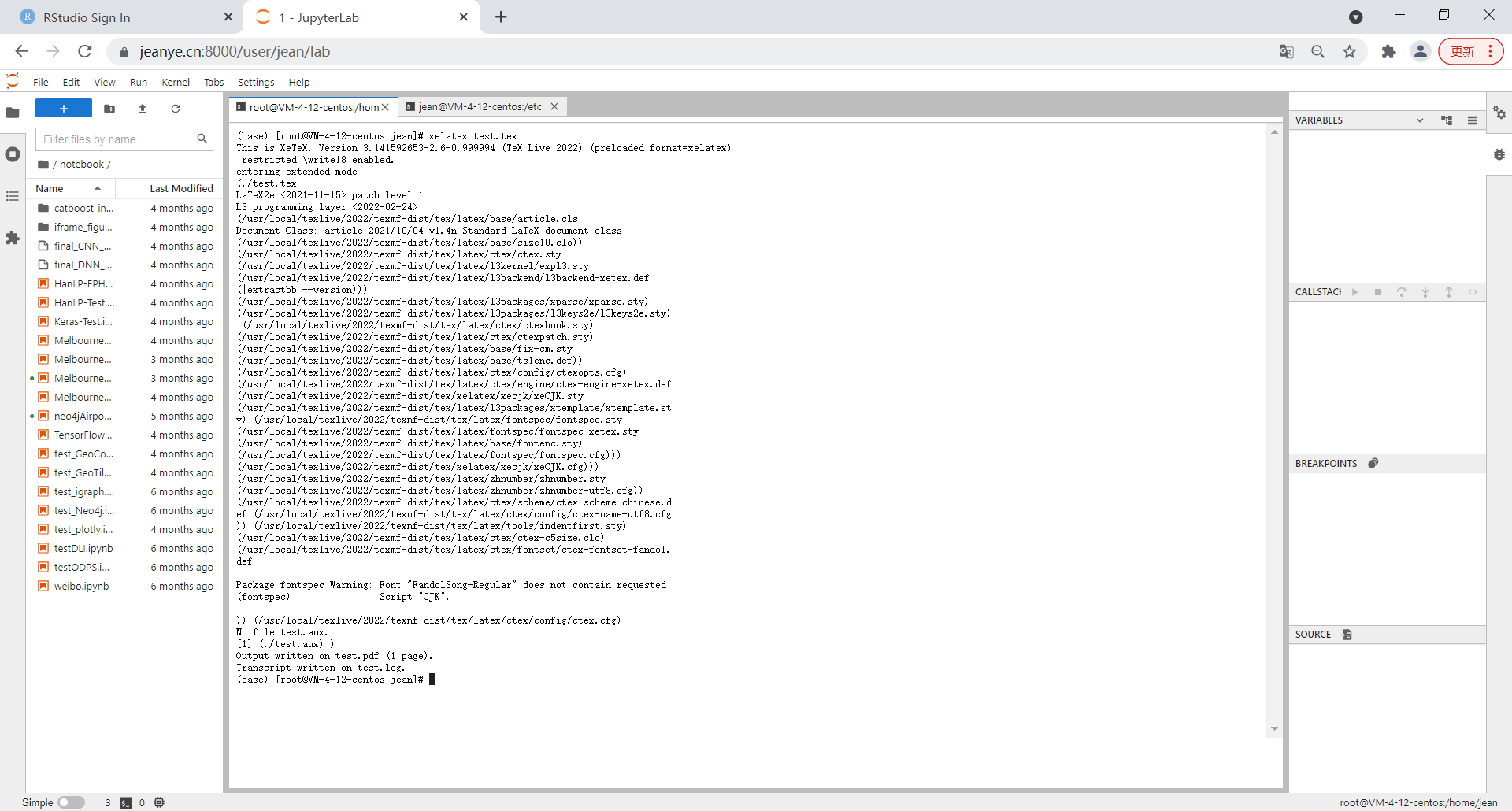
# source /etc/profile

卸载镜像：

# umount /mnt/

2）测试安装

A、测试texlive安装。测试用例test.tex同Windows平台，命令行输出要详细一点，生成的PDF是一样的。



B、在Rstudio中测试rmarkdown，注意/etc/profile中设置的texlive环境变量不会带入Rstudio Linux Server的 R session中，需要在Renviron配置文件中加进去，主要是PATH变量，具体可以参阅[文档1](https://github.com/rstudio/rstudio/issues/1878)和[文档2](https://www.anycodings.com/1questions/2281129/rstudio-shows-a-different-path-variable)。

# which R

/usr/lib64/R-4.1.2/bin/R

# cd /usr/lib64/R-4.1.2/lib64/R/etc

# ls

javaconf ldpaths Makeconf Renviron repositories

# vi Renviron

# Added by Jean for texlive 2022/09/15

PATH=${PATH}:/usr/local/texlive/2022/bin/x86\_64-linux

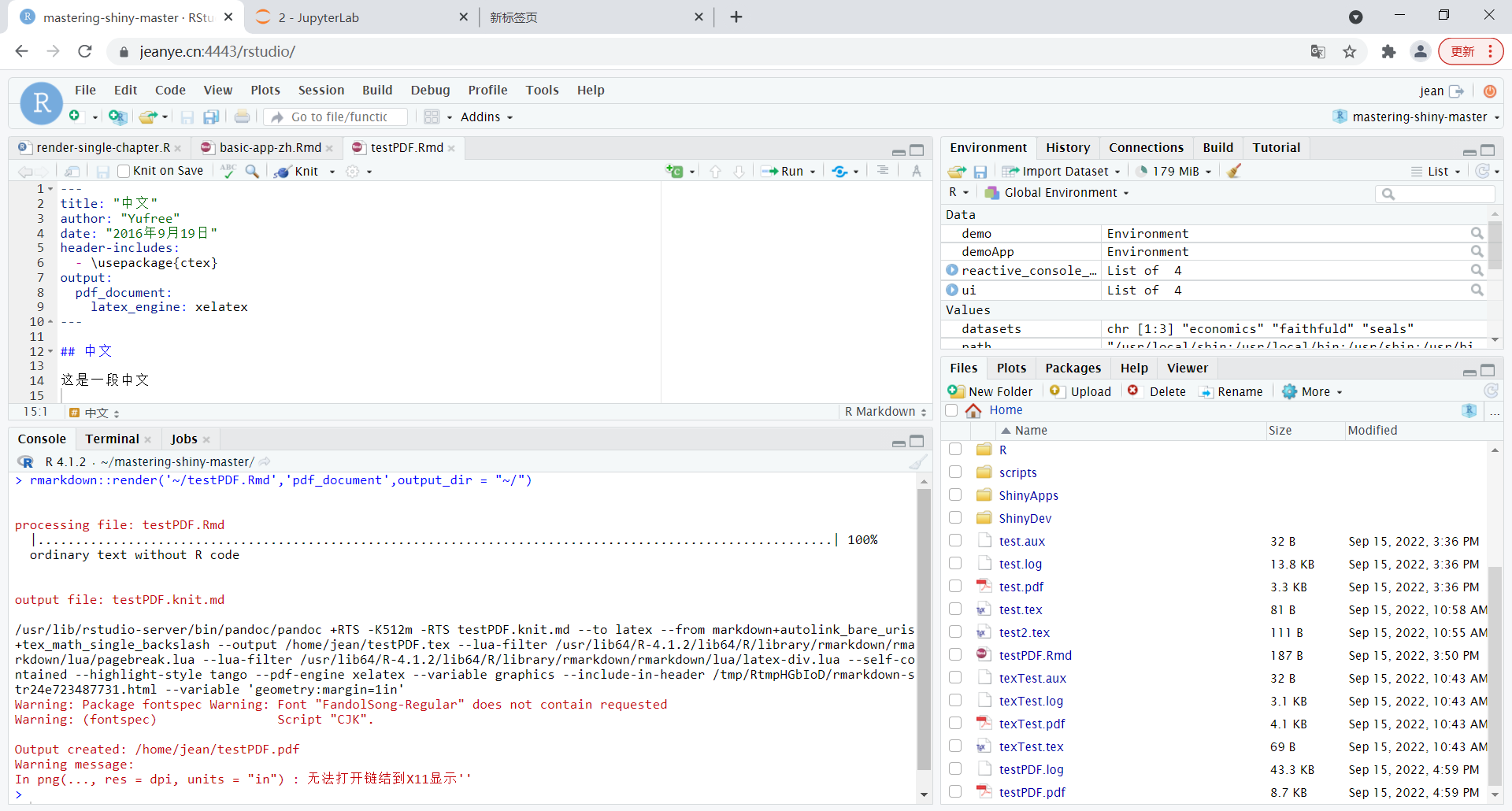
然后重启Rstudio Linux Server以使路径生效：

# cd /usr/sbin

# ./rstudio-server stop&

# ./rstudio-server start&

测试用例同Windows平台，渲染输出结果也相同。



第二节 Shiny 交互式动态深度分析报告

R markdown的交互式动态报告由Shiny支持，在Rstudio中通过菜单File->New File->R Markdown->Shiny即可创建，与其它R markdown文档的区别是它提供了反应式输入输出组件，根据输入实时更新文档中相应的输出。先看一个简单的例子。

1、内嵌反应式组件的Shiny文档

---

title: "Shiny Doc 测试"

output: html\_document

runtime: shiny

---

## R markdown Shiny文档测试

### 运行一些R代码

```{r}

date<-Sys.Date()

```

日期，inline形式的代码只显示结果，不会显示代码：`r date`

### 定义反应式输入，相当于ui.R

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

```{r}

numericInput("rows", "头部显示行数", 5)

```

### 定义反应式输出，相当于server.R，注意引用都要用render()函数。

```{r}

rows<-reactive({input$rows})

renderTable({ head(cars, rows())})

```

### 在后面可以继续引用反应式输入，包括反应式变量，render()输出等反应式组件。

行数：`r renderText({rows()})`。

### 定义另一个反应式输入

```{r}

sliderInput("tail","尾部显示行数：", min = 2, max = 6, value = 4)

```

### 在后面引用反应式输出

```{r}

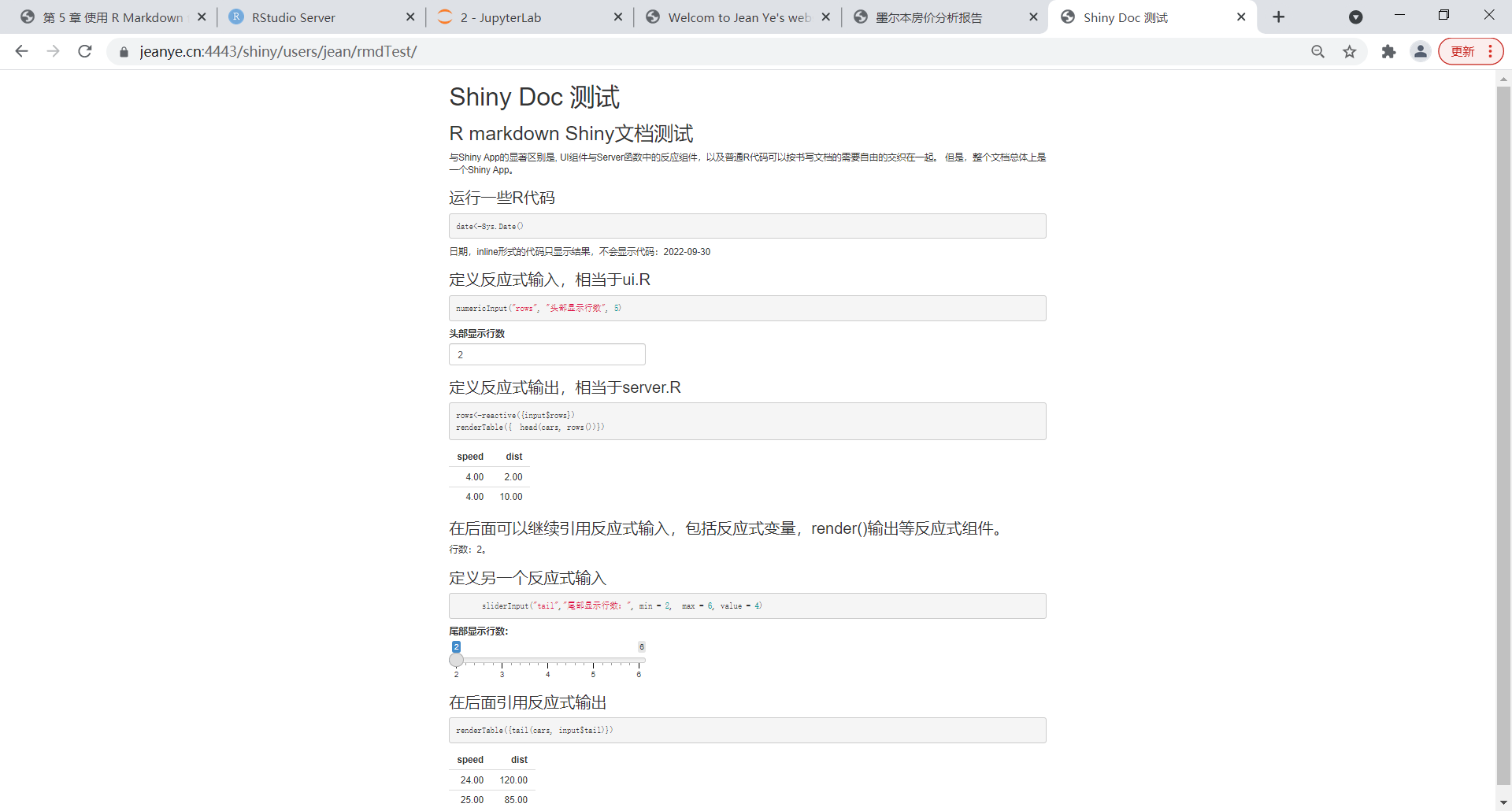
renderTable({tail(cars, input$tail)})

```

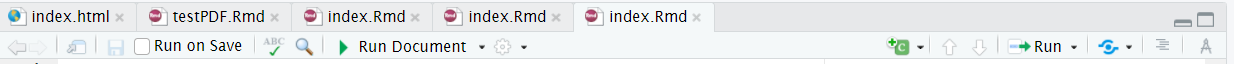
在头部的YAML定义中，通过runtime: shiny来定义它是个Shiny文档，它的输出只能是html\_document,在线交互式，不能是pdf\_document等其它格式。

然后在文档中直接嵌入Shiny反应式组件。与Shiny App的显著区别是, ui与server函数中的反应式组件，以及普通R代码可以按书写文档的需要自由的交织在一起，Shiny Server上部署运行时会自动分开抽取出来。

渲染效果如下图，改变行数输入值，输出就会相应调整。



这时Rstudio文档编辑工具栏上的运行按钮会自动由Knit变为Run Document，Rstuido识别它是个Shiny文档。它适用于从头开始创建报告。



2、内嵌独立Shiny App的文档

有3种嵌入的方式，虽然与Shiny文档的其它部分显示在同一个网页中，但它们操作不了嵌入Shiny APP内部的数据，没有这样的API。

1. shinyApp()函数嵌入，嵌入在文档中定义的Shiny APP。

---

title: "Shiny Doc"

output: html\_document

runtime: shiny

---

```{r, echo=FALSE}

shinyApp(

ui = fluidPage(

selectInput("region", "Region:",

choices = colnames(WorldPhones)),

plotOutput("phonePlot", height=270)

),

server = function(input, output) {

output$phonePlot <- renderPlot({

barplot(WorldPhones[,input$region]\*1000,

ylab = "Number of Telephones", xlab = "Year")

})

},

options = list(height = 345)

)

```

1. shinyAppDir()嵌入，嵌入部署运行在其它目录的Shiny APP。

---

title: "Shiny Doc"

output: html\_document

runtime: shiny

---

```{r, echo=FALSE}

# 如果有global.R，要在嵌入之前手工调用，它不会自动执行。

# https://github.com/rstudio/rmarkdown/issues/211

# source("../kmeansExample/global.R")

shinyAppDir(

"../kmeansExample",

options = list(

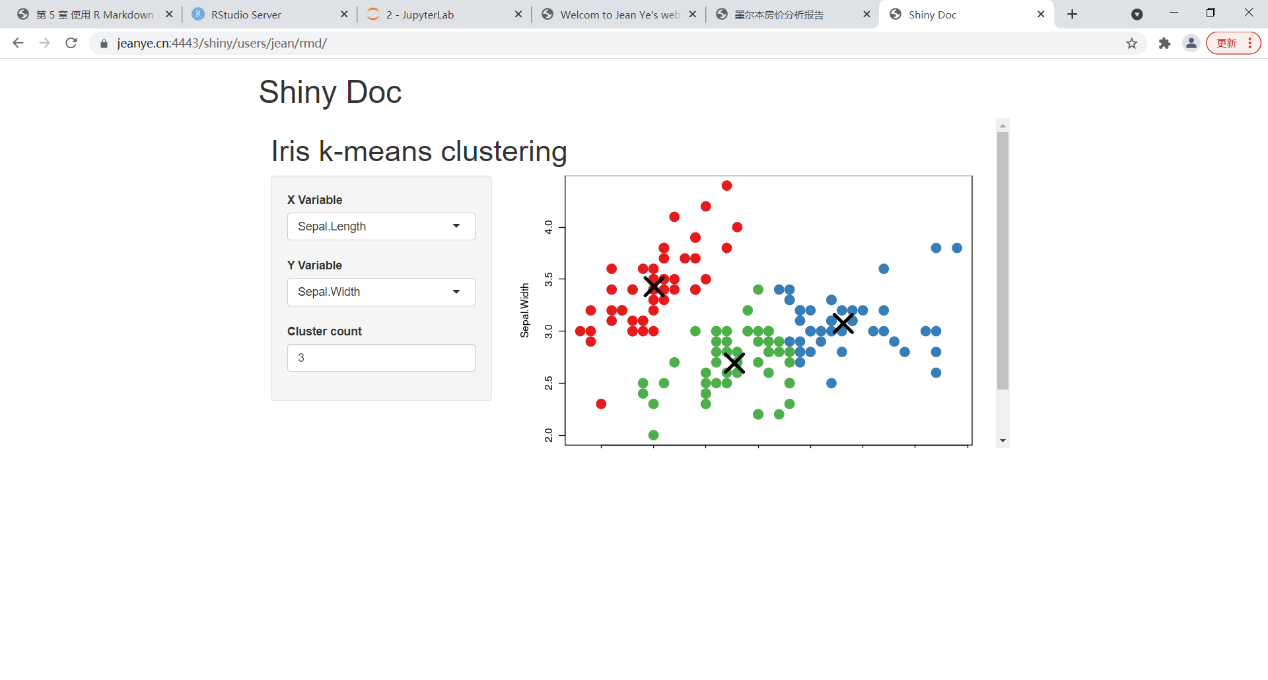
width = "100%"

)

)

```

这两种方式嵌入独立的Shiny App，都不支持global.R全局设置脚本，需要在文档中嵌入Shiny App之前自己手工调用执行。上面的Rmd执行效果如下：



1. knitr::include\_app()嵌入，网页IFRAME嵌入。

---

title: "墨尔本房价分析报告"

author: "Jean"

date: "`r Sys.Date()`"

output: html\_document

runtime: shiny

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

## 运行一些R代码，以便在报告里引用它们的结果。

显示当前的工作目录，显示代码及其执行结果是为了说明这是一个R markdown报告。

```{r somecode, echo=TRUE}

getwd()

```

## 嵌入完整Shiny App

通过在R Markdown 报告中使用 `shinyAppDir` 函数嵌入其它目录下运行的完整Shiny App，本例嵌入了Shiny App “墨尔本房价回归分析”，它调用底层的Python脚本在预处理数据的基础上完成各个回归模型的计算(当时是用Python研究的GBDT回归模型，当然也可以完全用R来完成)，Shiny App主要是管理各个回归模型数据的交互式可视化展示：

```{r shinyapp, eval=FALSE, echo=TRUE, out.width="100%"}

# R markdown嵌入Shiny App时，目前不支持加载global.R, 具体见下面的issue

# https://github.com/rstudio/rmarkdown/issues/211

# 对于使用global.R的Shiny App，要自己手动加载global.R

# 不过在Rstudio开发环境中虽然测试通过了，但在Shiny Server部署运行时有问题。

# global.R应该是正常加载了，但UI中不能显示所有render()函数渲染的组件，目前还没有找到原因。

source("../melbourne/global.R")

shinyAppDir(

"../melbourne",

options = list(

width = "100%", height=1000

))

#

# 这一行输出正常，证明在Shiny Server中手动加载global.R执行正常。

# perf

```

改用include\_app来嵌入App，因为它是通过IFRAME嵌入的，没有运行环境的影响。

```{r, eval=TRUE, echo=TRUE, out.width="100%"}

# knitr::include\_app()在Rstudio开发环境与Shiny Server部署环境运行都正常。

knitr::include\_app("https://jeanye.cn:4443/shiny/users/jean/melbourne/", height="1200px")

```



这是通过在当前的Shiny文档网页中插入一个IFRAME来嵌入，完全不受Shiny文档运行环境的影响。比如墨尔本房价分析App，如果用前一种方式嵌入，由于未知的原因，所有render()函数渲染输出的Widget都显示为空白，而用这种方式嵌入就可以正常显示。

3、通过shinytest包后台运行Shiny App嵌入

一般引用已经部署应用的Shiny App来撰写深度分析报告，往往需要比较不同输入参数下的分析结果，这就要求Shiny文档与引用的Shiny App之间有交互的能力，由Shiny文档中设定引用Shiny App的输入参数，引用它的输出结果，然后在文档中加以比较分析。幸运的是，可以通过[shinytest包](https://github.com/rstudio/shinytest)（以及它的升级版[shinytest2包](https://github.com/rstudio/shinytest2)）后台运行被引用的Shiny App来完成，虽然它没有可视的UI界面，但可以在Shiny文档中通过程序去模拟UI的操作，然后提取输出的结果展示或比较分析。下面以墨尔本房价回归分析APP为例具体看看。

---

title: "墨尔本房价分析报告"

author: "Jean"

date: "`r Sys.Date()`"

output: html\_document

runtime: shiny

---

## 后台加载墨尔本房价回归分析Shiny APP

需要用shinytest包，以便设置反应式输入，以及截图和提取反应式输出。

```{r}

library(shiny)

library(shinytest)

# Set loadTimeout to a long time

# Can not use shinytest for larger application

# Error in sd\_startShiny(self, private, path, seed) :

# Cannot find shiny port number. Error:

# https://community.rstudio.com/t/can-not-use-shinytest-for-larger-application/6514

app <- shinytest::ShinyDriver$new("../melbourne", loadTimeout=1000000)

# get background browser window size.

app$getWindowSize()

```

### 默认算法与异常值阀值：

算法：`r app$getValue("algo")` 异常值阀值：`r app$getValue("threshold")`% 屏幕截图。

```{r, out.width = "100%", fig.width=10, fig.height=10}

app$takeScreenshot()

```

### 设置新的算法与阀值参数

设置算法

```{r}

app$setInputs(algo="LigthGBM")

app$getValue("algo")

```

设置异常值阀值

```{r}

app$setInputs(threshold=35)

app$getValue("threshold")

```

算法：`r app$getValue("algo")` 异常值阀值：`r app$getValue("threshold")`% 屏幕截图。

```{r, out.width = "100%", fig.width=10, fig.height=10}

app$takeScreenshot()

```

## 直接显示其中一个Widget

### 提取Shiny APP中返回的数据

用shinytest包的findWidget()函数找到Widget，然后用Widget的getHtml()函数得到Widget的HTML源码（包括数据），

然后把HTML源码插入到当前页面中。因为shinytest后台运行的Shiny APP运行在另一个R进程中，Widget的源码和数据不在本进程中，所以要得到Widget的HTML源码，并插入到当前的页面。反应式输入的值可以通过getValue()函数直接得到，所以需要这样提取的一般是反应式输出的值，特别是render()函数渲染输出的HTML Widget。

参阅：

https://cran.r-project.org/web/packages/shinytest/shinytest.pdf

https://rstudio.github.io/shinytest/reference/Widget.html

显示所有Widget

```{r}

app$listWidgets()

```

### 性能数据

```{r}

perf<-app$findWidget("performance")

perfHtml<-perf$getHtml()

HTML(perfHtml)

```

### 训练集拟合效果

```{r}

trainPlot<-app$findWidget("trainPlot")

trainHtml<-trainPlot$getHtml()

HTML(trainHtml)

```

### 验证集拟合效果

```{r}

validPlot<-app$findWidget("validPlot")

validHtml<-validPlot$getHtml()

HTML(validHtml)

```

### 异常数据

DataTable Widget只返回第一页，因为getHtml()只返回了一页。

```{r}

outliers<-app$findWidget("outliers")

outliersHtml<-outliers$getHtml()

HTML(outliersHtml)

```

可以通过shinytest包的executeScript()函数执行JavaScript找到“下页”按钮，模拟点击翻页，用程序逐页把数据抓过来。

```{r}

app$executeScript("

var nextPage = document.getElementsByClassName('next');

if (nextPage !=null){nextPage[0].click();}"

)

```

```{r}

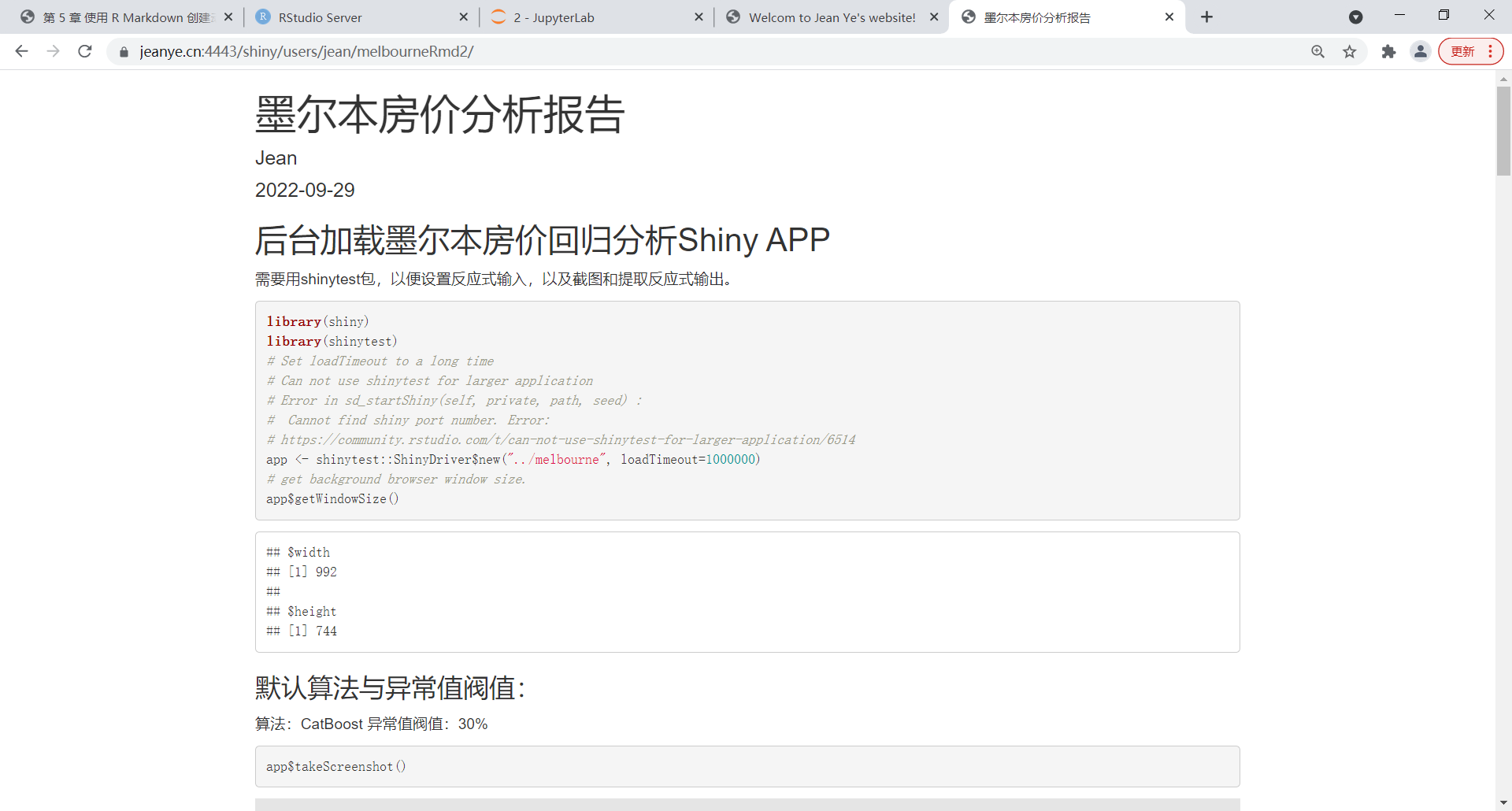
outliers<-app$findWidget("outliers")

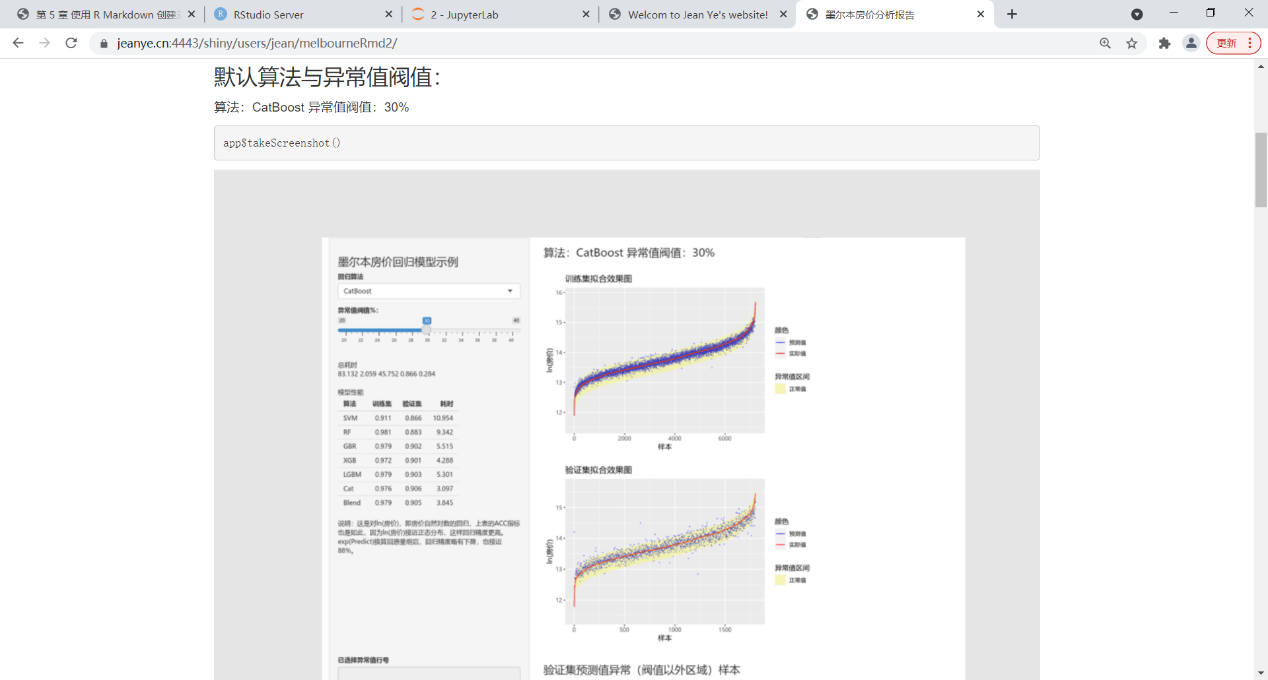
outliersHtml<-outliers$getHtml()

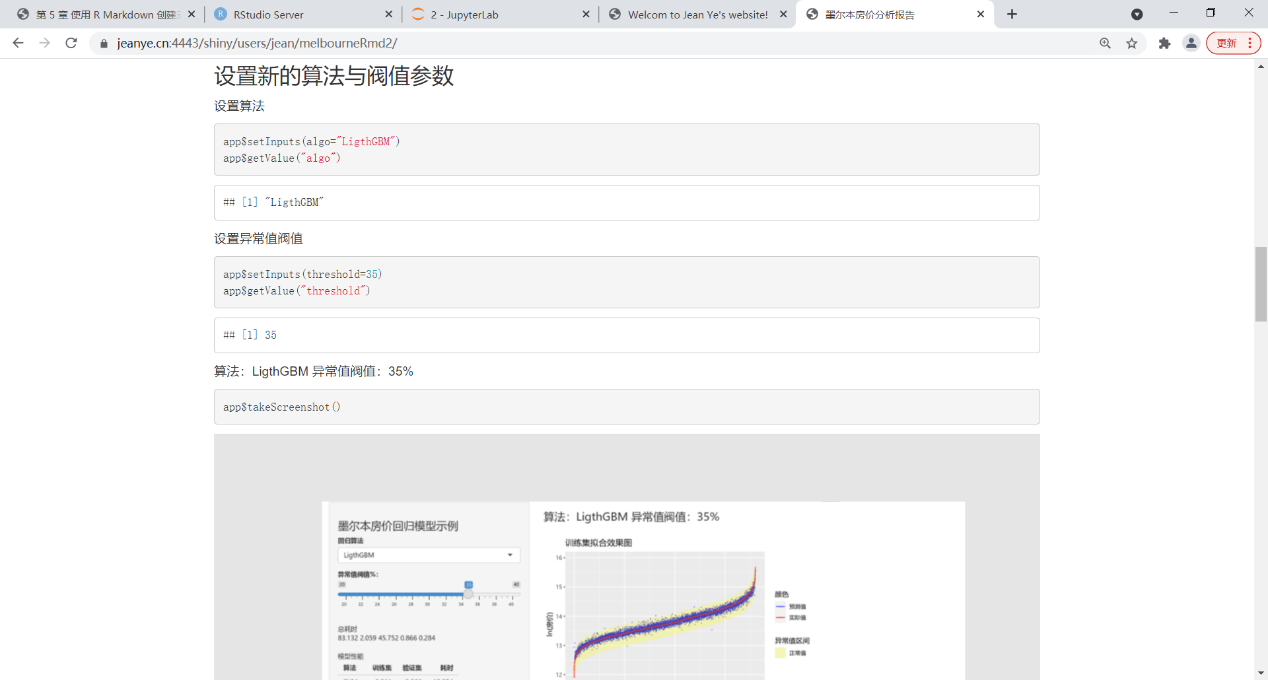
HTML(outliersHtml)

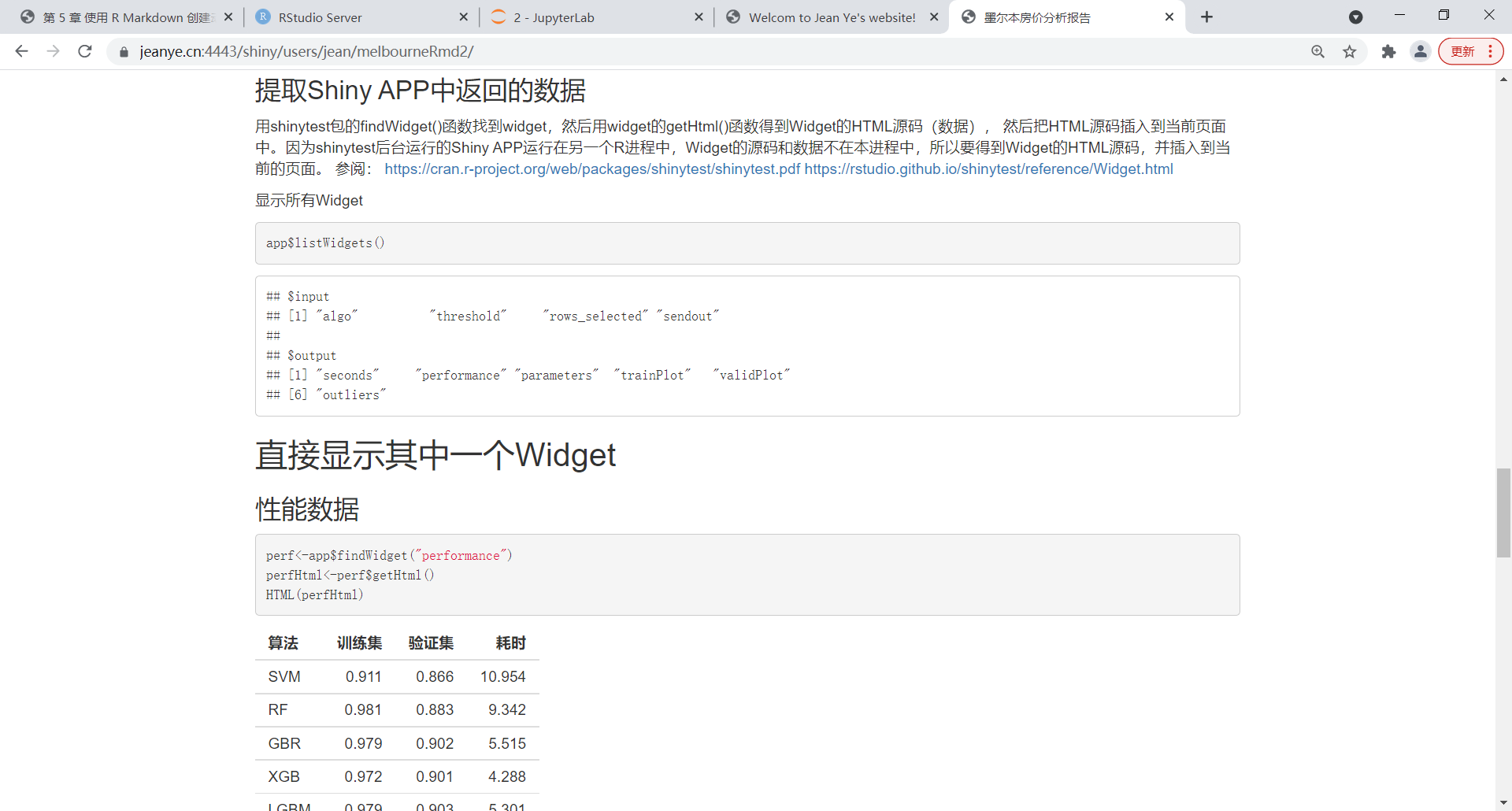
```

执行结果如下，这样写深度分析报告就比较充分和自由了。

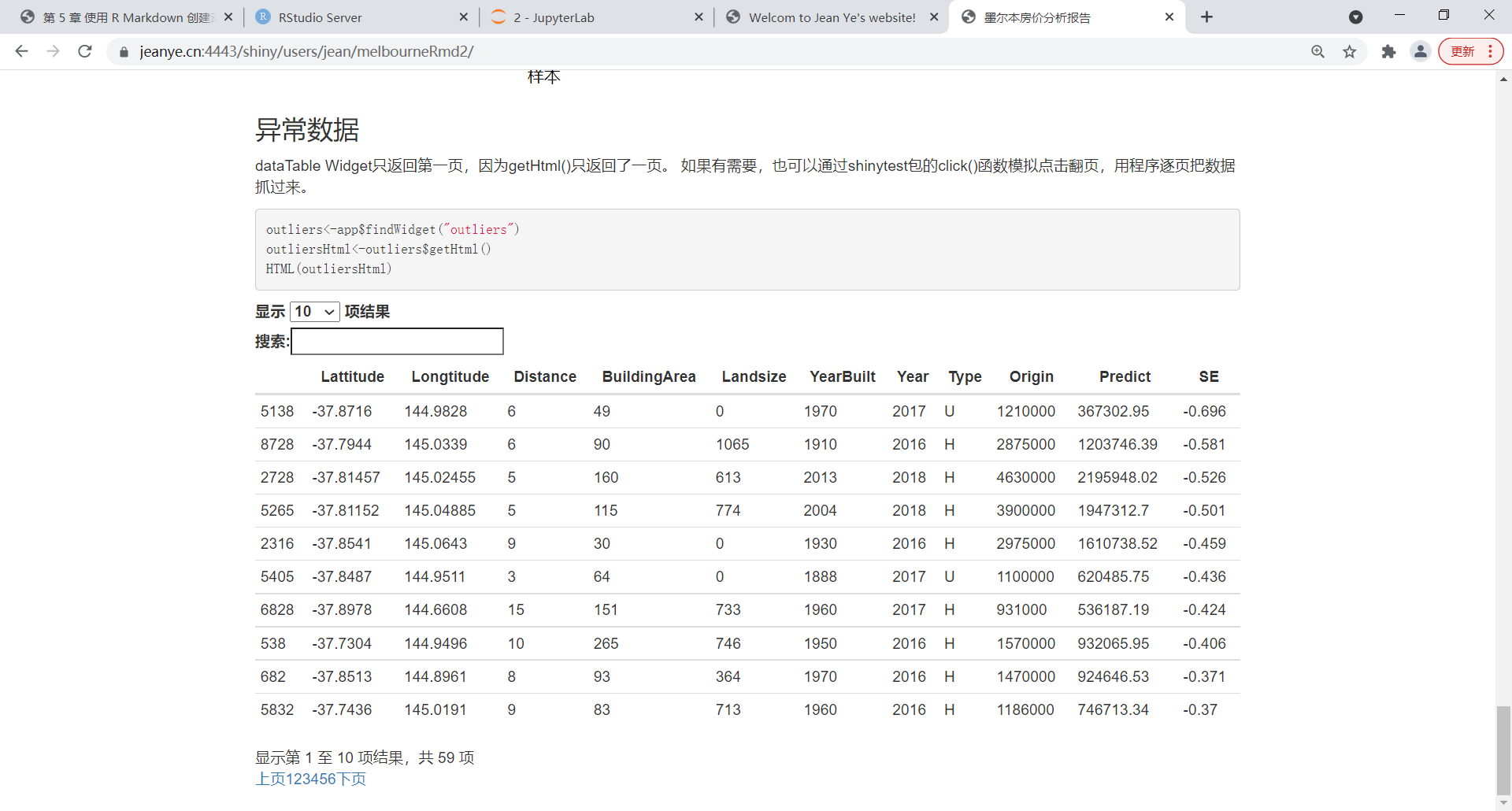


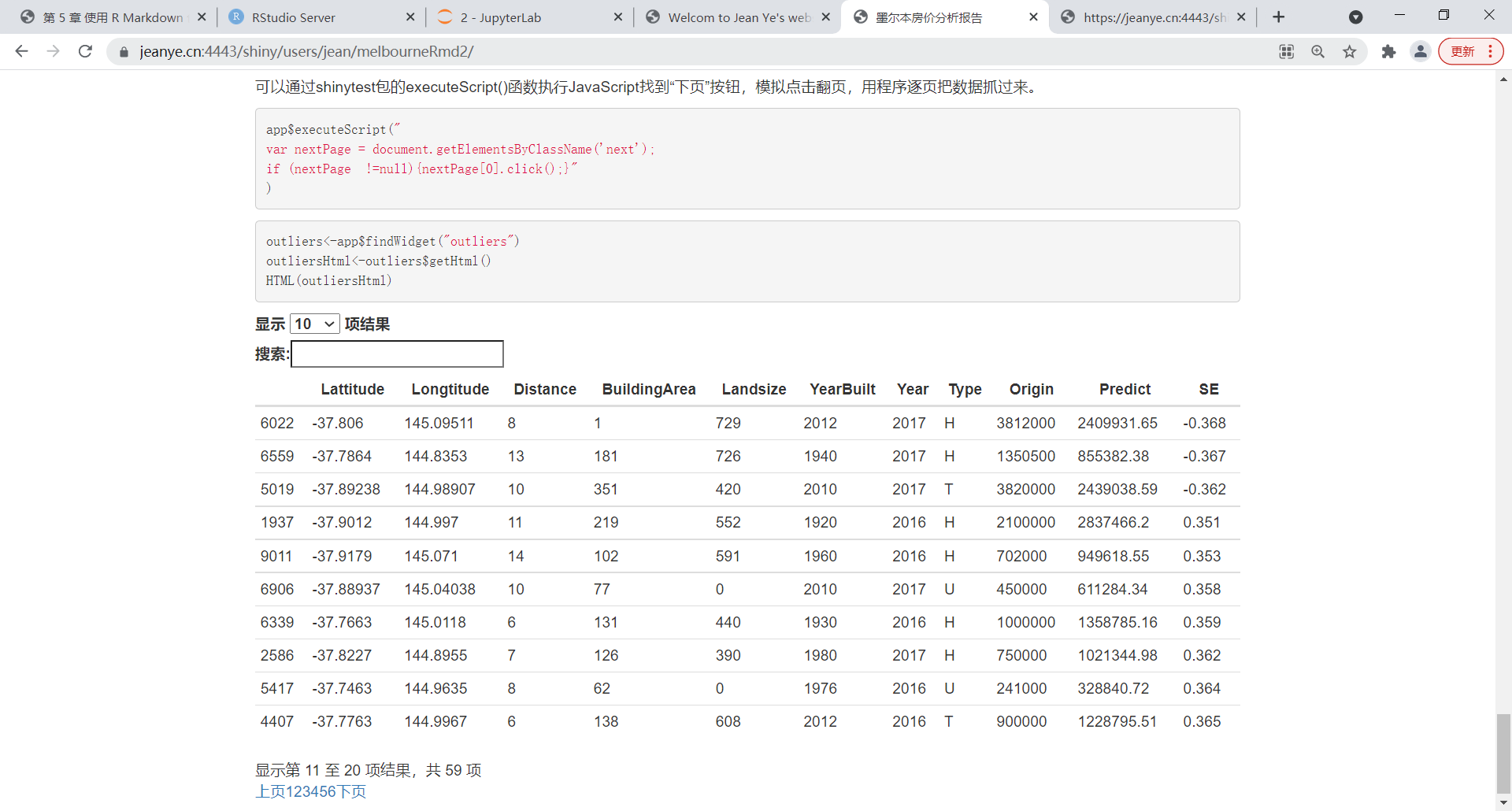










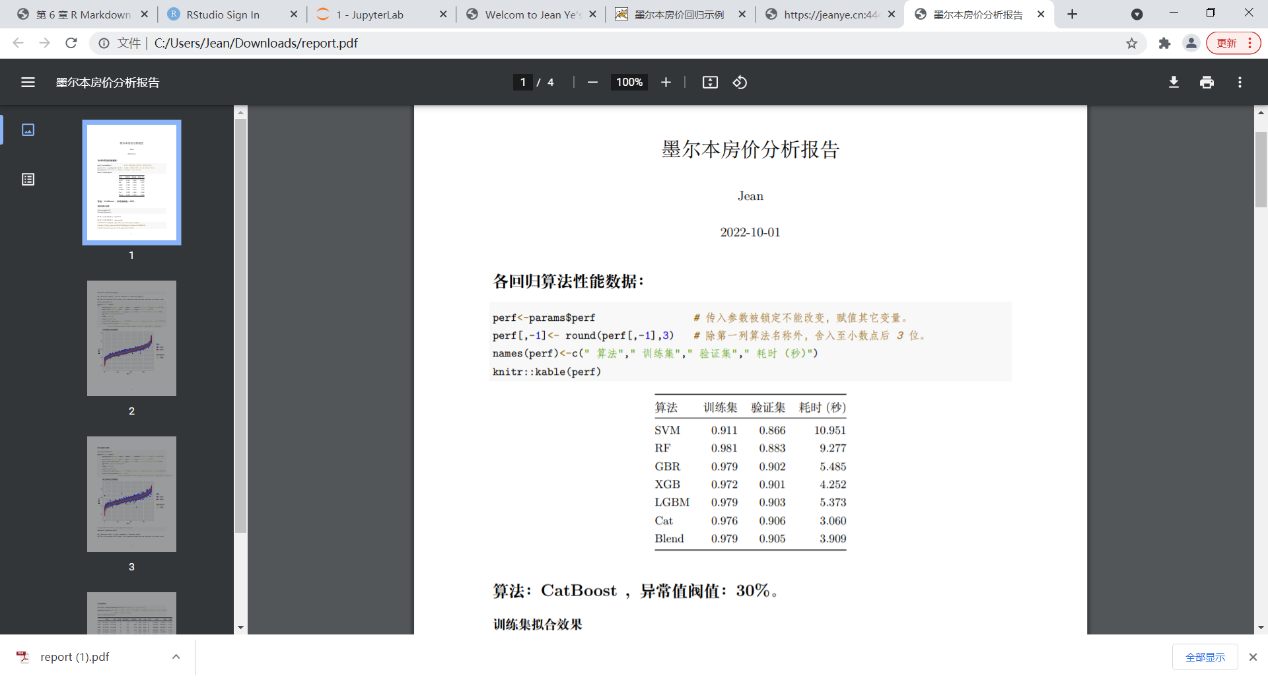


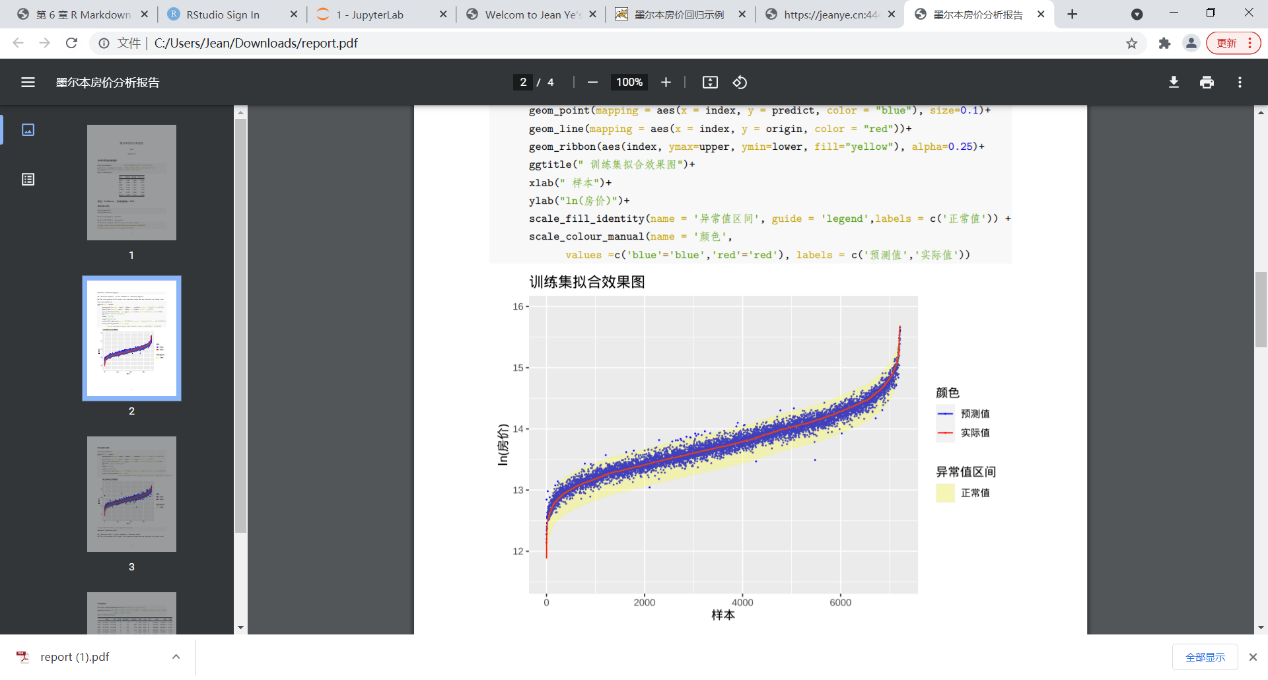
第三节 Shiny生成可下载的动态深度分析报告

上一节用R markdown生成Shiny交互式深度分析报告，它是在线的，只能是HTML格式，对于内网部署的APP，传阅分享就有所不便，有时候人们也需要有可打印的版本。本节在Shiny APP中增加根据APP当前的输入和输出生成PDF分析报告的功能，可以方便的传阅或打印，两种途径合在一起就比较完整了。

还是以墨尔本市房价分析APP为例，先看看生成的PDF分析报告，整合了APP的反应式输入参数和输出，以及R markdown程序脚本。

1、渲染结果PDF





2、对Shiny App要做一点改造

1）global.R。加载showtext包支持ggplot2绘图中文标注。

library(reticulate)

library(ggplot2)

library(DT)

library(showtext)

# R Plot添加中文及其他字体[showtext]

# https://blog.csdn.net/weixin\_46128755/article/details/125825935

showtext\_auto()

# 确定各回归模型Python脚本的目录位置

print(getwd())

path<- "../../scripts/Melbourne\_Regress.py"

print(path)

# 装入脚本，用经过预处理的数据集，在装入时一次性加载所有回归模型并完成训练集与验证集的拟合。

# 模型超参数调优另外用Python脚本完成。

# 这些脚本都可以用R语言完成，不过原来已经用Python写好了，通过reticulate包调用也可以，就不改了。

t3<- proc.time()

source\_python(path)

t4<- proc.time()

# 总耗时、系统耗时、用户耗时，...，...

# An object of class "proc\_time" which is a numeric vector of length 5,

# containing the user, system, and total elapsed times for the currently running R process,

# and the cumulative sum of user and system times of any child processes

# spawned by it on which it has waited.

seconds<-t4-t3

# 各模型训练与验证的结果，封装为data frame

train<- PredictTrain()

valid<- PredictValid()

# 各模型的性能数据

perf<- performance()

# 回归算法列表

algos<-names(train)

# 这个是显示在下拉列表中供选择的算法名称，作为命名列表的名字，它的内容是dataframe的列名。

# 从第2项开始。

names(algos)<-c("Origin","SVM","RandomForest","GBR","XGB","LigthGBM","CatBoost","Blend")

2）ui.R增加了下载按钮，imageOutput()居中显示图片。

# 浏览器端UI函数。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(

tags$script(HTML("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

Shiny.setInputValue('algo', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('rows\_selected');

if (selected ==null || selected.value.length==0){

alert('没有选中的行!');

}else{

if (parent == null){

alert(selected.value);

} else {

try{

parent.postMessage(selected.value, '\*');

} catch(error){

alert(error);

}

}

}

});

"))

),

sidebarLayout(

# Sidebar with a selectioninput and a numericInput

sidebarPanel(

h3("墨尔本房价回归模型示例"),

selectInput(

'algo',

'回归算法',

names(algos)[2:8],

selected = 'CatBoost'

),

sliderInput("threshold",

"异常值阀值%：",

min = 20, max = 40, value = 30),

HTML("<br>总耗时<br>"),

textOutput("seconds"),

HTML("<br>模型性能<br>"),

tableOutput("performance"),

HTML("说明：这是对ln(房价)，即房价自然对数的回归，上表的ACC指标也是如此，因为ln(房价)接近正态分布，这样回归精度更高。<br>

exp(Predict)换算回原量纲后，回归精度略有下降，也接近88%。"),

HTML("<br><br><br><br><br><br><br><br><br><br><br><br>"),

textAreaInput("rows\_selected","已选择异常值行号", rows =6, resize="vertical", value =""),

# 插入javascript，禁止自己修改 rows\_selected textAreaInput

tags$script(HTML("

var rows\_selected = document.getElementById('rows\_selected');

rows\_selected.disabled = true;

")),

HTML("<br>"),

actionButton("sendout", "传出选中的行", class = "btn-success"),

HTML("<br>"),HTML("<br>"),

downloadButton("report","下载分析报告")

),

# Show the network

mainPanel(

# 画拟合效果

h3(textOutput("parameters")),

HTML("<br>"),

imageOutput("trainPlot"),

HTML("<br>"),

HTML("<div style='text-align: center;'>"),

imageOutput("validPlot"),

HTML("</div>"),

HTML("<br><h3>验证集预测值异常（阀值以外区域）样本</h3><br>"),

dataTableOutput("outliers")

)

)

)

3）server.R增加了处理下载的函数output$report <- downloadHandler()。提取数据的那段代码有点重复，可以抽取为函数，这里为明确新增的代码，就不修改原来的代码，不抽取出来。这些数据作为参数在调用时传给Rmd脚本。同时展示了renderPlot()与renderImage()两种ggplot2作图的方法。

# 输出图形到SVG临时文件以支持中文标注

# https://stackoverflow.com/questions/39093777/renderimage-and-svg-in-shiny-app/39263320#39263320

# 多图层图例设置参阅

# https://stackoverflow.com/questions/18394391/custom-legend-for-multiple-layer-ggplot

# Shiny data table国际化语言支持参阅

# https://rstudio.github.io/DT/004-i18n.html

# Shiny Data Table的操作可以参考

# https://rstudio.github.io/DT/shiny.html

# 选中行的操作可以参考

# https://yihui.shinyapps.io/DT-rows/

# ggplot2中文支持可以参考

# https://blog.csdn.net/weixin\_46128755/article/details/125825935

# https://statisticsglobe.com/change-font-size-of-ggplot2-plot-in-r-axis-text-main-title-legend

shinyServer(function(input, output, session) {

algo<- reactive({

input$algo

})

threshold<- reactive({

input$threshold

})

# 更新由宿主页面传入的参数

observe({

print(algo())

# print(threshold())

# 更新由宿主页面传入的参数

updateTextInput(session, "algo", value = algo())

# 注意这里用updateTextInput更新文本到浏览器端，如果用updateSelectInput()会报错，可能要求的是下标。

# updateSelectInput(session, "algo", value = algo())

})

output$parameters<-renderText({

paste("算法：",algo()," 异常值阀值：",threshold(),"%",sep="")

})

output$seconds<-renderText({

seconds

})

output$performance<- renderTable({

names(perf)<-c("算法","训练集","验证集","耗时")

perf

},

digits = 3

)

# 第一种ggplot2作图支持中文的方法

output$trainPlot<-renderPlot({

colName<- algos[algo()]

data<- data.frame(origin = train["origin"], predict = train[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

p<- ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("训练集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))+

theme(plot.title = element\_text(size = 24), legend.title=element\_text(size=18),

legend.text=element\_text(size=14), axis.title = element\_text(size = 18),

axis.text.x = element\_text(size = 14), axis.text.y = element\_text(size = 14))

gridExtra::grid.arrange(egg::set\_panel\_size(p=p, width=unit(6, "in"), height=unit(4, "in")))

})

# 第二种ggplot2作图支持中文的方法

output$validPlot<-renderImage({

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

image<-ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("验证集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))

# 输出SVG到临时文件

# This file will be removed later by renderImage

outfile <- tempfile(fileext='.svg')

#This actually save the plot in a image

#ggsave(file=outfile, plot=image, width=mysvgwidth, height=mysvgheight)

ggsave(file=outfile, plot=image, width=6, height=4)

# Return a list containing the filename

list(src = normalizePath(outfile),

contentType = 'image/svg+xml',

#width = width,

height = 400,

alt = "验证集拟合效果图")

},

# SVG图发到客户端后，renderImage删除临时文件

deleteFile = TRUE

)

output$outliers<- renderDataTable({

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data["outlier"]<- FALSE

data[which(data["predict"] < data["origin"]+log(1-threshold()/100)),"outlier"] <-TRUE

data[which(data["predict"] > data["origin"]+log(1+threshold()/100)),"outlier"] <-TRUE

valid\_X3<- valid\_X2[,c("Lattitude","Longtitude","Distance","BuildingArea","Landsize","YearBuilt","Year")]

valid\_X3["Type"]<-valid\_X2["Type\_h"]\*4+valid\_X2["Type\_t"]\*2+valid\_X2["Type\_u"]

valid\_X3[which(valid\_X3["Type"]==4),"Type"]<-"H"

valid\_X3[which(valid\_X3["Type"]==2),"Type"]<-"T"

valid\_X3[which(valid\_X3["Type"]==1),"Type"]<-"U"

#valid\_X3["Origin"]<-data["origin"]

#valid\_X3["Predict"]<-data["predict"]

# 转换为原量纲

valid\_X3["Origin"]<-round(exp(data["origin"]),2)

valid\_X3["Predict"]<-round(exp(data["predict"]),2)

outliers<-valid\_X3[which(data["outlier"]==TRUE),]

outliers["SE"]<-round((outliers["Predict"] - outliers["Origin"])/outliers["Origin"],3)

#outliers["ID"]<- row.names(outliers)

outliers<-outliers[order(outliers["SE"]),]

# 放入 session 中以便后面选择行时引用，不能用行索引，它不是全局的， 行名才是全局的。

session$userData$outliers<- outliers

},

#rownames = FALSE,

options = list(

pageLength = 10,

#language = list(url = '//cdn.datatables.net/plug-ins/1.10.11/i18n/Chinese.json')

language = list(url = 'Chinese.json')

)

)

# 返回并显示选中行的行名，行名是全局的，用于定位对应的行，更新浏览器端显示。

observe({

outliers<- session$userData$outliers

rows<- outliers[input$outliers\_rows\_selected,]

outliersStr<- paste(row.names(rows), collapse=",")

updateTextAreaInput(session, "rows\_selected", value = outliersStr)

})

# 下载分析报告，output$id与downloadButton(id)对应。

output$report <- downloadHandler(

filename = "report.pdf",

content = function(file) {

# 训练集拟合数据

colName<- algos[algo()]

data<- data.frame(origin = train["origin"], predict = train[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

trainSelected<-data

# 验证集拟合数据

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

validSelected<-data

# 异常值数据

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data["outlier"]<- FALSE

data[which(data["predict"] < data["origin"]+log(1-threshold()/100)),"outlier"] <-TRUE

data[which(data["predict"] > data["origin"]+log(1+threshold()/100)),"outlier"] <-TRUE

valid\_X3<- valid\_X2[,c("Lattitude","Longtitude","Distance","BuildingArea","Landsize","YearBuilt","Year")]

valid\_X3["Type"]<-valid\_X2["Type\_h"]\*4+valid\_X2["Type\_t"]\*2+valid\_X2["Type\_u"]

valid\_X3[which(valid\_X3["Type"]==4),"Type"]<-"H"

valid\_X3[which(valid\_X3["Type"]==2),"Type"]<-"T"

valid\_X3[which(valid\_X3["Type"]==1),"Type"]<-"U"

#valid\_X3["Origin"]<-data["origin"]

#valid\_X3["Predict"]<-data["predict"]

# 转换为原量纲

valid\_X3["Origin"]<-round(exp(data["origin"]),2)

valid\_X3["Predict"]<-round(exp(data["predict"]),2)

outliers<-valid\_X3[which(data["outlier"]==TRUE),]

outliers["SE"]<-round((outliers["Predict"] - outliers["Origin"])/outliers["Origin"],3)

#outliers["ID"]<- row.names(outliers)

outliers<-outliers[order(outliers["SE"]),]

# 生成参数列表，传入报告所需的参数。

params <- list(

algo = input$algo,

threshold = input$threshold,

rows = input$rows\_selected,

perf = perf,

train = trainSelected,

valid = validSelected,

outliers = outliers

)

id <- showNotification(

"正在生成报告...",

duration = NULL,

closeButton = FALSE

)

on.exit(removeNotification(id), add = TRUE)

rmarkdown::render("report.Rmd",

output\_file = file,

params = params,

envir = new.env(parent = globalenv())

)

}

)

})

3、R markdown脚本report.Rmd，定义参数接收Shiny APP传入的数据，放在Shiny APP相同的目录下，由上面的rmarkdown::render()调用。

---

title: "墨尔本房价分析报告"

author: "Jean"

date: "`r Sys.Date()`"

header-includes:

- \usepackage{ctex}

output:

pdf\_document:

latex\_engine: xelatex

params:

algo: NA

threshold: NA

rows: NA

perf: NA

train: NA

valid: NA

outliers: NA

---

# 各回归算法性能数据：

```{r}

perf<-params$perf # 传入参数被锁定不能改变，赋值其它变量。

perf[,-1]<- round(perf[,-1],3) # 除第一列算法名称外，舍入至小数点后3位。

names(perf)<-c("算法","训练集","验证集","耗时(秒)")

knitr::kable(perf)

```

# 算法：`r params$algo` ，异常值阀值：`r params$threshold`%。

## 训练集拟合效果

```{r fig.showtext = TRUE}

library(ggplot2)

library(showtext)

# Rmarkdown安装配置、输出PDF正文中文以及图片中文配置

# https://blog.csdn.net/weixin\_46128755/article/details/125825935

# 设置showtext\_auto()，打开ggplot2中文支持。

showtext\_auto()

data<-params$train

ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("训练集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))

```

## 验证集拟合效果

```{r fig.showtext = TRUE}

data<-params$valid

ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("验证集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))

```

## 异常值列表

```{r}

outliers<-head(params$outliers,10) # 演示性质，只打印前10行。

names(outliers)<-c("纬度","经度","距离","建筑面积","占地面积","建成","交易","类型",

"价格","预测","偏差")

knitr::kable(outliers)

```

## 选中的异常值行号：`r params$rows`。

ggplot2作图标注中文用showtext包处理,参阅[资料](https://blog.csdn.net/u012111465/article/details/79945372)。在R代码块的开头选项中设置{r fig.showtext = TRUE}，表示用showtext包来处理ggplot2的标注文本，然后在加载showtext包后执行showtext\_auto（），后面ggplot2的绘图都会由showtext包来处理。

第四节 在 R markdown报告中混合使用各种语言

R markdown目前支持数十种编程语言，所以是个适用范围很广的强大专业编著工具，下面的命令可以列出支持的语言。

> names(knitr::knit\_engines$get())



这里面还没有java，不过 [scala](https://baike.baidu.com/item/Scala/2462287?fr=aladdin)是由java派生的，也是运行在JVM上。

R markdown对其它语言的支持都是通过定义语言引擎处理代码块来实现的，[这里](https://github.com/yihui/knitr-examples)有各种语言代码块的示例。如果有需要，也可以在[knitr](https://cran.r-project.org/web/packages/knitr/index.html)中自定义自己的语言引擎，无论是支持新的语言还是替换原有的语言引擎，具体可以参阅[该文档](https://bookdown.org/yihui/rmarkdown-cookbook/custom-engine.html)。

需要了解的是，knitr并不直接运行其它语言的代码，只是把代码块当成字符串传递给相应的语言，比如Python解析器，然后合并返回的结果到输出文档中。代码块的选项，比如message=TRUE等，则是告知knitr怎样处理代码块源码与返回的结果。所以这些代码块一般运行在独立的会话进程中。

其中只有R、Python、Julia三种语言的会话是连续的，即前面代码块中定义的变量，后面的代码块中可以引用。reticulate包让R与Python中的变量与函数可以互相引用，它会自动作变量类型转换，具体可参阅[该文档](https://rstudio.github.io/reticulate/)。R markdown中的SQL代码块依赖于DBI包，支持DBI API的数据库，如Oracle，SQLite可以在SQL代码块中直接执行SQL语句，并且执行结果可以返回成R data frame，相当于R、Python、SQL在R markdown是连通的。

其它的语言，可以在R或Python中分别调用，它们的会话是独立的，即前面代码块中定义的变量，后面的代码块中不能引用。但可以通过把结果赋值给R或Python变量，通过它们把各种语言不同的代码块连接起来，从而达到看起来像是在同一个会话中的效果。

本节演示混合使用R、Python、SQL、JavaScript、HTML/CSS等6中语言，这是税务系统环境中常用的。至于服务器端的Java、C、Fortran等资源，由R或Python脚本去调用即可。

本节以航空乘客数据airpassengers为例,从云端等各个数据源加载数据，然后插入SQLite内存数据库中，再执行SQL语句分析。实际操作中连接数据源后，可以直接执行SQL语句操作数据。因为有多个数据源，这里用SQLite做演示。

**大数据计算可在云端等强算力环境上完成，Rstudio Server、Shiny Server、R markdown等适合处理预处理好的中间或结果小数据集，作深度的分析和可视化。这是个不错的组合，它们也的确可以连接和增强不同的后端，让孙悟空乘上飞快的筋斗云。R与Python等有数万个各种各样的工具包，提供了各种各样强大的处理能力，连接着世界上庞大而充满活力的开源社区，这是一笔宝贵而丰富的IT资产，正是各个云平台所缺乏的，二者是相辅相成的互补关系，人们应该充分利用。**

**海纳百川，有容乃大。上善若水，万物并育而不相害，生态系统因此而得以平衡和良性循环。**

下面具体看看源码，各段源码拼接起来就是完整的Rmd源码文件。

1、加载用到的包。用showtext包处理ggplot2绘图中文标签的显示。

---

title: "多语言混合使用分析报告示例"

author: "Jean"

date: "`r Sys.Date()`"

output: html\_document

runtime: shiny

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

### 在R markdown中混合使用R、Python、SQL、JavaScript、HTML、CSS等6种语言

# 加载用到的包

```{r, message=FALSE}

library(shiny)

library(reticulate)

library(DBI)

library(RODPS)

library(ggplot2)

library(plotly)

library(showtext)

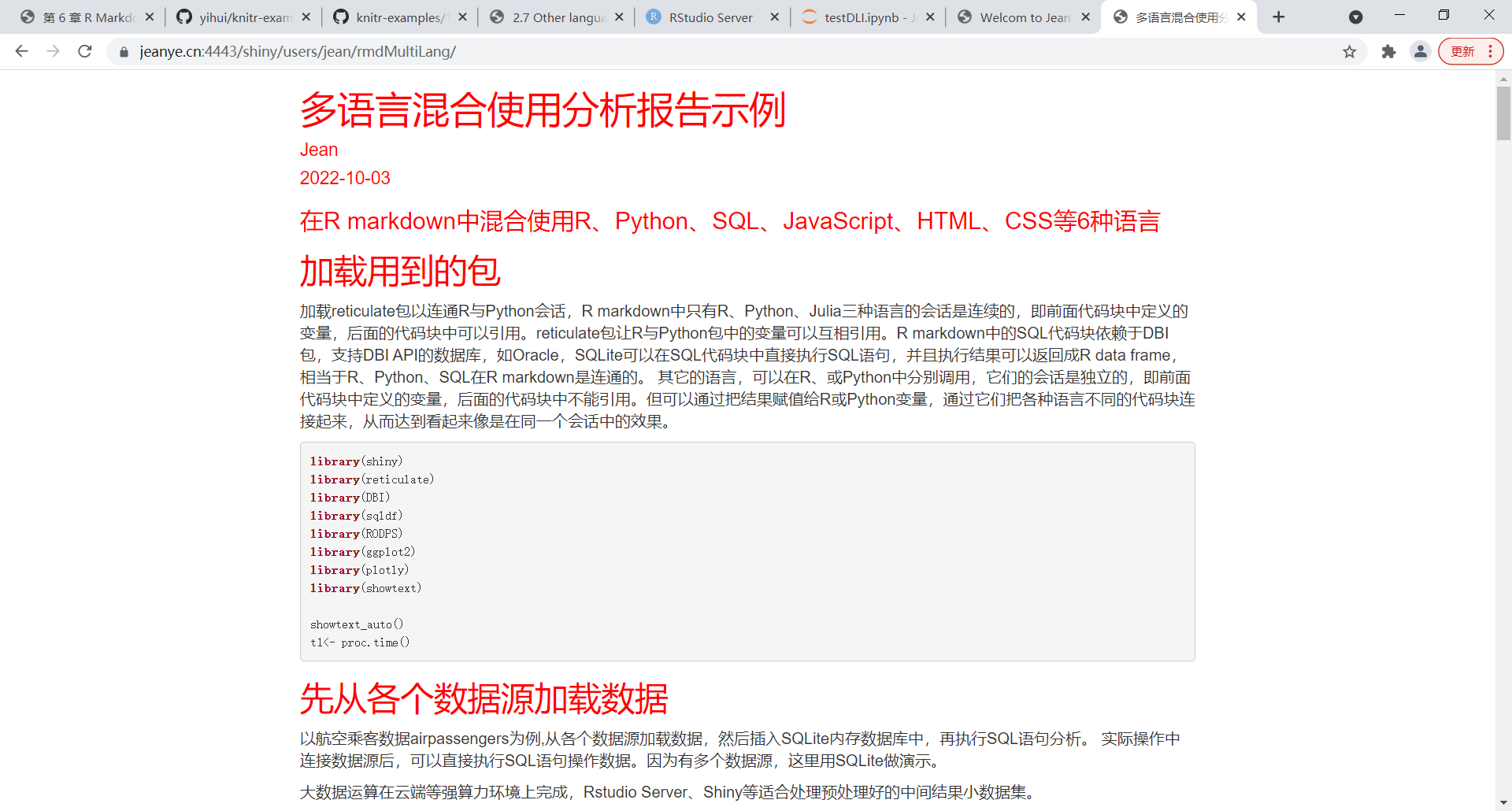
# 支持ggplot2绘图中文标注

showtext\_auto()

t1<- proc.time()

```

渲染结果：



2、从各种数据源加载数据，这里展示了R与Python的合用，以及相互引用变量。R可以连接几乎所有现有的数据源。阿里云有提供R接口包，直接用R语言，华为云用的是Python。需要连接其它云等后端的，至少会有Python接口包可用。Oracle有提供R接口包，很多历史数据还是在Oracle数据库里。

# 先从各个数据源加载数据

## 从Oracle加载数据

```{r, eval=FALSE}

# 如果连接Oracle

library(ROracle)

## create a Oracle instance and create one connection.

ora <- Oracle() ## or dbDriver("Oracle")

con <- dbConnect(ora, username = "user", password = "password", dbname = "SID")

rs <- dbSendQuery(con, "select \* from airpassengers ")

passengers <- fetch(rs, n = -1)

dbClearResult(rs) ## done with this query

```

## 从阿里云载入数据

```{r}

passengers<-rodps.sql("select \* from airpassengers")

names(passengers)<-c("month","passengers")

knitr::kable(head(passengers,5))

```

## 从华为云载入数据

### 设置local

```{python}

from dli.dli\_client import DliClient

from obs import \*

#这是自己添加的自定义python模块，存放访问DLI的access key与secret key，因为源码是网上可见的。

from dli import dlikeys

from obs import obskeys

from datetime import datetime

import locale

import time

import os

import sys

import pandas as pd

from io import BytesIO

import numpy as np

locale.setlocale(locale.LC\_ALL, 'en\_US.UTF-8')

print(locale.getlocale())

GMT\_DATE\_FORMAT = '%a, %d %b %Y %H:%M:%S GMT'

LONG\_DATE\_FORMAT = '%Y%m%dT%H%M%SZ'

SHORT\_DATE\_FORMAT = '%Y%m%d'

EXPIRATION\_DATE\_FORMAT = '%Y-%m-%dT%H:%M:%SZ'

now\_date = datetime.utcnow()

str\_date = now\_date.strftime(GMT\_DATE\_FORMAT)

print(str\_date)

```

### 初始化OBS client

```{python}

def init\_obs\_client():

AK = obskeys.obsAK

SK = obskeys.obsSK

server = 'https://obs.cn-south-1.myhuaweicloud.com'

# Constructs a obs client instance with your account for accessing OBS

obsClient = ObsClient(access\_key\_id=AK, secret\_access\_key=SK, server=server)

return obsClient

db\_name = 'geshui'

queue\_name = 'default'

bucketName = 'gsmx'

obsClient = init\_obs\_client()

```

### 从OBS加载数据

```{python}

key = 'airpassengers.csv'

cols = ["month","passengers"]; types = {"month":str,"passengers":int}

resp = obsClient.getObject(bucketName, key, loadStreamInMemory=True)

fileBytes = BytesIO( resp.body.buffer )

passengers = pd.read\_csv(fileBytes,header=1,names=cols,dtype=types,encoding='utf-8')

# print(passengers)

```

### 在R脚本中引用Python变量

通过加'py$'前缀引用。

```{r}

knitr::kable(head(py$passengers,5))

```

### 在Python中引用R变量

通过加'r.'前缀引用。

```{python}

print(r.passengers[0:5])

```

### 设置数据湖客户端

```{python}

def init\_aksk\_dli\_client():

auth\_mode = 'aksk'

region = 'cn-south-1'

project\_id = dlikeys.dli\_project\_id

ak = dlikeys.dliAK

sk = dlikeys.dliSK

dli\_client = DliClient(auth\_mode=auth\_mode, region=region, project\_id=project\_id,ak=ak, sk=sk)

return dli\_client

def run\_sql(dli\_client, db\_name, queue\_name, sql):

# execute SQL

try:

sql\_job = dli\_client.execute\_sql(sql, db\_name)

result\_set = sql\_job.get\_result()

except DliException as e:

print(e)

return

if result\_set.row\_count == 0:

return None

return result\_set

dli\_client = init\_aksk\_dli\_client()

```

### 从数据湖DLI加载数据

```{python}

db\_name = 'geshui'

queue\_name = 'default'

# OBS bucket name

bucketName = 'gsmx'

t1 = time.time()

month = [] ; passengers=[]

sql = "select month,passengers from airpassengers;"

result\_set = run\_sql(dli\_client, db\_name, queue\_name, sql)

print("loading data......")

for row in result\_set:

cols = row.columns

month.append(cols[0])

passengers.append(cols[1])

t2 = time.time()

print("time: ",t2-t1)

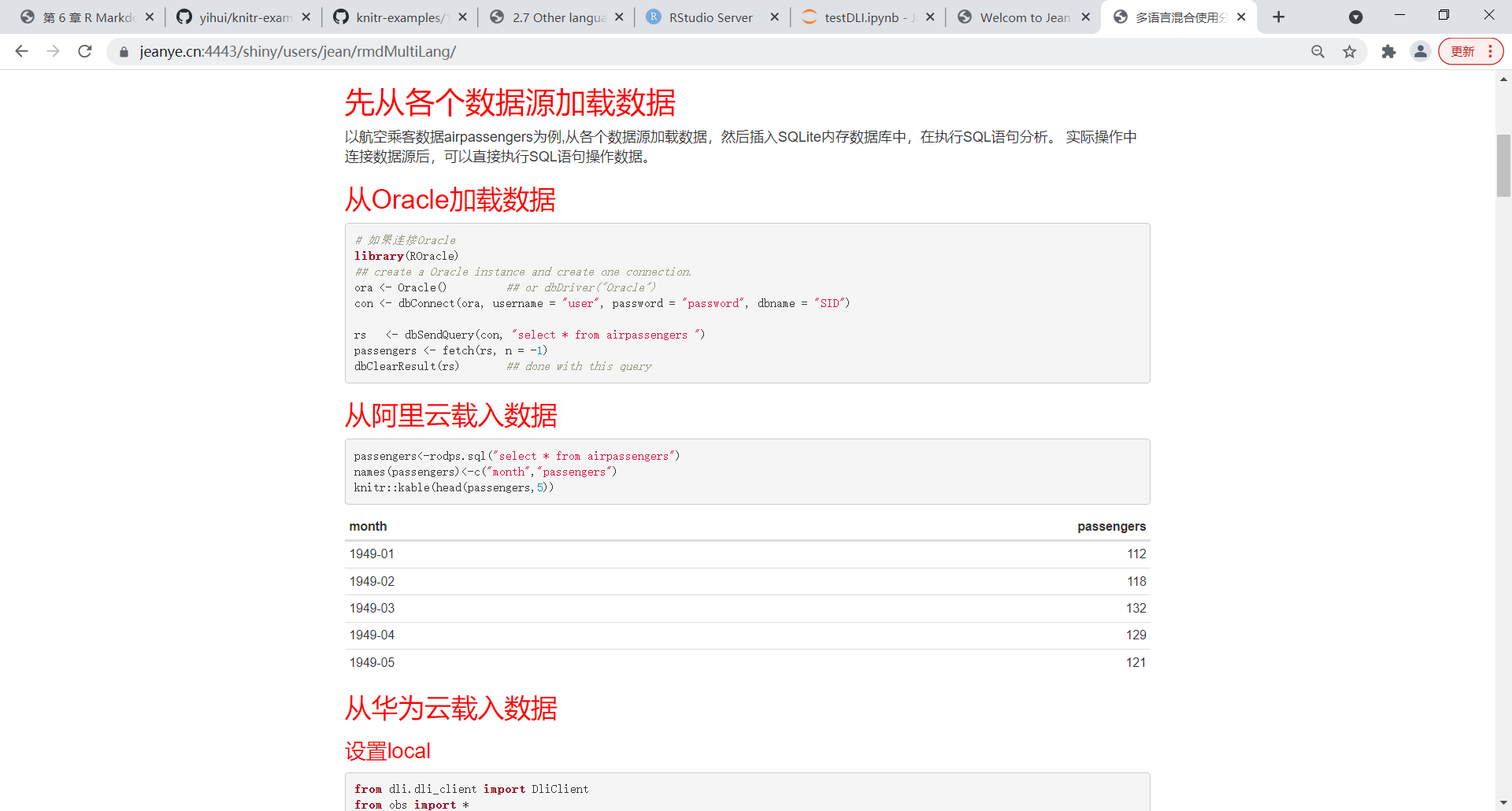
# 从 list建立 pandas data frame

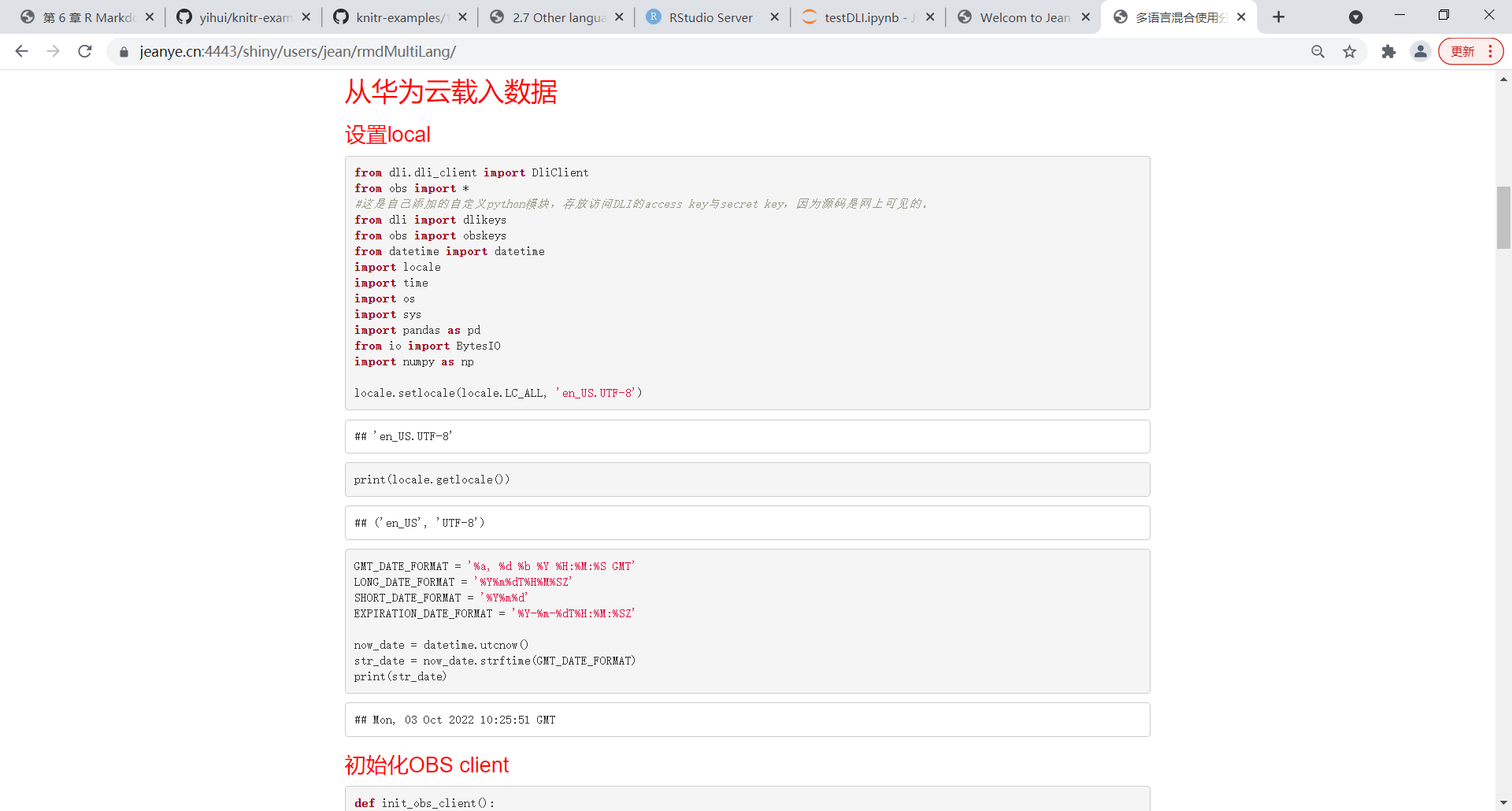
passengers = pd.DataFrame({"month":month,"passengers":passengers})

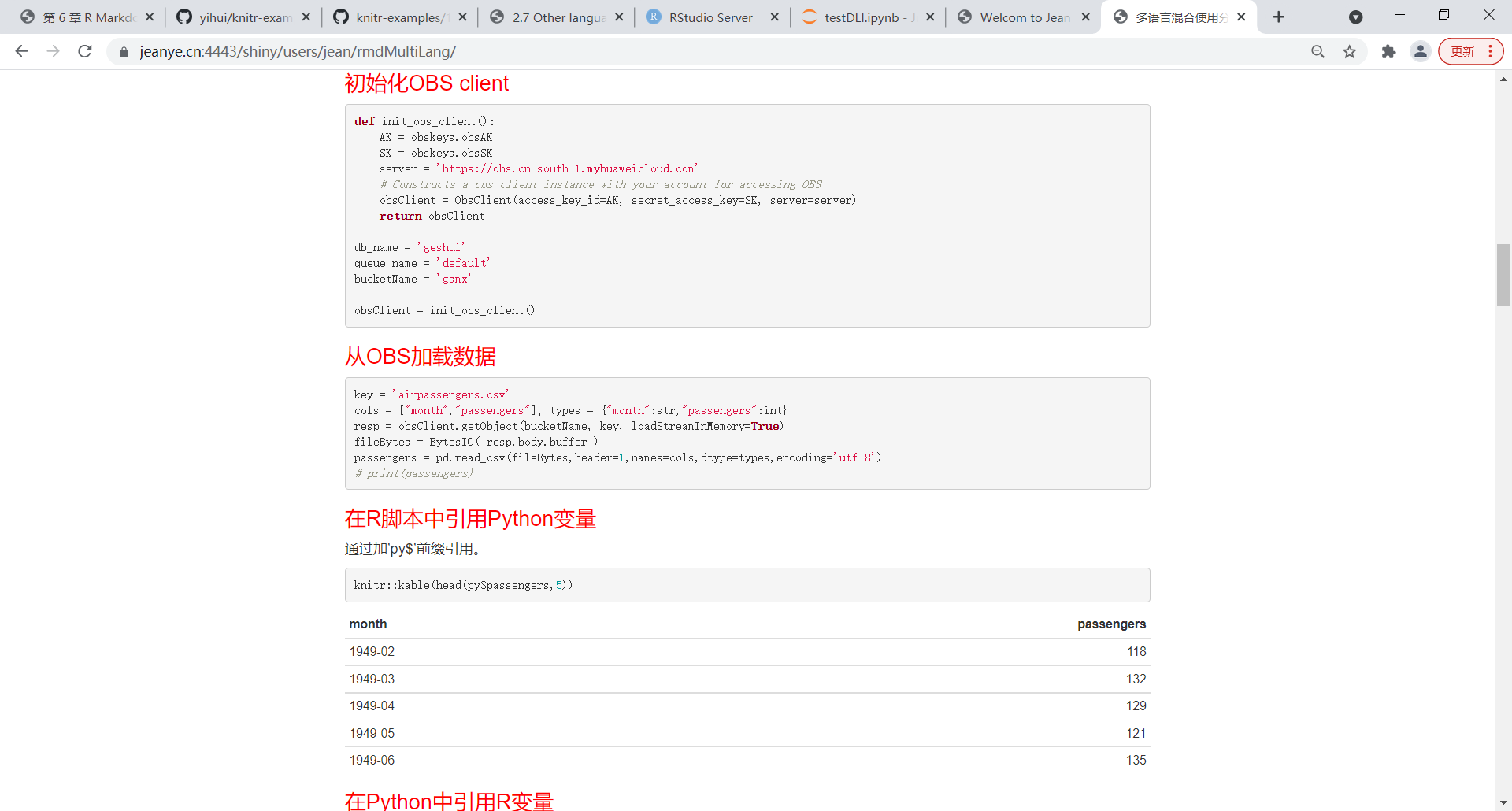
print(passengers[0:5])

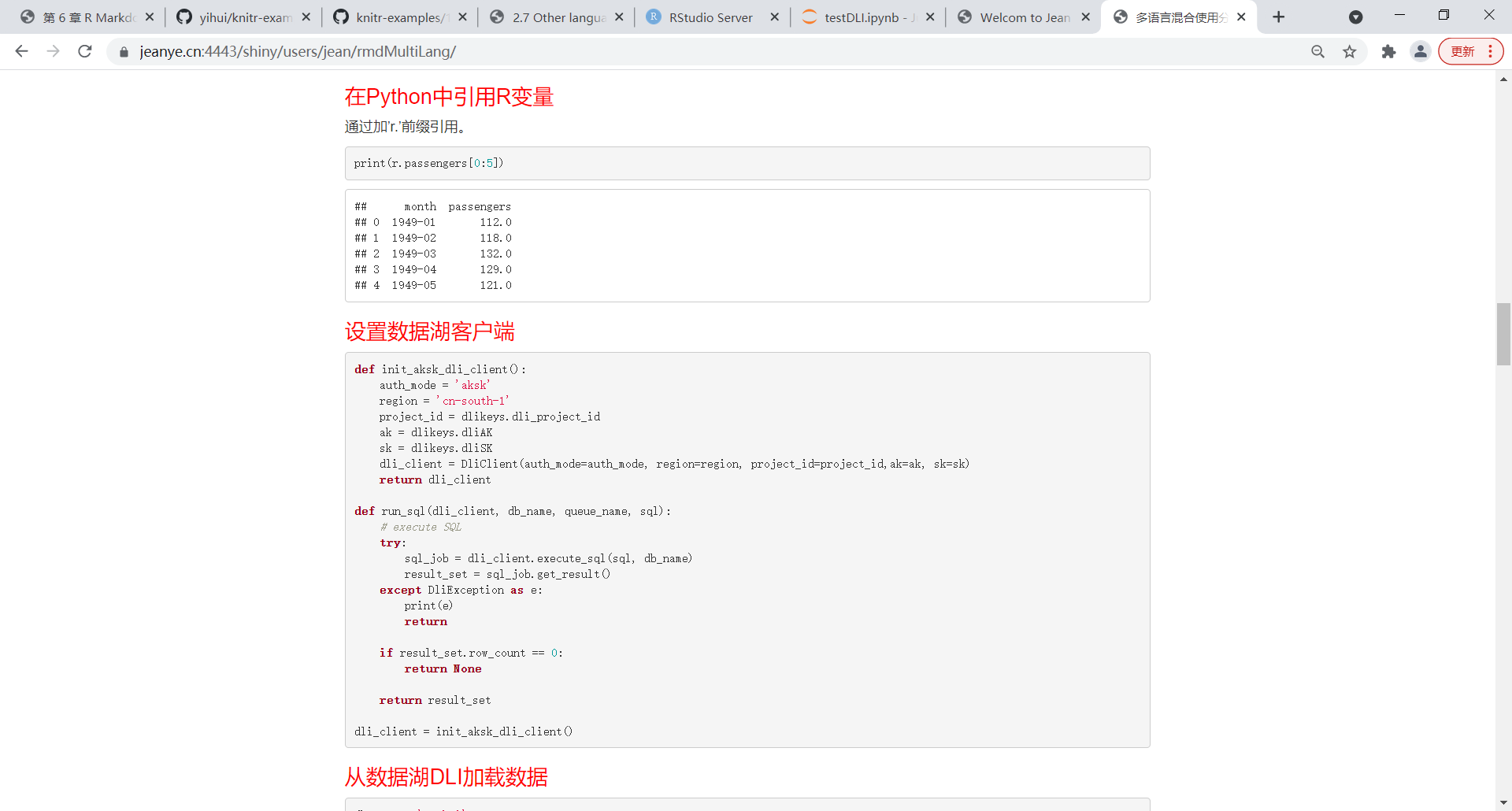
```

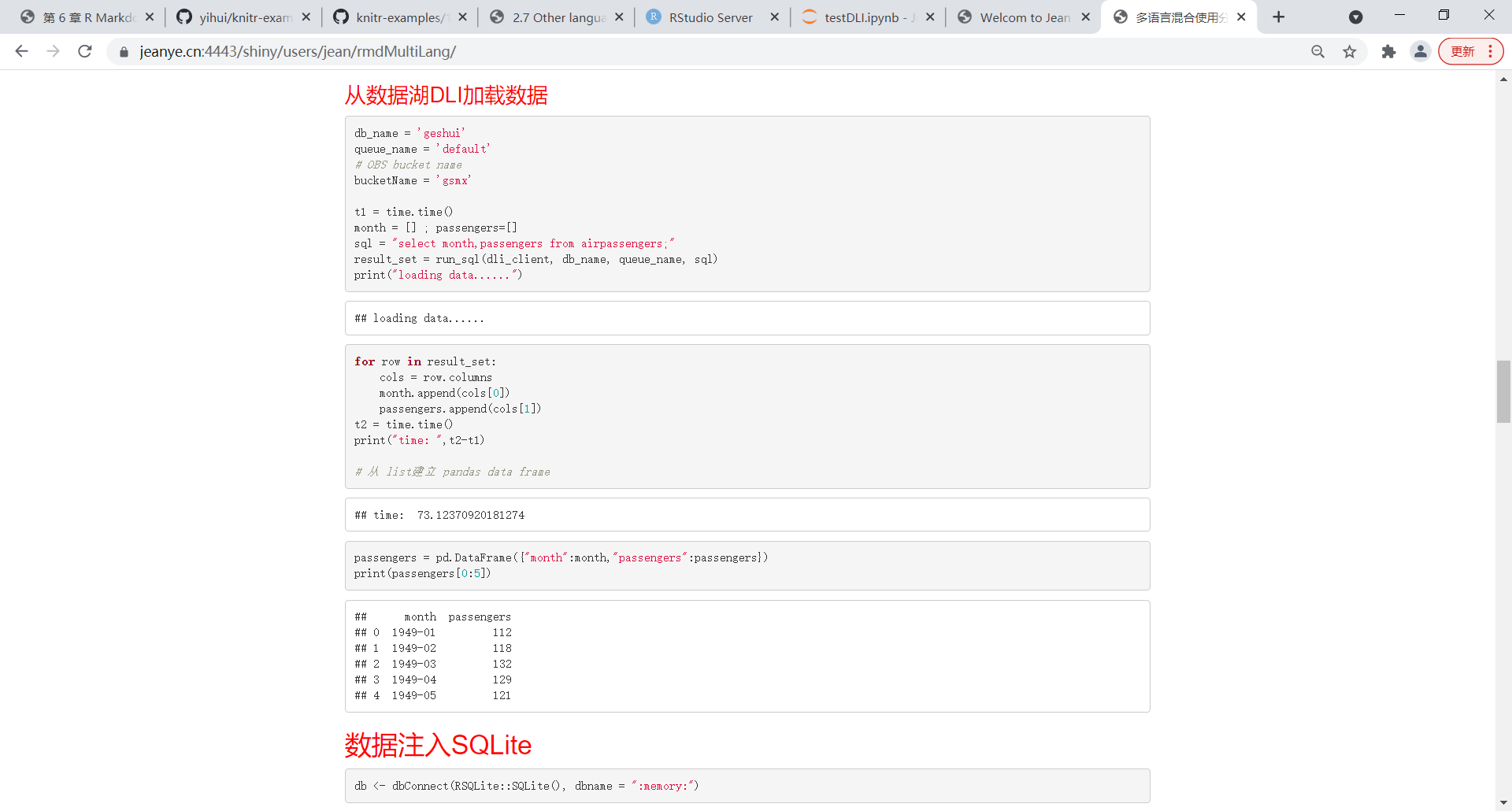
渲染结果：











3、为演示直接执行SQL，把各种数据源收集的数据，注入内存的SQLite数据库中。

## 数据注入SQLite

```{r message=FALSE}

db <- dbConnect(RSQLite::SQLite(), dbname = ":memory:")

```

###

如果SQL代码块上的运行按钮不见了，或者运行SQL代码块时出现错误"Error: unexpected symbol"

按该帖子panporter描述的办法解决。

https://github.com/rstudio/rstudio/issues/6220

Steps to reproduce:

1.Create a simple RMD file with one SQL chunk.

2.Switch "Global Options - R Markdown - Show output inline for all R Markdown documents" OFF.

3.Run SQL chunk.

Results in error in console:

Error: unexpected string constant in "SELECT 1"

4.Reload file: The play button disappears.

5.Switch "Global Options - R Markdown - Show output inline for all R Markdown documents" back ON.

And reload file.

Play button reappears. SQL chunk runs as expected.

This seems to be origin of problem #8786 as well.

My setting:

RStudio

Version 1.4.1717

Linux and Windows

```{sql, connection=db}

DROP TABLE IF EXISTS airpassengers

```

### 引用Python返回的数据

```{r}

dbWriteTable(db,"airpassengers", py$passengers)

```

### 执行SQL语句

如果没有指定output.var选项，SQL语句执行结果显示在文档输出中。

```{sql connection=db}

SELECT \* FROM airpassengers;

```

如果指定了output.var选项，SQL语句执行结果不会显示在文档输出中，而是保存到指定的R变量中。

下面按年统计各年乘客的累计数量，存入y\_sum data frame中。

```{sql connection=db, output.var="y\_sum"}

SELECT substring(month,1,4) as year, sum(passengers) as passengers

from airpassengers

group by substring(month,1,4)

order by substring(month,1,4)

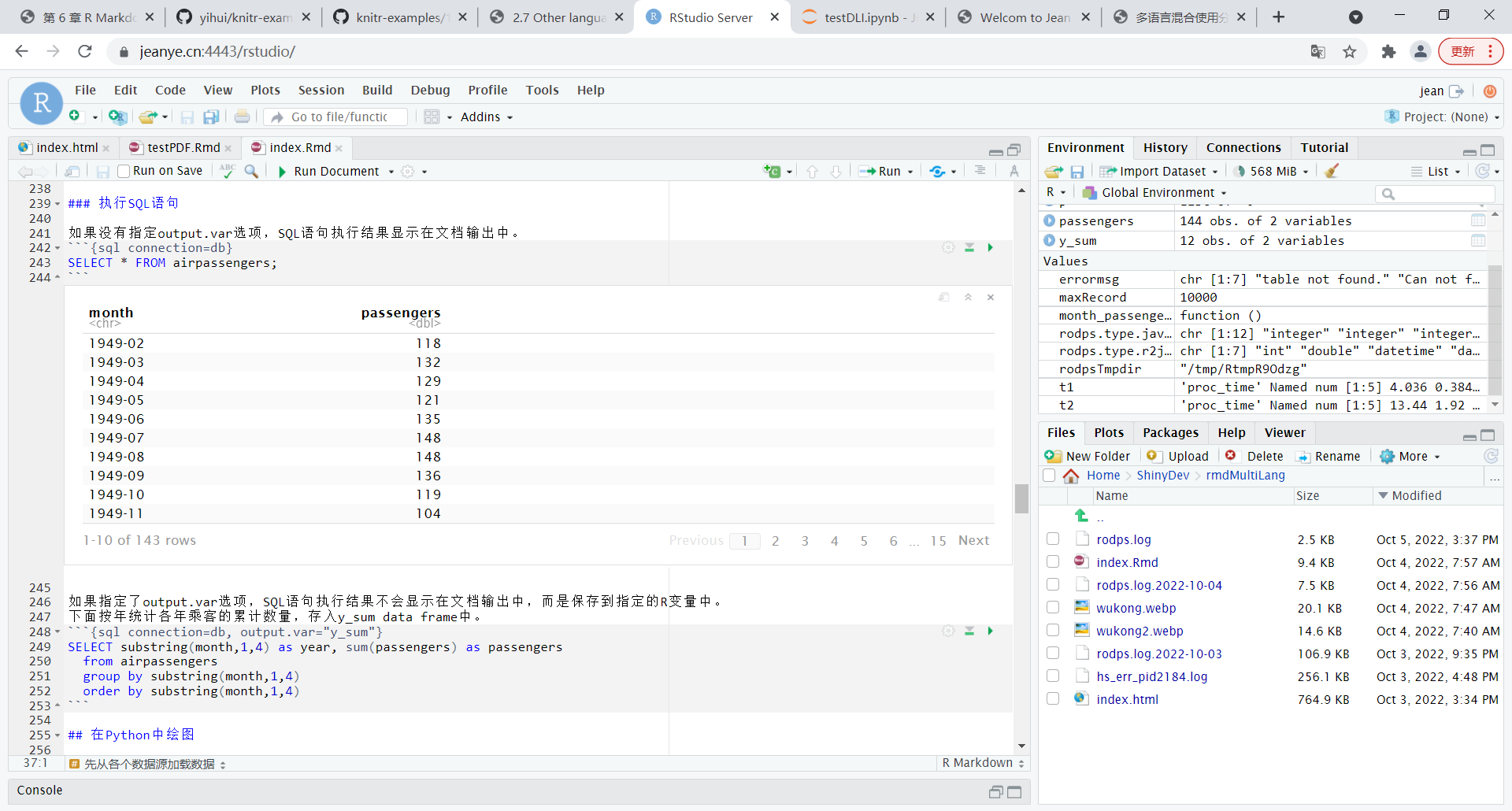
```

渲染结果：





SQL代码块的选项在执行结果里没有显示出来，在Rstudio里编辑Rmd源码时，执行结果是可翻页的交互式表格，与渲染结果只显示一页是不同的。



4、在Python中绘图。

## 在Python中绘图

```{python}

import matplotlib.pyplot as plt

# fig = plt.figure(figsize = (3, 2)) # 没有这一行，Rstudio中会显示在Plots窗口

plt.bar(r.y\_sum["year"],r.y\_sum["passengers"], color="orange") # 通过"r."前缀引用R中的变量

plt.xlabel("年度")

plt.ylabel("累计乘客")

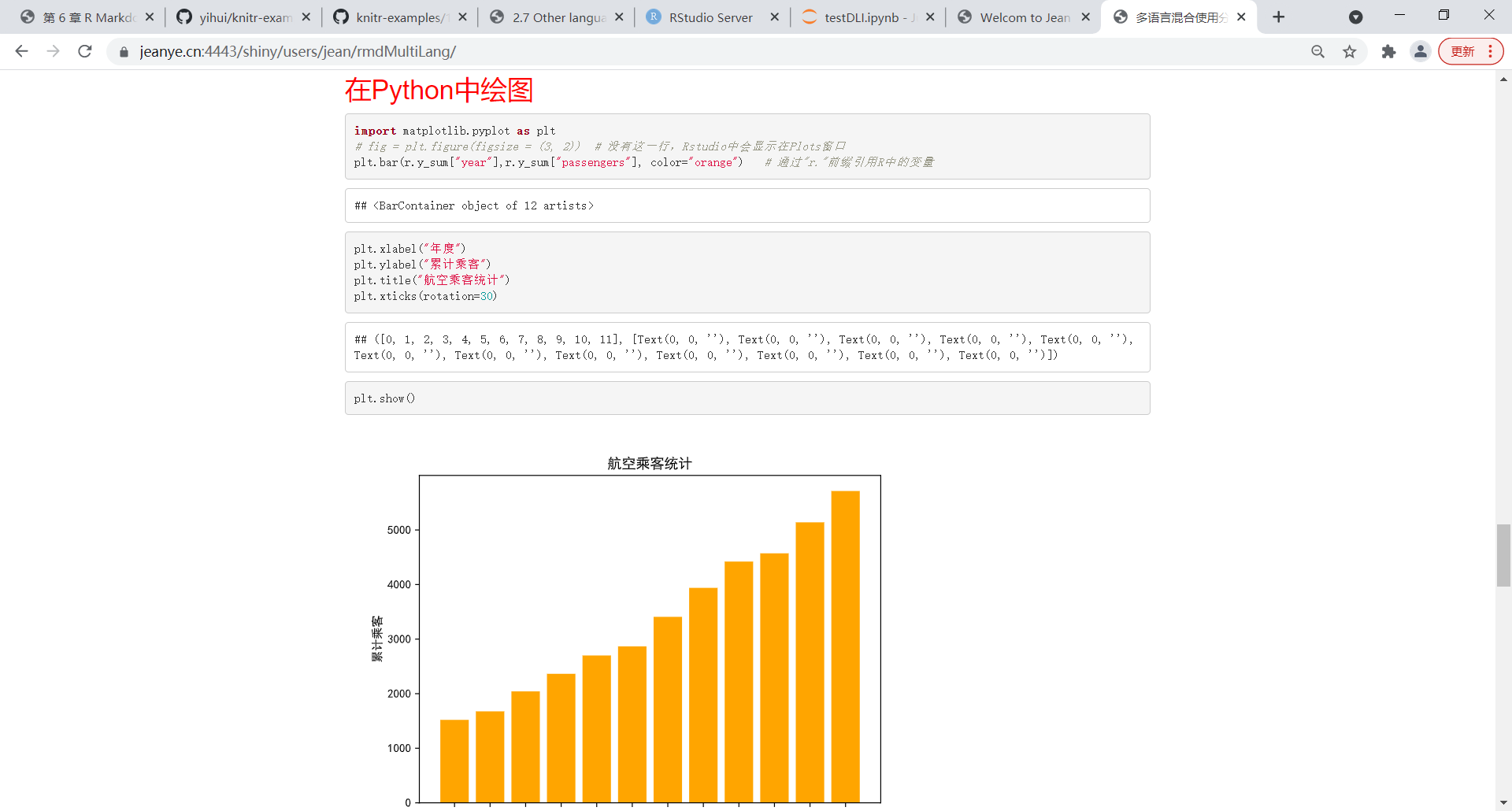
plt.title("航空乘客统计")

plt.xticks(rotation=30)

plt.show()

```

渲染结果：



5、嵌入基于JavaScript的HTML Widget，提供浏览器端的即时交互，有很多各种各样的HTML Widget，具体可以参阅[该文档](https://www.htmlwidgets.org/index.html)。

## HTML widgets

HTML widgets框架由R 软件包htmlwidgets实现，提供了browser端交互式的控件，它与Shiny反应式控件的区别是，这些widget是纯JavaScript实现，它的反应只在浏览器端，不涉及服务器端。而Shiny的反应式UI组件在浏览器端依赖于htmlwidgets包，在服务器端实现反应机制。

此处用ggplotly画一个交互式的条形图，由JavaScript库ploty支持，在条形图上移动鼠标会显示相应的数据。

```{r}

# https://stackoverflow.com/questions/13067751/how-to-run-r-scripts-on-servers-without-x11

options(bitmapType='cairo') # 服务器端没有X11支持，要打开这个选项。

p <- ggplot(data = y\_sum, aes(x = year, y = passengers)) +

geom\_bar(stat="identity", color="orange", fill="orange")+

ggtitle("航空乘客统计图")+

xlab("年度")+

ylab("乘客数量")

ggplotly(p)

```

渲染结果：



6、交互式的Shiny文档，提供服务器端的即时交互，第二节有详细介绍。

## Shiny 文档

交互式文档，画选定年份的航空乘客数量条形图。

### 选择年度

```{r}

selectInput(

'year', label = '年度',

choices = y\_sum["year"], selected = "1950"

)

```

### 提取年度数据

```{r}

month\_passengers<-reactive({py$passengers[which(substr(py$passengers$month,1,4)==input$year),]})

```

### 画图

```{r}

renderPlot({

ggplot(data = month\_passengers(), aes(x = substr(month,6,7), y = passengers)) +

geom\_bar(stat="identity", color="orange", fill="orange")+

ggtitle("航空乘客统计图")+

xlab("年度")+

ylab("乘客数量")

})

```

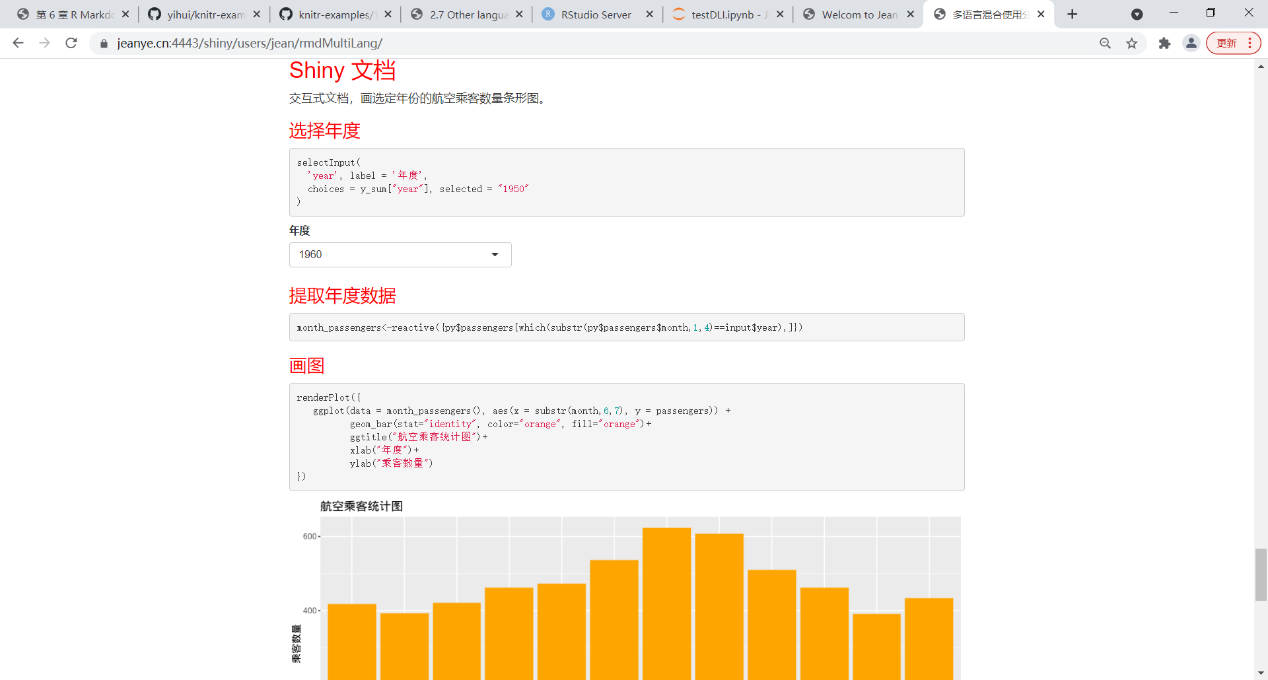
### 画表

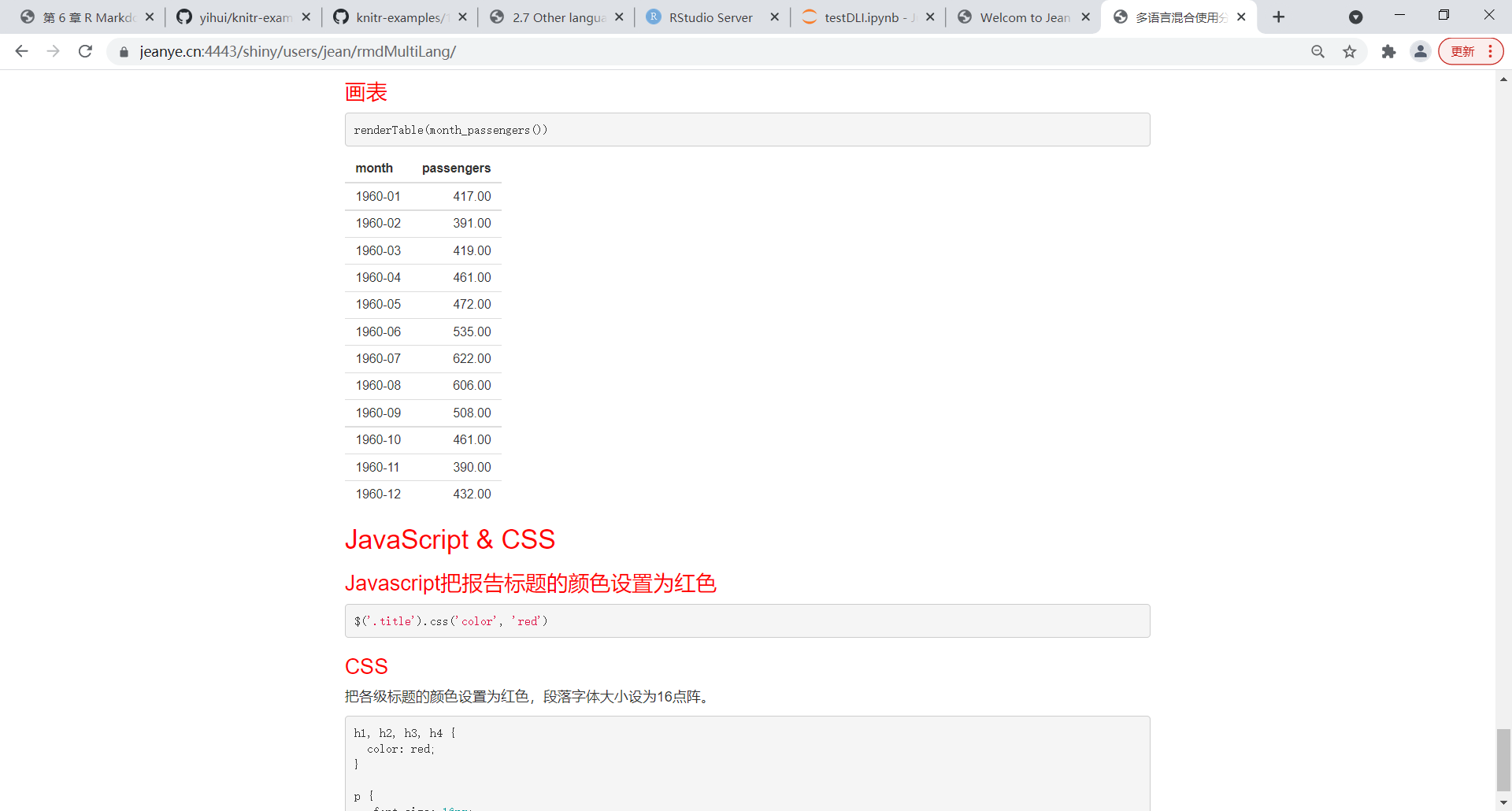
```{r}

renderTable(month\_passengers())

```

渲染结果：





7、嵌入JavaScript脚本与CSS脚本改善报告的可视化效果。渲染效果见上面各图。

## JavaScript & CSS

### Javascript把报告标题的颜色设置为红色

```{js}

$('.title').css('color', 'red')

```

### CSS

把各级标题的颜色设置为红色，段落字体大小设为16点阵。

```{css, echo=TRUE}

h1, h2, h3, h4 {

color: red;

}

p {

font-size: 16px;

}

```

8、直接嵌入HTML脚本。

### HTML

在页面中直接插入HTML代码。

```{r}

HTML("<div style='color: red; font-size: 20px; text-align: center;'>")

```

总耗时：

```{r}

t2<-proc.time()

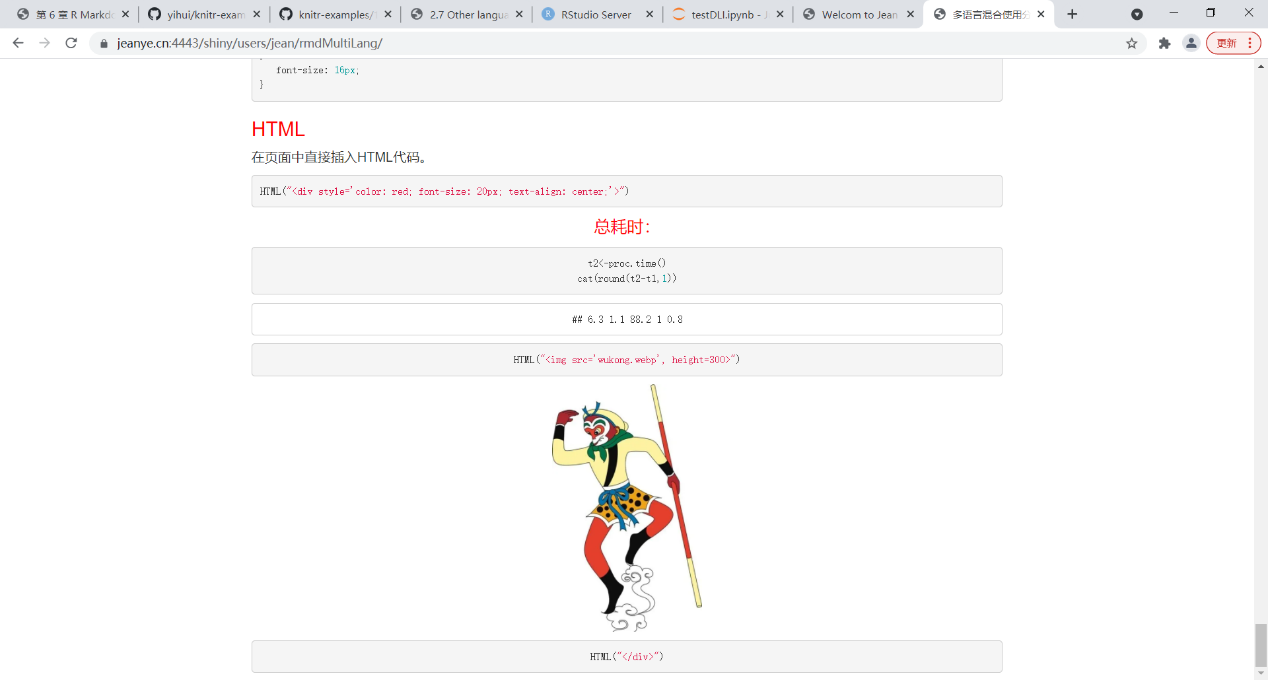
cat(round(t2-t1,1))

HTML("<img src='wukong.webp', height=300>")

HTML("</div>")

```

渲染结果：



第五节 用 R markdown生成动态幻灯片

前面几节介绍了用R markdown生成动态深度分析报告，有时候需要对这些报告进行概览性的简要介绍，这就需要幻灯片。这里的动态指的是数据动态，如前面各节所示，而非幻灯片的动画效果，所以它是数据嵌入各种程序语言脚本渲染的结果。另外，平常字处理所见的多数是微软的幻灯片PPT，R markdown还支持其它几种格式的幻灯片，有基于PDF的Beamer，以及基于HTML的ioslides、Slidy、Xaringan等，具体可参阅该[文档](https://bookdown.org/yihui/rmarkdown/presentations.html)及[例子](https://github.com/yihui/knitr-examples)。下面看看简单的例子。

1. MS Power Point幻灯片，Rstudio->File->New File->R markdown->Presentation->Power Point。

---

title: "Power Point幻灯片示例"

author: "Jean"

date: "`r Sys.Date()`"

output: powerpoint\_presentation

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = FALSE)

```

## R Markdown

这是一个R Markdown 幻灯片。 Markdown 是编写 HTML、PDF、及 MS Word 的简单语法。 详情请阅 <http://rmarkdown.rstudio.com>.

## 带要点的幻灯片

- 要点 1

- 要点 2

- 要点 3

## 幻灯片输出R语言数据表格

```{r cars, echo = TRUE}

summary(cars)

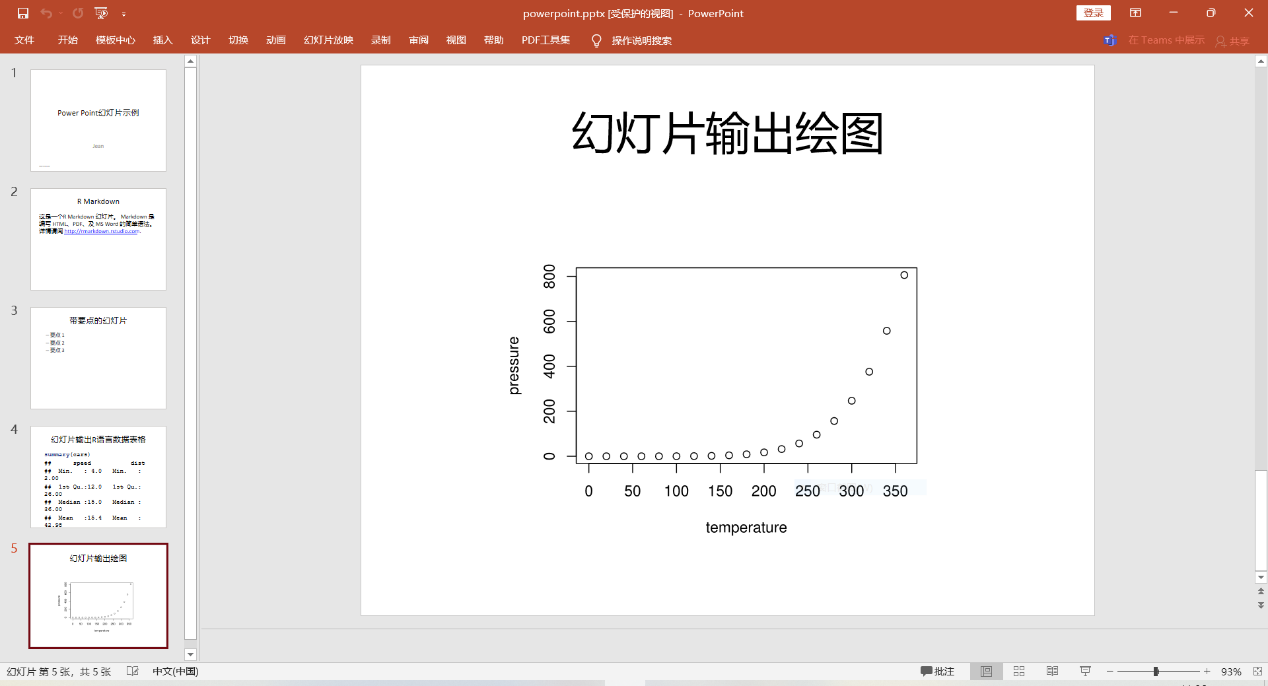
```

## 幻灯片输出绘图

```{r pressure}

plot(pressure)

```



1. PDF Beamer幻灯片，需要在YAML头部加入中文支持的描述，Rstudio->File->New File->R markdown->Presentation->PDF。

---

title: "PDF Beamer幻灯片示例"

author: "Jean"

date: "`r Sys.Date()`"

output:

beamer\_presentation:

latex\_engine: xelatex

header-includes:

- \usepackage{ctex}

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = FALSE)

```

## R Markdown

这是一个R Markdown 幻灯片。 Markdown 是编写 HTML、PDF、及 MS Word 的简单语法。 详情请阅 <http://rmarkdown.rstudio.com>.

## 带要点的幻灯片

- 要点 1

- 要点 2

- 要点 3

## 幻灯片输出R语言数据表格

```{r cars, echo = TRUE}

summary(cars)

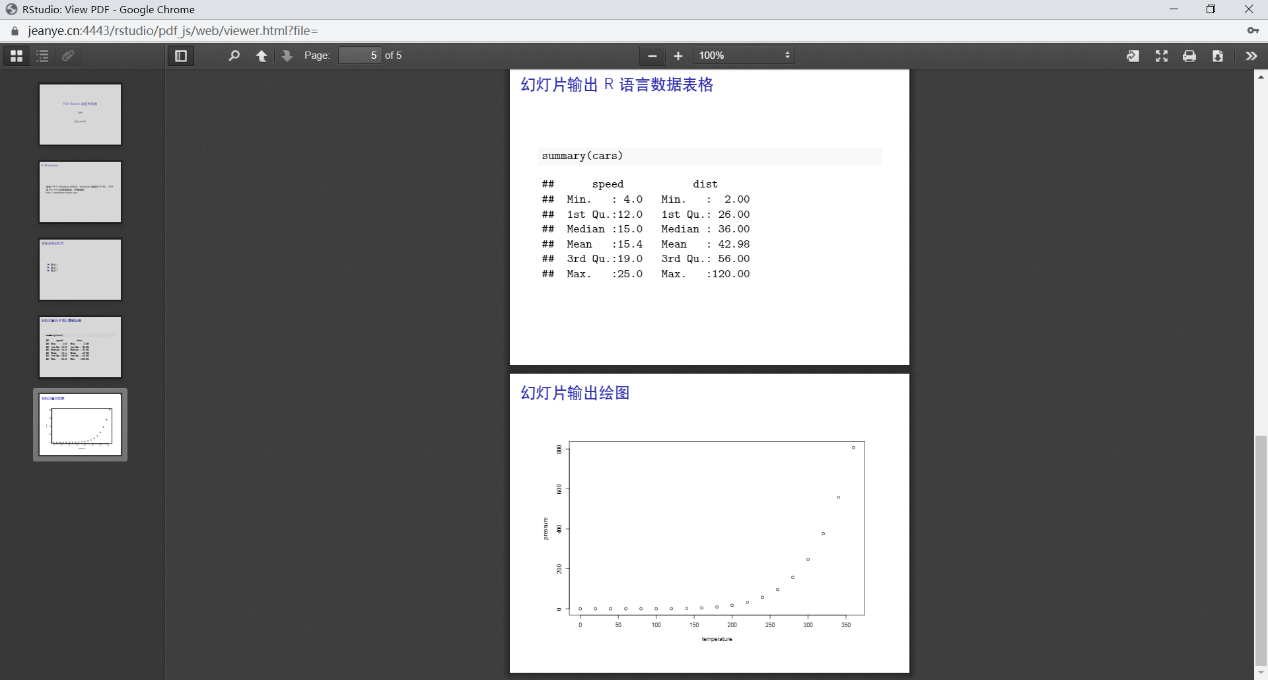
```

## 幻灯片输出绘图

```{r pressure}

plot(pressure)

```

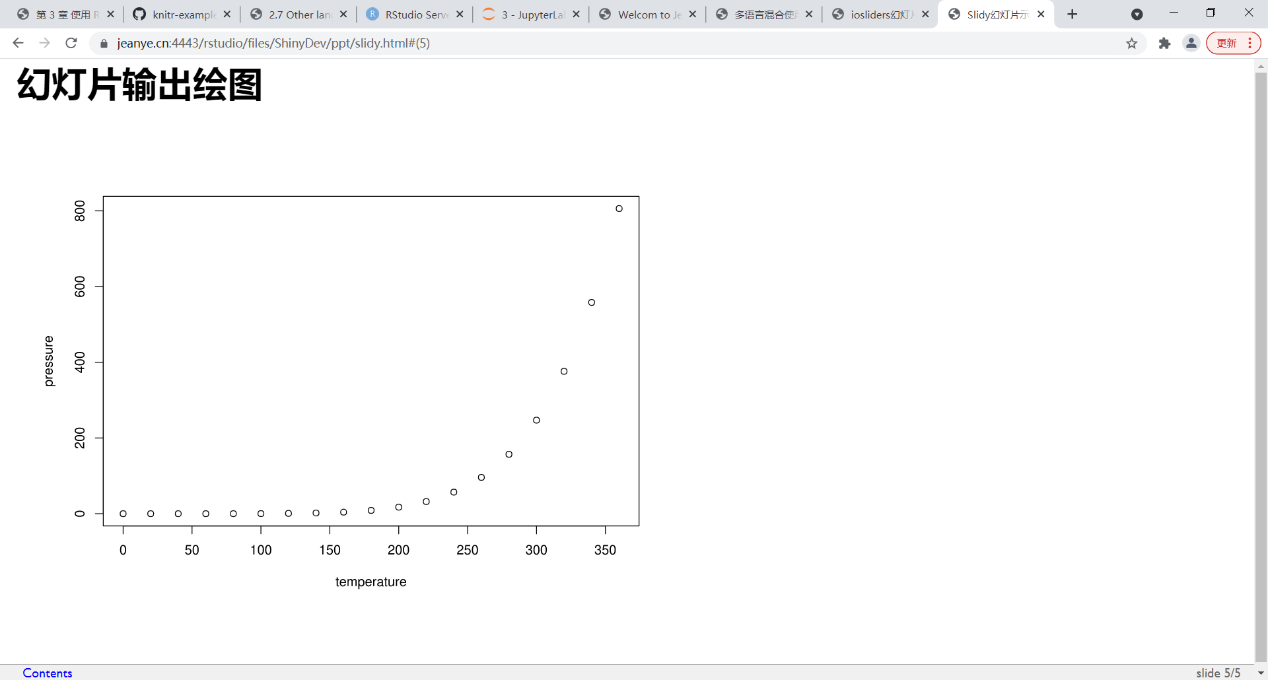


1. ioslides、Slidy幻灯片，把power point幻灯片例子的输出改成ioslides\_presentation或slidy\_presentation即可。创建过程同上。

Iosliders幻灯片：



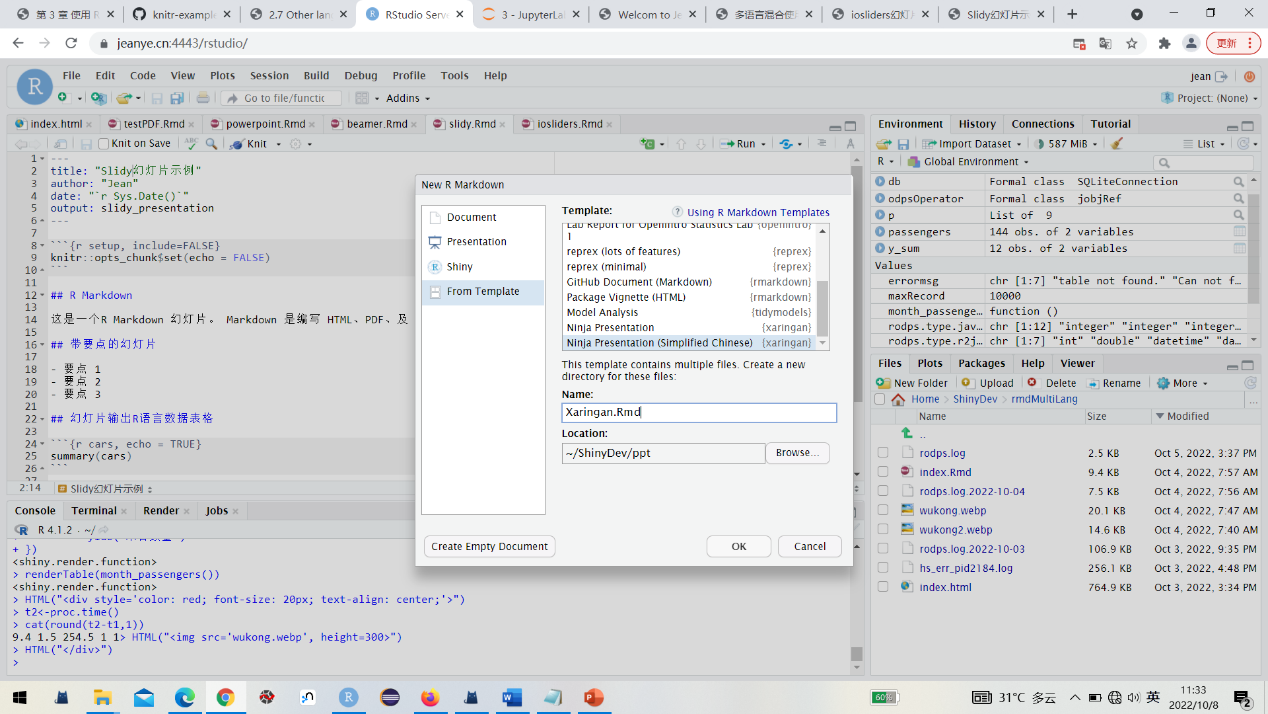
Slidy幻灯片：



1. Xaringan幻灯片。忍者幻灯片由Rstudio的谢益辉开发，对HTML幻灯片作了各个方面的增强，添加了很多方便的功能。需要先安装xaringan包及其扩展包，它也用showtext包来处理ggplot2绘图的中文标示：

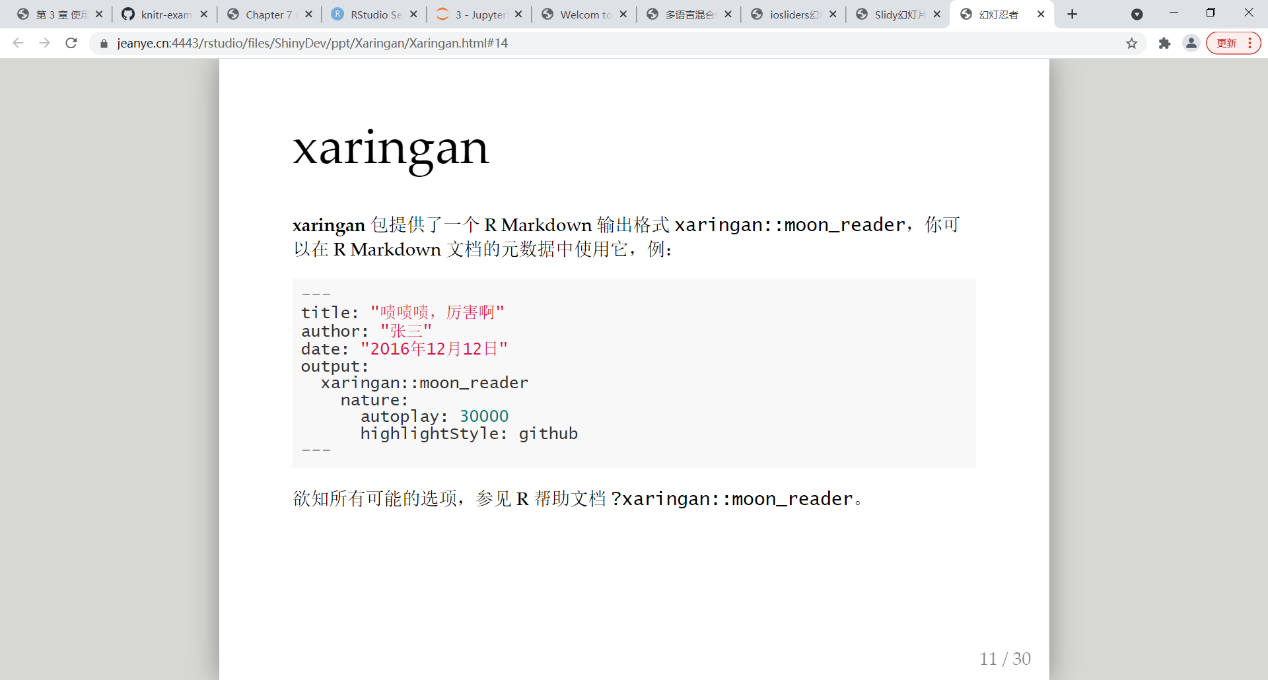
install.packages(c(“xaringan”, “xaringanExtra”, “xaringanthemer”))

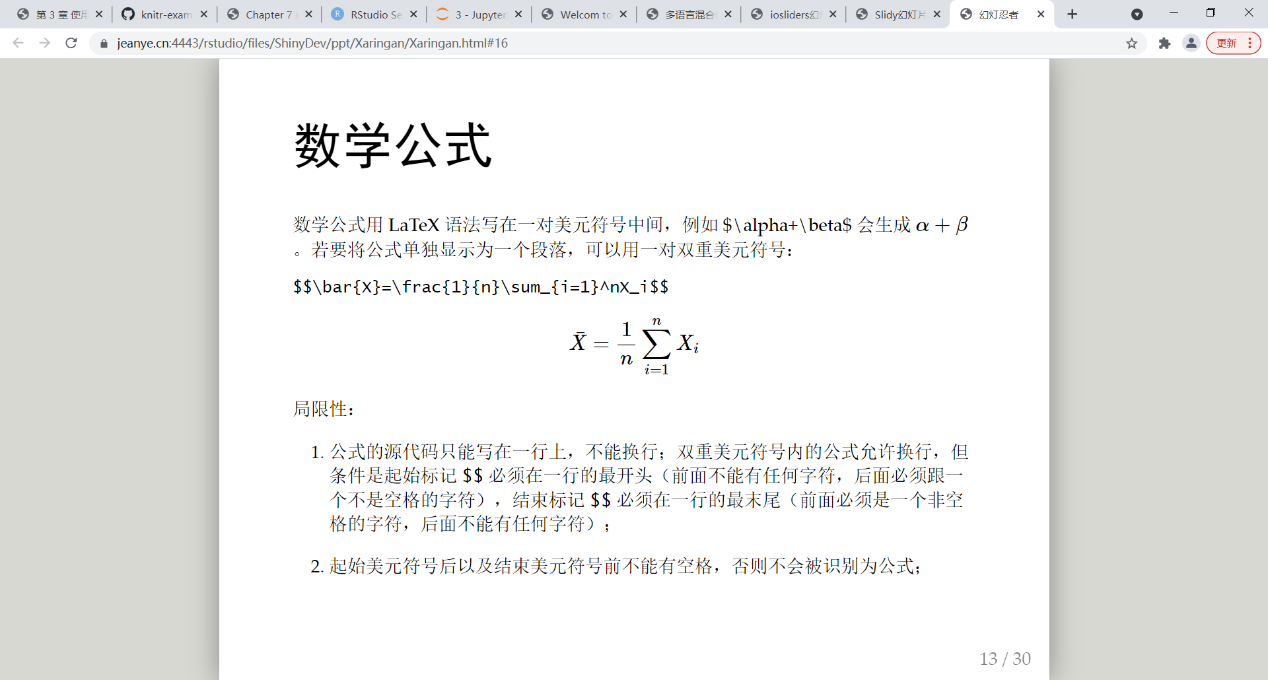
然后在Rstudio菜单上File->New->R markdown->From Template，然后选择 “Ninja Presentation (Simplified Chinese)”。填写必要信息之后，即可在一个新目录中创建“写轮眼”忍者幻灯片。创建的示例幻灯片本身有详细的介绍，这里就不展开，详细可看示例幻灯片的源码及下面的[文档](https://bookdown.org/yihui/rmarkdown/xaringan.html)。

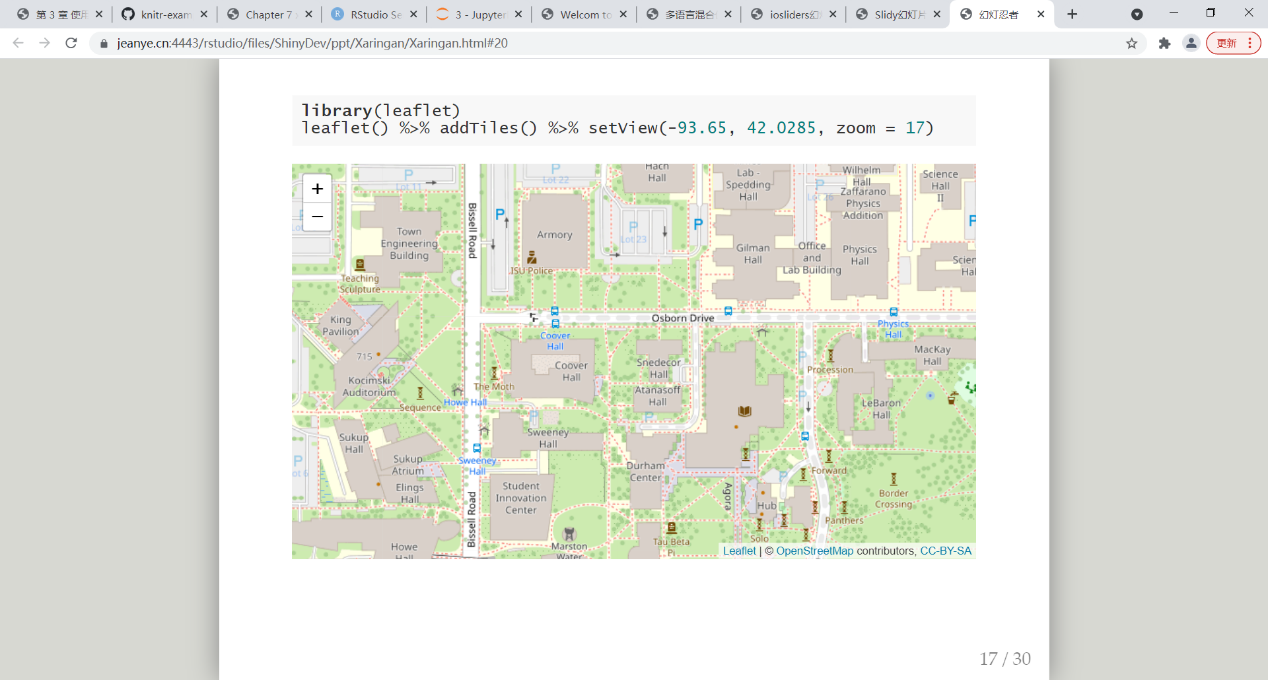


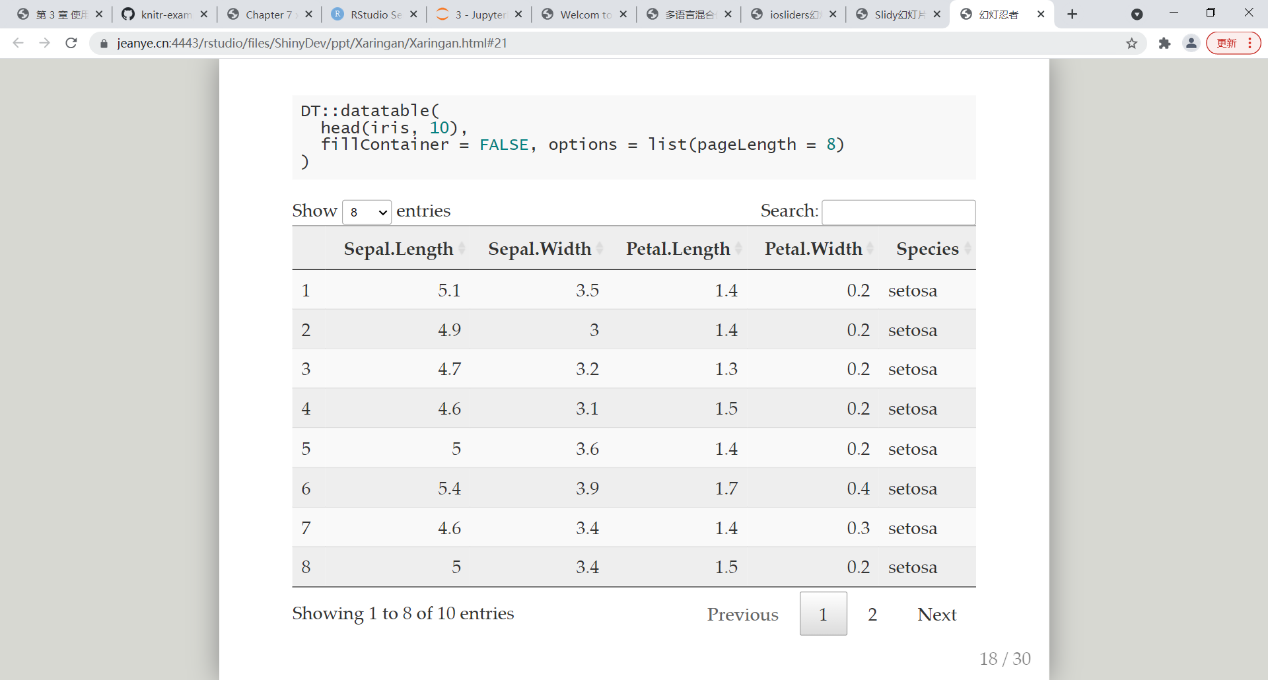
忍者幻灯片效果：











Rmd源码的开头部分，主要是YAML的定义，剩余的代码就不列出了，自己新建一个就有：

---

title: "幻灯忍者"

subtitle: "写轮眼"

author: "谢益辉"

institute: "RStudio, PBC"

date: "2016/12/12"

output:

xaringan::moon\_reader:

css: [default, zh-CN.css]

lib\_dir: libs

nature:

highlightStyle: github

highlightLines: true

countIncrementalSlides: false

---

background-image: url(https://upload.wikimedia.org/wikipedia/commons/b/be/Sharingan\_triple.svg)

```{r setup, include=FALSE}

options(htmltools.dir.version = FALSE)

```

图片来源：[Wikimedia Commons](https://commons.wikimedia.org/wiki/File:Sharingan\_triple.svg)

---

class: center, middle

# xaringan

### /ʃaː.'riŋ.ɡan/

---

class: inverse, center, middle

# 出发！

---

基于HTML的幻灯片运行在浏览器中，因为有HTML/CSS及JavaScript的加持，可以充分利用各种库与widget，可以创造出比PPT及Beamer幻灯片丰富得多得视觉效果，有兴趣的话可以试一下。

第六节 用 R markdown生成仪表板

  仪表板是数据的简要表示，应用的场景不少，人们都喜欢这种形式，大屏上一刷，高大上的样子。要做好事情当然要踏踏实实地干，分析建模的苦功在水下，人们看不到也看不懂。但要讲好故事也要做好表面功夫，一个精美的仪表板往往能得到人们的认同，所以也不差那么几百行代码了。

  R markdown通过 [flexdashboard](https://cran.r-project.org/web/packages/flexdashboard/index.html)包支持仪表板，它是R markdown的一个扩充。Shiny也通过 [shinydashboard](https://rstudio.github.io/shinydashboard/)包提供仪表板的支持，本篇先介绍flexdashboard的一个实例，因为篇幅的关系，将在下一篇中再介绍shinydashboard的实现，以便比较一下。

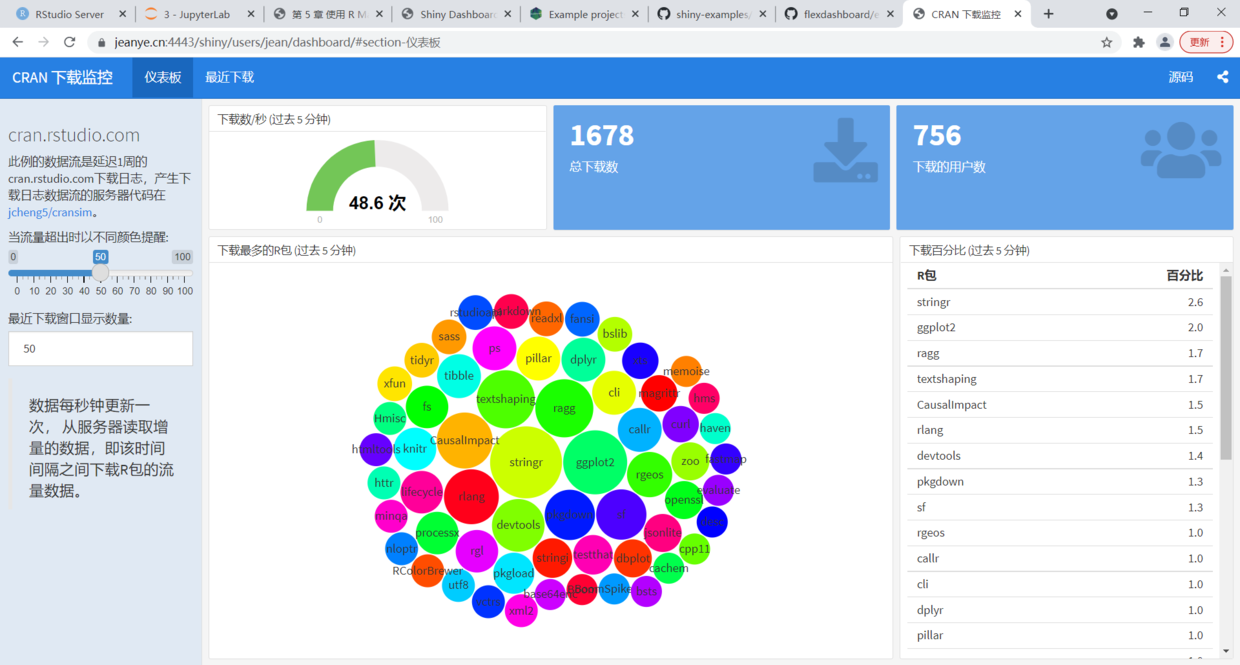
  本节不介绍flexdashboard包的具体用法，可参阅[《 R Markdown: The Definitive Guide》](https://bookdown.org/yihui/rmarkdown/)一书的第5章 [《Dashboards》](https://bookdown.org/yihui/rmarkdown/dashboards.html)。

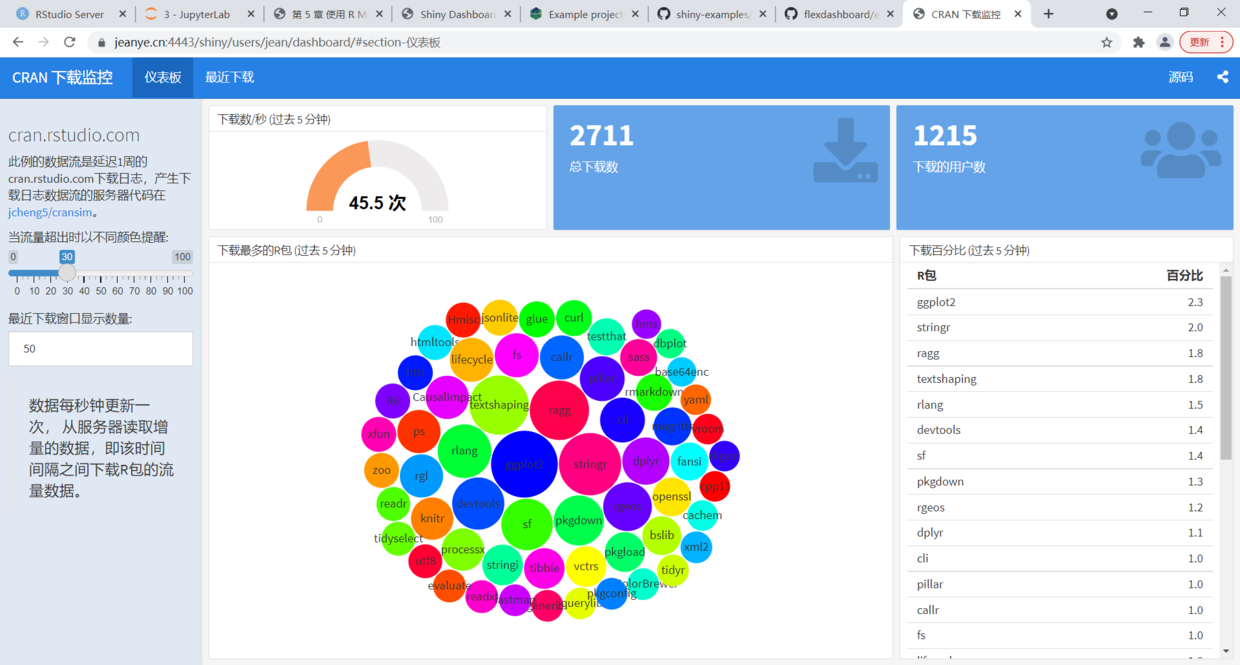
  本例通过从Github上部署的服务（项目）读取Rstudio CRAN上R软件包的下载日志，监控下载流量的变化，实时监控是仪表板的典型用法，就像驾驶位上的仪表盘一样。这里是延迟1周的备份下载日志，不过作为演示也足够了。 [项目源码](https://github.com/rstudio/flexdashboard/tree/main/examples/04_shiny-cran-downloads)在github上，[J.J.Allaire](https://github.com/jjallaire)的作品，简要精当（实际上是Rstudio三巨头合作的示例，bubbles包与扫描日志的服务器端代码是[CTO Joe Cheng](https://github.com/jcheng5)，shinySignals包是[首席科学家Hadley Wickham](https://github.com/hadley)）。

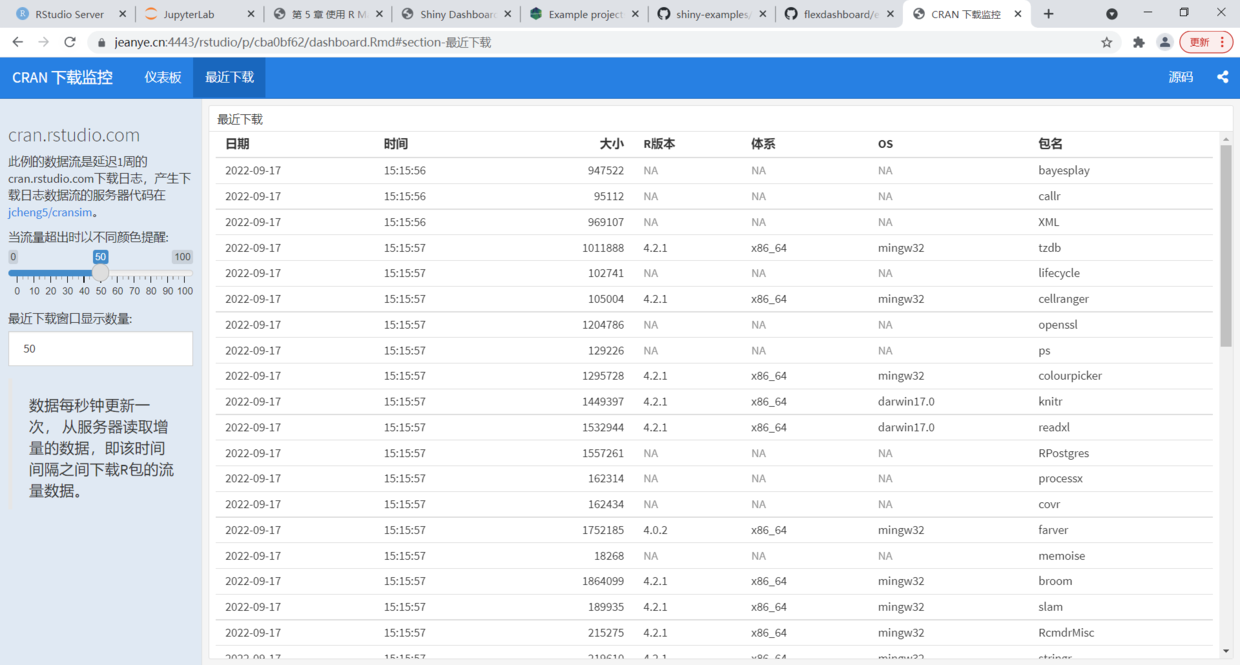
  更多的flexdashboard例子请看[这里](https://pkgs.rstudio.com/flexdashboard/articles/examples.html)。

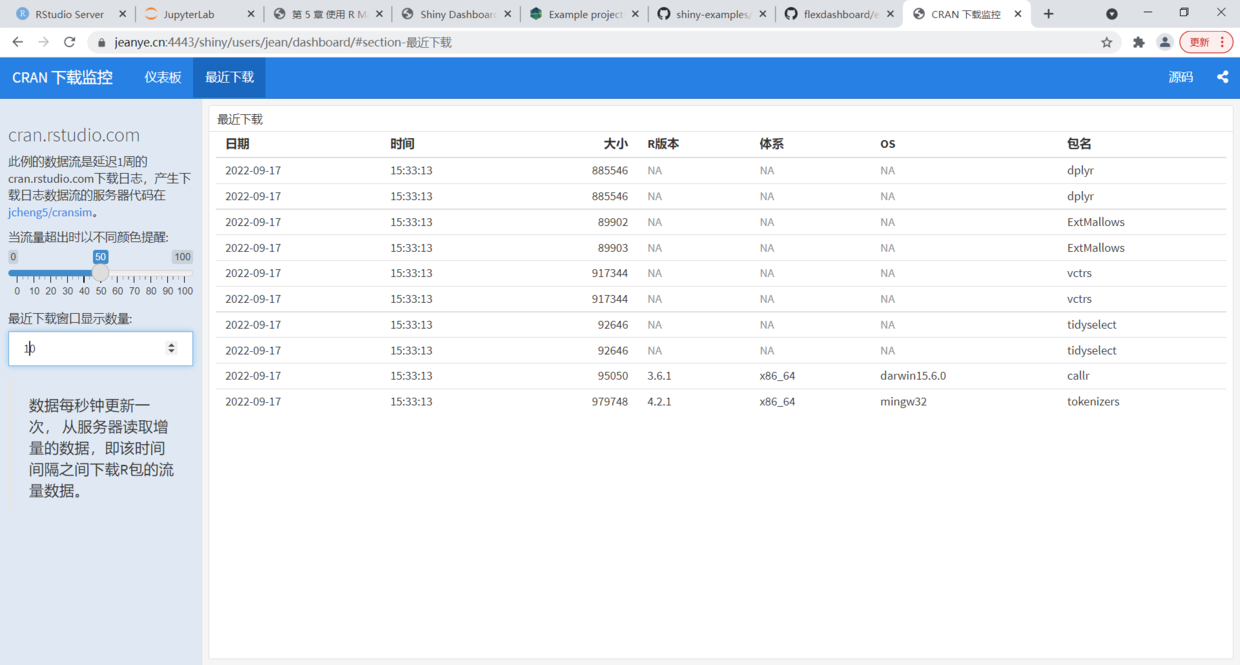
一、先看看运行效果，界面已经汉化了。

  这个仪表板有两个维度的动态，一是数据每秒更新一次，二是调整左边的两个反应式输入变量，会动态的改变右边仪表板的显示。

1、开始的时候，下载流量小于50次/秒时，流量表的颜色是绿色的。

2、调整左边滑杆的值，大于30次/秒就提醒，流量表的颜色变成了橙色。 

3、最近下载列表显示的行数，由左边的一个反应式数字输入来调整。 这是调整前的，显示50行。

这是调整后的，显示10行。

二、flexdashboard的仪表板组件。

  这个例子很简单，但已经具备了典型仪表板的所有要素。一个flexdashboard仪表板可以有7种组件，如上所见。

1、基于 HTML 小部件的交互式 JavaScript 数据可视化图形。上面的泡泡图就是一个htmlwidget [bubbles](https://github.com/jcheng5/bubbles)，生成了一个单旋臂星系图，更多的htmlwidget可以看 [这里](https://gallery.htmlwidgets.org/)，提供了各种各样丰富的可视化工具。

2、R 图形，包括基础、栅栏和网格图形。上面例子中没有，R markdown代码块中R绘图的输出，前面的系列文章中很多了。

3、表格，如上面的下载百分比列表与最近下载列表（本例中是Shiny渲染输出）。

  前面的3种组件是R markdown文档中通用的，后面的4中组件则是flexdashboard独有的。

4、数值框（展示重要数据），如上图中顶端蓝色的下载总数与下载的用户数两个指标。

5、仪表盘，如上图中的流量表。

6、文本注释，如上图中“数据每秒更新一次”的说明，它有特定的语法，用”>“开头。

7、导航栏（提供与仪表板相关的更多链接），如上图中最右上角的“源码”链接。

  左边的反应式输入变量，滑杆与数字，是Shiny的反应式组件，它们不是flexdashboard的组件。flexdashboard仪表板可以是一个动态的Shiny R markdown文档，如本例所见，这提供了更好的交互性和动态。

三、仪表板源码

  源码有2个文件，一个仪表板R markdown文件和一个从服务器读取数据的R函数文件。

1、dashboard.Rmd

  先看Rmd文件的YAML头，输出类型是flexdashboard::flex\_dashboard，实质上是HTML页面。然后设置了它的样式主题是cosmo，宇宙。flexdashboard内置的主题有“default”, “bootstrap”, “cerulean”, “cosmo”, “darkly”, “flatly”, “journal”, “lumen”, “paper”, “readable”, “sandstone”, “simplex”, “spacelab”, “united”, “yeti”，如果想用其他更多的样式主题，可以到[Bootswatch](https://bootswatch.com/)上看看，上面各主题的具体颜色配置，也可以到该站上看看。比如要设置成spacelab主题，用下面代码块中注释的部分代替即可。有关flexdashboard样式主题的设置，可以参阅[这篇文章](https://rstudio.github.io/flexdashboard/articles/theme.html)。这些样式的[源码在这里](https://github.com/rstudio/flexdashboard/tree/feature/logo-and-favicon/inst/rmarkdown/templates/flex_dashboard/resources)，也可以参考定义自己的样式。

  上面的第7种flexdashboard组件，导航栏，在YAML头中用navbar定义。

  它的runtime是shiny，这是一个交互式的Shiny R markdown文档，要部署在Shiny Server上，当然笔记本上单机运行也可以。

---  
title: "CRAN 下载监控"  
output:   
 flexdashboard::flex\_dashboard:  
 theme: cosmo  
 # theme:   
 # version: 4  
 # bootswatch: spacelab  
 orientation: rows  
 social: menu  
 navbar:  
 - { title: "源码", href: "https://github.com/rstudio/flexdashboard/tree/main/examples/04\_shiny-cran-downloads" }  
runtime: shiny  
---

```{r setup, include=FALSE}  
library(flexdashboard)  
  
# 这个工具类R函数源码从 CRAN上读取R包下载日志的增量数据流，后面再介绍。  
source("helpers.R")  
  
# pkgStream是一个反应表达式，它代表了增量的R包下载日志数据流。  
# 每秒更新一次并返回上次更新后的增量下载数据data frame。  
# 通过invalidateLater()函数让反应表达式1秒后自动失效来自动从服务器更新数据。  
# 参阅 https://mastering-shiny.org/reactivity-objects.html#timed-invalidation-adv  
pkgStream <- packageStream()  
  
# pkgData 也是一个反应表达式，它累积了之前所有pkgStream返回的数据，  
# 然后抛弃了所有超过maxAgeSecs秒的数据，这里包含的是5分钟内的数据。  
# 有需要的话，也可以把该参数变成反应式变量，动态调整。  
maxAgeSecs <- 60 \* 5   
pkgData <- packageData(pkgStream, maxAgeSecs)  
```

  Sidebar {.sidebar}是兼容[shinydashboard](https://rstudio.github.io/shinydashboard/)的写法，表示下面的Shiny反应式输入组件安排在左边，下面一排等号是分页符，相当于一级标题”#“，要把Sidebar作为一个单独的框架页面。具体markdown语法请参阅[《Markdown syntax》](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html)一节。这里在R代码块中定义了两个Shiny反应式输入组件sliderInput()与numericInput()。

  仪表板中的每个组件都可以包括标题和注释部分。三级标题 “###” 后面的文本为标题；“>” 开头的文本是注释。

Sidebar {.sidebar}  
=======================================================================  
  
### cran.rstudio.com  
  
此例的数据流是延迟1周的cran.rstudio.com下载日志，产生下载日志数据流的服务器代码在[jcheng5/cransim](https://github.com/jcheng5/cransim)。  
  
  
```{r}  
# 下载高流量颜色阀值  
sliderInput("rateThreshold", "当流量超出时以不同颜色提醒:",  
 min = 0, max = 100, value = 50, step = 1  
)  
  
# 最近下载窗口显示数量  
numericInput("maxrows", "最近下载窗口显示数量:", 50)  
```  
  
> 数据每秒钟更新一次， 从服务器读取增量的数据，即该时间间隔之间下载R包的流量数据。

  仪表板这一页中，Row下面一行减号相当于二级标题”##Row”，用于在仪表板的布局中分行，它是个网格结构，具体布局可以参阅[《Layout》](https://bookdown.org/yihui/rmarkdown/layout.html)一节，这里源码中用等号行分页减号行分行是为了在源码中便于阅读。在flexdashboard中，一级标题会作为整个仪表板的标题显示，在本例中就是顶部“仪表板”Tab；三级标题会作为仪表板组件的标题显示，二级标题是布局标题，它们不会显示。这一行的网格中会安放流量表、总下载数、下载用户数3个组件。

仪表板  
====================================================================  
  
Row  
--------------------------------------------------------------------  
  
### 下载数/秒 (过去 5 分钟)   
  
```{r}  
# downloadRate 是一个计算仪表板运行期间下载流量的反应表达式  
# 记住pkgData()存放的是5分钟内R包下载日志的数据。  
startTime <- as.numeric(Sys.time())  
downloadRate <- reactive({  
 elapsed <- as.numeric(Sys.time()) - startTime  
 nrow(pkgData()) / min(maxAgeSecs, elapsed)  
})  
  
# 输出下载流量指标。因为是在Shiny反应式编程环境中，要用render()函数封装渲染。  
# gauge()是flexdashboard的仪表htmlwidget，根据上面的反应式变量rateThreshold设置仪表的颜色。  
renderGauge({  
 rate <- formatC(downloadRate(), digits = 1, format = "f")  
 gauge(rate, min = 0, max = 100, symbol="次", gaugeSectors(  
 #success = c(0, 33), warning = c(34, 66), danger = c(77, 100)  
 success = c(0, input$rateThreshold), warning = c(input$rateThreshold, 100)  
 ))  
})  
```

### 总下载数 {.value-box}  
  
```{r}  
# dlCount 是一个反应表达式，  
# 记录了从 pkgStream收到的所有数据行数，跨越了5分钟的时间窗口。  
dlCount <- downloadCount(pkgStream)  
  
# 输出总下载数   
renderValueBox({  
 valueBox(dlCount(), icon = "fa-download")  
})  
```

### 下载的用户数 {.value-box}  
  
```{r}  
# usrCount 是一个反应表达式，  
# 记录了仪表板运行期间下载过R包的单个用户计数。  
usrCount <- userCount(pkgStream)  
  
# 输出下载的用户数   
renderValueBox({  
 valueBox(value = usrCount(), icon = "fa-users")  
})  
```

  这一行安放2个flexdashboard组件，过去5分钟内下载最多的R包，它们所占百分比的列表和泡泡图。这是render()泡泡图，记住，反应式编程环境要用render()函数渲染输出。

Row  
--------------------------------------------------------------------  
  
### 下载最多的R包 (过去 5 分钟) {data-width=700}  
  
```{r}  
# 泡泡图HTML widget bubbles, https://github.com/jcheng5/bubbles。  
renderBubbles({  
 if (nrow(pkgData()) == 0)  
 return()  
  
 order <- unique(pkgData()$package)  
 df <- pkgData() %>%  
 group\_by(package) %>%  
 tally() %>%  
 arrange(desc(n), tolower(package)) %>%  
 # 只显示前60，否则可视化效果不好。  
 head(60)  
  
 bubbles(df$n, df$package, key = df$package, color = rainbow(60, alpha=NULL)[sample(60)])  
})  
```

### 下载百分比 (过去 5 分钟) {data-width=340}  
  
```{r}  
renderTable({  
 df <- pkgData() %>%  
 group\_by(package) %>%  
 tally() %>%  
 arrange(desc(n), tolower(package)) %>%  
 mutate(percentage = n / nrow(pkgData()) \* 100) %>%  
 select("Package" = package, "Percent" = percentage) %>%  
 as.data.frame() %>%  
 head(30)  
 # 列名改为中文，只显示前30。  
 names(df)<- c("R包","百分比")  
 df  
}, digits = 1)  
  
```

  等号行分页，这一页只有一个Shiny输出组件最近下载列表，不需要布局代码，所以没有二级标题。可以在仪表板顶端按“最近下载”Tab切换。

最近下载  
====================================================================  
  
### 最近下载  
  
```{r}  
renderTable({  
 downloads <- tail(pkgData(), n = input$maxrows)  
 downloads <- downloads[,c("date", "time", "size", "r\_version",   
 "r\_arch", "r\_os", "package")]  
 downloads[order(nrow(downloads):1),]  
 # 列名改为中文。  
 names(downloads)<-c("日期", "时间", "大小", "R版本",   
 "体系", "OS", "包名")  
})  
```

2、helpers.R

  用到的两个包shinySignals、bubbles需要从Github安装。

library(shiny)  
# devtools::install\_github("hadley/shinySignals")  
library(shinySignals)  
library(dplyr)  
# devtools::install\_github("jcheng5/bubbles")  
library(bubbles)  
  
# 这是一个空的 data frame 原型，用于存放服务器返回的下载日志数据。  
prototype <- data.frame(date = character(), time = character(),  
 size = numeric(), r\_version = character(), r\_arch = character(),  
 r\_os = character(), package = character(), version = character(),  
 country = character(), ip\_id = character(), received = numeric())

  packageStream()连接服务器读取cran.rstudio.com的下载日志stream，返回一个data frame stream。返回的是一个反应表达式，通过Shiny invalidateLater()机制每1000毫秒更新一次。产生下载日志数据流的服务器代码在[jcheng5/cransim](https://github.com/jcheng5/cransim)，用go语言编写，扫描下载日志归档文件并返回相应的增量数据。

packageStream <- function(session = getDefaultReactiveDomain()) {  
 # Connect to data source  
 sock <- socketConnection("cransim.rstudio.com", 6789, blocking = FALSE, open = "r")  
 # Clean up when session is over  
 session$onSessionEnded(function() {  
 close(sock)  
 })  
   
 # 通过一个定时失效的反应表达式得到日志中新的行。  
 newLines <- reactive({  
 invalidateLater(1000, session)  
 readLines(sock)  
 })  
   
 # 将日志行数据转换为data frame，并以反应表达式返回。  
 reactive({  
 if (length(newLines()) == 0)  
 return()  
 read.csv(textConnection(newLines()), header=FALSE, stringsAsFactors=FALSE,  
 col.names = names(prototype)  
 ) %>% mutate(received = as.numeric(Sys.time()))  
 })  
}

  把反应表达式pkgStream返回的行，累积起来，然后按时间窗口（5分钟）过滤。使用了上面packageStream()为数据添加的received时间标签。可以用?shinySignals::reducePast看该函数的文档，这应该是最难理解的一个函数了，会用即可。[shinySignals](https://github.com/hadley/shinySignals)包提供了Shiny反应机制反应讯号处理的扩展工具，文档见其Github项目。

packageData <- function(pkgStream, timeWindow) {  
 shinySignals::reducePast(pkgStream, function(memo, value) {  
 rbind(memo, value) %>%  
 filter(received > as.numeric(Sys.time()) - timeWindow)  
 }, prototype)  
}

  累计反应表达式pkgStream收到的行数。看懂了上面shinySignals::reducePast()的用法，这里也就看懂了。

downloadCount <- function(pkgStream) {  
 shinySignals::reducePast(pkgStream, function(memo, df) {  
 if (is.null(df))  
 return(memo)  
 memo + nrow(df)  
 }, 0)  
}

  累计的单独用户计数。这个算法有点复杂，本篇的重点不在根据IP和日期确定跨日的两条下载记录是否为同一个用户，可以先不管它，就当是个测试数据好了，有时间再仔细分析它的算法，所以把注解原文留下。

# Use a bloom filter to probabilistically track the number of unique  
# users we have seen; using bloom filter means we will not have a  
# perfectly accurate count, but the memory usage will be bounded.  
userCount <- function(pkgStream) {  
 # These parameters estimate that with 5000 unique users added to  
 # the filter, we'll have a 1% chance of false positive on the next  
 # user to be queried.  
 bloomFilter <- BloomFilter$new(5000, 0.01)  
 total <- 0  
 reactive({  
 df <- pkgStream()  
 if (!is.null(df) && nrow(df) > 0) {  
 # ip\_id is only unique on a per-day basis. To make them unique  
 # across days, include the date. And call unique() to make sure  
 # we don't double-count dupes in the current data frame.  
 ids <- paste(df$date, df$ip\_id) %>% unique()  
 # Get indices of IDs we haven't seen before  
 newIds <- !sapply(ids, bloomFilter$has)  
 # Add the count of new IDs  
 total <<- total + length(newIds)  
 # Add the new IDs so we know for next time  
 sapply(ids[newIds], bloomFilter$set)  
 }  
 total  
 })  
}  
  
# Quick and dirty bloom filter. The hashing "functions" are based on choosing  
# random sets of bytes out of a single MD5 hash. Seems to work well for normal  
# values, but has not been extensively tested for weird situations like very  
# small n or very large p.  
  
library(digest)  
library(bit)  
  
BloomFilter <- setRefClass("BloomFilter",  
 fields = list(  
 .m = "integer",  
 .bits = "ANY",  
 .k = "integer",  
 .bytesNeeded = "integer",  
 .bytesToTake = "matrix"  
 ),  
 methods = list(  
 # @param n - Set size  
 # @param p - Desired false positive probability (e.g. 0.01 for 1%)  
 initialize = function(n = 10000, p = 0.001) {  
 m = (as.numeric(n) \* log(1 / p)) / (log(2)^2)  
   
 .m <<- as.integer(m)  
 .bits <<- bit(.m)  
 .k <<- max(1L, as.integer(round((as.numeric(.m)/n) \* log(2))))  
   
 # This is how many \*bytes\* of data we need for \*each\* of the k indices we need to  
 # generate  
 .bytesNeeded <<- as.integer(ceiling(log2(.m) / 8))  
 .bytesToTake <<- sapply(rep\_len(.bytesNeeded, .k), function(byteCount) {  
 # 16 is number of bytes an md5 hash has  
 sample.int(16, byteCount, replace = FALSE)  
 })  
 },  
 .hash = function(x) {  
 hash <- digest(x, "md5", serialize = FALSE, raw = TRUE)  
 sapply(1:.k, function(i) {  
 val <- rawToInt(hash[.bytesToTake[,i]])  
 # Scale down to fit into the desired range  
 as.integer(val \* (as.numeric(.m) / 2^(.bytesNeeded\*8)))  
 })  
 },  
 has = function(x) {  
 all(.bits[.hash(x)])  
 },  
 set = function(x) {  
 .bits[.hash(x)] <<- TRUE  
 }  
 )  
)  
  
rawToInt <- function(bytes) {  
 Reduce(function(left, right) {  
 bitwShiftL(left, 8) + right  
 }, as.integer(bytes), 0L)  
}