Data Table,Dplyr,Manipulate,Shiny

Ryan Zhang

November 5, 2015

Data Table

```
library(data.table)
```

- Fast aggregation of large data
- Fast add/modify/delete of columns by group using no copies at all
- Fast file reader (fread).

Data table

- Example data
- Thanks, thanks, thanks Monica !

```
emp <- fread("emp_t.csv")
dept <- fread("dep_t.csv")
product <- fread("product_t.csv")
price <- fread("price_t.csv")
order <- fread("order_t.csv")
orderline <- fread("orderline_t.csv")
customer <- fread("customer_t.csv")
```

- If a data.table/data.frame object is been called by a function that only knows data.frame, it will be passed as a data.frame

```
class(emp)
```

```
## [1] "data.table" "data.frame"
```

- Basic data.table syntax

```
DT [i,
.(j1 = col1, j2 = col2, j3 = func1(col3)),
by = .(col4, col5)]
```

Data Table

- Provide a shortcut `.N` for number of rows in data table

```
emp[1:(nrow(emp)/4)]
```

```
##    EMPNO EFNAME EMIDDLE_INIT ELNAME  EMP_JOB  MGR  HIREDATE  SAL COMM
## 1:  7369    Bob             R  Smith    CLERK 7902 17-DEC-12  800   NA
## 2:  7499    Tim             K  Allen SALESMAN 7698 12-FEB-13 1600  300
## 3:  7654    Sam             L Martin SALESMAN 7698 28-SEP-13 1250 1400
##    DEPTNO
## 1:     20
## 2:     20
## 3:     20
```

```
emp[1:(.N/4)]
```

```
##    EMPNO EFNAME EMIDDLE_INIT ELNAME  EMP_JOB  MGR  HIREDATE  SAL COMM
## 1:  7369    Bob             R  Smith    CLERK 7902 17-DEC-12  800   NA
## 2:  7499    Tim             K  Allen SALESMAN 7698 12-FEB-13 1600  300
## 3:  7654    Sam             L Martin SALESMAN 7698 28-SEP-13 1250 1400
##    DEPTNO
## 1:     20
## 2:     20
## 3:     20
```

- Subset by row, similar to the following SQL query:

```
SELECT *
FROM emp
WHERE MGR = 7698;
```

```
emp[MGR == 7698]
```

```
##    EMPNO  EFNAME EMIDDLE_INIT ELNAME  EMP_JOB  MGR  HIREDATE  SAL COMM
## 1:  7499     Tim             K  Allen SALESMAN 7698 12-FEB-13 1600  300
## 2:  7654     Sam             L Martin SALESMAN 7698 28-SEP-13 1250 1400
## 3:  7521     Kim                Ward SALESMAN 7698 22-FEB-13 1250  500
## 4:  7844    Jose             T Turner SALESMAN 7698 08-SEP-13 1500    0
## 5:  7900 Lakshmi             F Manchu    CLERK 7698 03-DEC-13 1800   NA
##    DEPTNO
## 1:     20
## 2:     20
## 3:     30
## 4:     30
## 5:     30
```

- Subset by row, and specify columns wanted:

```
SELECT EMPNO, EFNAME, ELNAME, EMP_JOB
FROM emp
WHERE MGR = 7698;
```

```
emp[MGR == 7698,
    .(EMPNO, EFNAME, ELNAME, EMP_JOB)]
```

```
##    EMPNO  EFNAME ELNAME  EMP_JOB
## 1:  7499     Tim  Allen SALESMAN
## 2:  7654     Sam Martin SALESMAN
## 3:  7521     Kim   Ward SALESMAN
## 4:  7844    Jose Turner SALESMAN
## 5:  7900 Lakshmi Manchu    CLERK
```

- Provide a method to apply function on columns

```
c(min(emp$SAL), mean(emp$SAL))
```

```
## [1]  800.000 2019.643
```

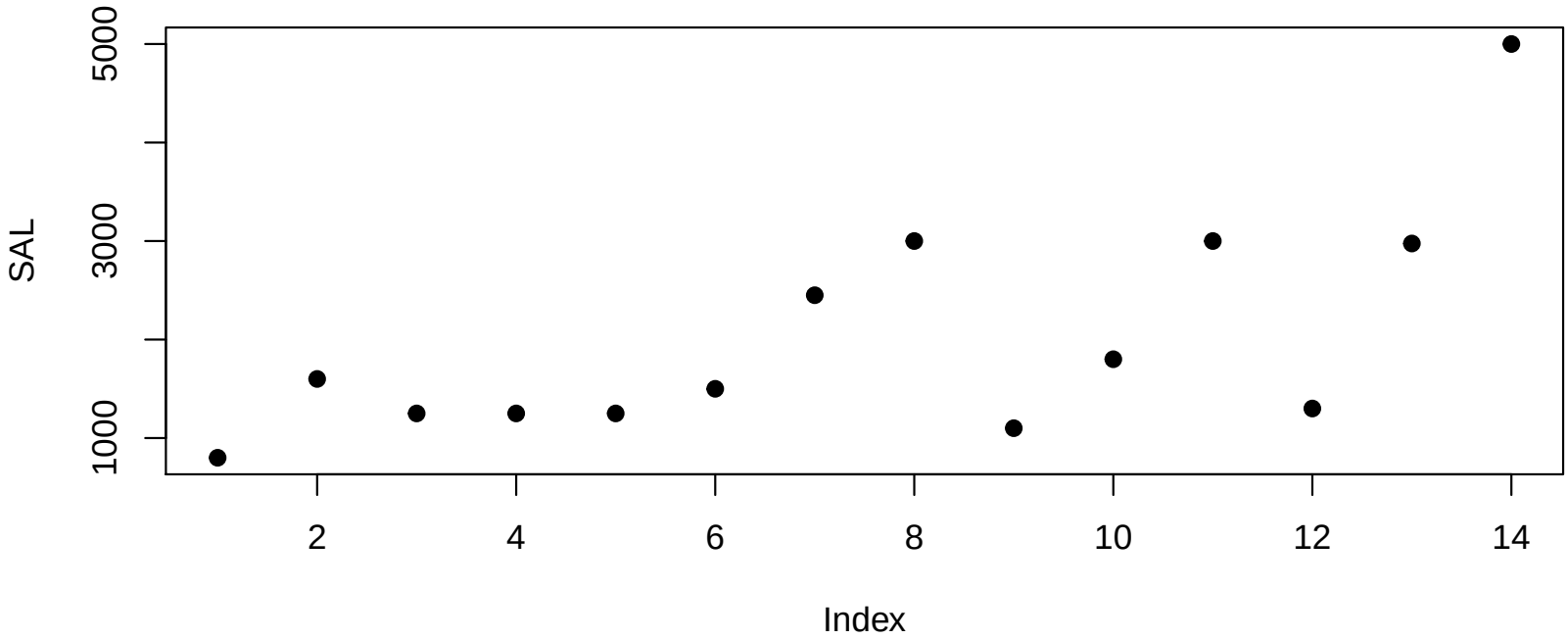
```
emp[,
    .(MinSalary = min(SAL), AvgSalary = mean(SAL))]
```

```
##    MinSalary AvgSalary
## 1:       800  2019.643
```

Data Table

- More than math function can be used on columns in data table

```
emp[,
    .(plot(SAL, pch = 19))]
```




```
## Empty data.table (0 rows) of 1 col: V1
```

- Group by in data table
- Getting more similar to SQL now...

```
SELECT DEPTNO, AVG(SAL) AS "AvgSalary"
FROM emp
GROUP BY DEPTNO;
```

```
emp[,
    .(AvgSalary = mean(SAL)),
    by = .(DEPTNO)]
```

```
##    DEPTNO AvgSalary
## 1:     20  1960.714
## 2:     30  1450.000
## 3:     10  2916.667
```

- More powerful group by than SQL
- Compare the mean Salary of SALESMAN and NON-SALESMAN
- Need two SQL queries or sub queries to get the result

```
emp[,
    .(AvgSalary = mean(SAL)),
    by = .(EMP_JOB == "SALESMAN")]
```

```
##    EMP_JOB AvgSalary
## 1:   FALSE  2380.556
## 2:    TRUE  1370.000
```

- Compare the mean Salary of SALESMAN and NON-SALESMAN within DEPTNO == 20
- Think about how to do it in SQL for a minute...

```
emp[DEPTNO == 20,
    .(AvgSalary = mean(SAL)),
    by = .(EMP_JOB == "SALESMAN")]
```

```
##    EMP_JOB AvgSalary
## 1:   FALSE      2175
## 2:    TRUE      1425
```

- Find the lowest paid employees name
- You need nested Select in SQL, but much simpler in data.table

```
SELECT EFNAME, ELNAME
FROM emp
WHERE SAL = (SELECT MIN(SAL) FROM emp);
```

```
emp[SAL == min(SAL),
    .(EFNAME, ELNAME)]
```

```
##    EFNAME ELNAME
## 1:    Bob  Smith
```

- Remember: what's been returned is a data table
- Find all orders that have more than 3 orderlines on them. List out the order id

```
SELECT orderid
FROM orderline
GROUP BY orderid
HAVING count(*) > 3;
```

```
orderline[,
          .(number = .N),
          by = .(ORDERID)][number >3,
                           .(ORDERID)]
```

```
##    ORDERID
## 1:       4
```

- sqldf allow you use SQLite syntax to query dataframes

```
library(sqldf)
sqldf("SELECT EFNAME, EMIDDLE_INIT, ELNAME, emp.DEPTNO
       FROM dept, emp
       WHERE dept.DEPTNO = emp.DEPTNO
            AND SAL > (SELECT MAX(SAL)
                       FROM dept, emp
                       WHERE dept.DEPTNO = emp.DEPTNO AND emp.DEPTNO = 20);")
```

```
##    EFNAME EMIDDLE_INIT ELNAME DEPTNO
## 1 Korinna             I  Grant     10
```

R can work ontop of many DBMS

```
install.packages("RPostgreSQL")
install.packages("RMySQL")
install.packages("RMongo")
install.packages("RODBC")
install.packages("RSQLite")
```

- They said there is also a **ROracle** library, but I have trouble installing it.

- They said, you can join tables using data.table, which I never figure out how to do....
- Maybe you can find out and teach me how to do it.
- If I need to join tables, I will use dplyr.
- So... Let's talk about dplyr now.

```
library(dplyr)
```

- Fast data bla bla bla
- Can work on top of database
- Very descriptive syntax, like SQL
  - “tell me what you want”
  - “Not necessary how I should find it for you”

- Verbs
  1. filter()
  2. slice()
  3. arrange()
  4. select()
  5. distinct()
  6. transform()
  7. summarise()
  8. group_by()

- filter() allows you to select a subset of rows by a set of conditions.

```
SELECT PRODUCTFINISH, PRODUCTSTANDARDPRICE
FROM product
WHERE PRODUCTFINISH = "Cherry" AND PRODUCTSTANDARDPRICE > 200;
```

```
filter(product, PRODUCTFINISH == "Cherry", PRODUCTSTANDARDPRICE > 200)
```

```
##    PRODUCTID   PRODUCTDESCRIPTION PRODUCTFINISH PRODUCTSTANDARDPRICE
## 1:         4 Entertainment Center        Cherry                 1650
##    PRODUCTONHAND
## 1:             0
```

- slice() allows you the select a subset of rows by positions:
- How to do this in SQL?
- LIMIT and OFFSET are supported by PostgreSQL, but not Oracle? (If my memory is correct.)

```
slice(orderline, 1:6)
```

```
##    ORDERLINEID ORDERID PRODUCTID ORDEREDQUANTITY
## 1:           1       1         2              18
## 2:          31       1         6               2
## 3:           2       1        10               9
## 4:           3       2         3              12
## 5:           4       2         8               2
## 6:           5       2        14               2
```

- arrange() reorders rows according to values in specified columns.
- Similar to ORDER BY in SQL.

```
SELECT *
FROM product
ORDER BY PRODUCTFINISH, PRODUCTSTANDARDPRICE;
```

```
arrange(product, PRODUCTFINISH, PRODUCTSTANDARDPRICE)
```

```
##     PRODUCTID      PRODUCTDESCRIPTION PRODUCTFINISH PRODUCTSTANDARDPRICE
##  1:         2     Birch Coffee Tables         Birch                  200
##  2:        14          Writer's Desk         Birch                  300
##  3:         6       8-Drawer Dresser         Birch                  750
##  4:        13             Nightstand        Cherry                  150
##  5:         1       Cherry End Table        Cherry                  175
##  6:         4   Entertainment Center        Cherry                 1650
##  7:        17 High Back Leather Chair       Leather                  362
##  8:         8            48 Bookcase          Oak                  175
##  9:        10            96 Bookcase          Oak                  200
## 10:         5          Writer's Desk          Oak                  325
## 11:        11       4-Drawer Dresser          Oak                  500
## 12:         3      Oak Computer Desk          Oak                  750
## 13:        12       8-Drawer Dresser          Oak                  800
## 14:        18   6' Grandfather Clock          Oak                  890
## 15:        19   7' Grandfather Clock          Oak                 1100
## 16:        21          Pine End Table         Pine                  256
## 17:         7            48 Bookcase       Walnut                  150
```

- Use desc() to order a column in descending order.

```
arrange(product, PRODUCTFINISH, desc(PRODUCTSTANDARDPRICE))
```

```
##     PRODUCTID      PRODUCTDESCRIPTION PRODUCTFINISH PRODUCTSTANDARDPRICE
##  1:         6        8-Drawer Dresser         Birch                  750
##  2:        14           Writer's Desk         Birch                  300
##  3:         2     Birch Coffee Tables         Birch                  200
##  4:         4    Entertainment Center        Cherry                 1650
##  5:         1        Cherry End Table        Cherry                  175
##  6:        13              Nightstand        Cherry                  150
##  7:        17 High Back Leather Chair       Leather                  362
##  8:        19    7' Grandfather Clock           Oak                 1100
##  9:        18    6' Grandfather Clock           Oak                  890
## 10:        12        8-Drawer Dresser           Oak                  800
## 11:         3       Oak Computer Desk           Oak                  750
## 12:        11        4-Drawer Dresser           Oak                  500
## 13:         5           Writer's Desk           Oak                  325
## 14:        10             96 Bookcase           Oak                  200
## 15:         8             48 Bookcase           Oak                  175
## 16:        21          Pine End Table          Pine                  256
## 17:        20                  Amoire        Walnut                 1200
## 18:         9             96 Bookcase        Walnut                  225
## 19:         7             48 Bookcase        Walnut                  150
##     PRODUCTONHAND
##  1:             0
```

- select() allows you to select columns:

```
SELECT STDPRICE, MINPRICE
FROM price;
```

```
select(price, STDPRICE, MINPRICE)
```

```
##    STDPRICE MINPRICE
## 1:      150      100
## 2:      200      150
```

- distinct() to find unique values in a table.
- Remind you of the DISTINCT in SQL?

```
SELECT DISTINCT PRODUCTFINISH
FROM product;
```

```
distinct(select(product, PRODUCTFINISH))
```

```
##    PRODUCTFINISH
## 1:        Cherry
## 2:         Birch
## 3:           Oak
## 4:        Walnut
## 5:       Leather
## 6:          Pine
```

- Add new columns with mutate(), you can directly work on the columns you just created.
- Think about how to do this in SQL.

```
mutate(product2,
       DISCOUNT = ifelse(PRODUCTFINISH == "Leather", 0.85, 0.7),
       DISCOUNTEDPRICE = DISCOUNT * PRODUCTSTANDARDPRICE)
```

```
## Source: local data frame [19 x 7]
##
##    PRODUCTID      PRODUCTDESCRIPTION PRODUCTFINISH PRODUCTSTANDARDPRICE
##        (int)                   (chr)         (chr)                (int)
## 1          1       Cherry End Table        Cherry                  175
## 2          2    Birch Coffee Tables         Birch                  200
## 3          3      Oak Computer Desk           Oak                  750
## 4          4   Entertainment Center        Cherry                 1650
## 5          5          Writer's Desk           Oak                  325
## 6          6      8-Drawer Dresser         Birch                  750
## 7          7            48 Bookcase        Walnut                  150
## 8          8            48 Bookcase           Oak                  175
## 9          9            96 Bookcase        Walnut                  225
## 10        10            96 Bookcase           Oak                  200
## 11        11      4-Drawer Dresser           Oak                  500
## 12        12      8-Drawer Dresser           Oak                  800
## 13        13             Nightstand        Cherry                  150
## 14        14          Writer's Desk         Birch                  300
```

- use transmute() is only include the columns created

```
transmute(product2,
          DISCOUNT = ifelse(PRODUCTFINISH == "Leather", 0.85, 0.7),
          DISCOUNTEDPRICE = DISCOUNT * PRODUCTSTANDARDPRICE)
```

```
## Source: local data frame [19 x 2]
##
##    DISCOUNT DISCOUNTEDPRICE
##       (dbl)           (dbl)
## 1      0.70           122.5
## 2      0.70           140.0
## 3      0.70           525.0
## 4      0.70          1155.0
## 5      0.70           227.5
## 6      0.70           525.0
## 7      0.70           105.0
## 8      0.70           122.5
## 9      0.70           157.5
## 10     0.70           140.0
## 11     0.70           350.0
## 12     0.70           560.0
## 13     0.70           105.0
## 14     0.70           210.0
## 15     0.85           307.7
## 16     0.70           623.0
```

- use summarise() to do aggregation

```
SELECT AVG(PRODUCTSTANDARDPRICE)
FROM product;
```

```
summarise(product,
  meanPrice = mean(PRODUCTSTANDARDPRICE))
```

```
##    meanPrice
## 1:  534.6316
```

- use group_by() to do grouping.

```
SELECT AVG(PRODUCTSTANDARDPRICE)
FROM product
GROUP BY PRODUCTFINISH;
```

```
product_by_FINISH <- group_by(product, PRODUCTFINISH)
summarise(product_by_FINISH,
          meanPrice = mean(PRODUCTSTANDARDPRICE))
```

```
## Source: local data table [6 x 2]
##
##   PRODUCTFINISH meanPrice
##           (chr)     (dbl)
## 1        Cherry  658.3333
## 2         Birch  416.6667
## 3           Oak  592.5000
## 4        Walnut  525.0000
## 5       Leather  362.0000
## 6          Pine  256.0000
```

- Chaining.

```
SELECT AVG(PRODUCTSTANDARDPRICE) AS "meanPrice"
FROM product
GROUP BY PRODUCTFINISH
HAVING AVG(PRODUCTSTANDARDPRICE) > 300
ORDER BY AVG(PRODUCTSTANDARDPRICE) DESC;
```

```
arrange(
    filter(
        summarise(group_by(product,
                           PRODUCTFINISH),
                  meanPrice = mean(PRODUCTSTANDARDPRICE)),
        meanPrice > 300),
    desc(meanPrice))
```

```
## Source: local data table [5 x 2]
##
##   PRODUCTFINISH meanPrice
##           (chr)     (dbl)
## 1        Cherry  658.3333
## 2           Oak  592.5000
## 3        Walnut  525.0000
## 4         Birch  416.6667
## 5       Leather  362.0000
```

- Use %>% this strange symbol.. To make code more readable.

```
product %>%
    group_by(PRODUCTFINISH) %>%
    summarise(meanPrice = mean(PRODUCTSTANDARDPRICE)) %>%
    filter(meanPrice > 300) %>%
    arrange(desc(meanPrice))
```

```
## Source: local data table [5 x 2]
##
##   PRODUCTFINISH meanPrice
##           (chr)     (dbl)
## 1        Cherry  658.3333
## 2           Oak  592.5000
## 3        Walnut  525.0000
## 4         Birch  416.6667
## 5       Leather  362.0000
```

- Lots of join operations as well...

```
inner_join(emp, dept)
```

```
## Joining by: "DEPTNO"

## Source: local data table [14 x 12]
##
##    DEPTNO EMPNO  EFNAME EMIDDLE_INIT  ELNAME   EMP_JOB   MGR  HIREDATE
##     (int) (chr)   (chr)         (chr)   (chr)     (chr) (chr)     (chr)
## 1      10  7782  Sandra             G   Clark   MANAGER  7839 09-JUN-13
## 2      10  7934    Paul             F  Miller     CLERK  7782 23-JAN-13
## 3      10  7839 Korinna             I   Grant PRESIDENT       08-NOV-13
## 4      20  7369     Bob             R   Smith     CLERK  7902 17-DEC-12
## 5      20  7499     Tim             K   Allen  SALESMAN  7698 12-FEB-13
## 6      20  7654     Sam             L  Martin  SALESMAN  7698 28-SEP-13
## 7      20  7788 Cameron             W   Scott   ANALYST  7566 09-DEC-13
## 8      20  7876      Li             F    Zhou     CLERK  7788 01-DEC-13
## 9      20  7902    Luis             F  Suarez   ANALYST  7566 03-DEC-13
## 10     20  7566    Joan             F    Jett   MANAGER  7839 05-APR-13
## 11     30  7698   Blake             L Shelton  SALESMAN       28-SEP-13
## 12     30  7521     Kim                 Ward  SALESMAN  7698 22-FEB-13
## 13     30  7844    Jose             T  Turner  SALESMAN  7698 08-SEP-13
## 14     30  7900 Lakshmi             F  Manchu     CLERK  7698 03-DEC-13
## Variables not shown: SAL (int), COMM (int), DNAME (chr), LOC (chr)
```

- Find all Products purchased by at least 4 customers.

```
SELECT PRODUCTID
FROM order, orderline
WHERE order.ORDERID = orderline.ORDERID
GROUP BY PRODUCTID, CUSTOMERID
HAVING COUNT(*) >= 4;
```

```
inner_join(order, orderline, by = "ORDERID") %>%
    group_by(PRODUCTID, CUSTOMERID) %>%
    summarise(countCustomer = n()) %>%
    filter(countCustomer >= 4) %>%
    select(PRODUCTID)
```

```
## Source: local data table [1 x 1]
## Groups: PRODUCTID
##
##   PRODUCTID
##       (int)
## 1         1
```

- Find all employees who make more than their manager

```
SELECT EFNAME, EMIDDLE_INIT, ELNAME
FROM emp employee, emp manager
WHERE employee.mgr = manager.empno
AND employee.SAL > manager.SAL;
```

- I don't know how to do self join using dplyr
- Maybe you can dig into it and teach me...

- Interactive plotting functions for use within RStudio.
- The manipulate function accepts a plotting expression and a set of controls (e.g. slider, picker, checkbox, or button) which are used to dynamically change values within the expression.

```
library(manipulate);library(ggplot2)
manipulate(
qplot(x = diamonds[,variable], geom = geomtype, fill = diamonds[, colorby])+
    xlim(c(xmin, xmax)),
variable = picker("carat" = "carat", "depth" = "depth", "table" = "table",
                  "price" = "price", "x" = "x", "y" = "y","z" = "z"),
geomtype = picker("histogram" = "histogram", "freqpoly" = "freqpoly"),
colorby = picker("cut" = "cut", "color" = "color", "clarity" = "clarity"),
xmin = slider(0,300), xmax = slider(5.2,19000))
```

- Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications).
- ui.R
- server.R

```
library(shiny)
runApp()
```

- ui.R

```
library(shiny)
shinyUI(pageWithSidebar(
  headerPanel("Standard Normal Density"),# Application title
  sidebarPanel(# Sidebar with a slider input for
    p("computes probability for a standard normal distribution."),
    p("choose a score by sliding the slide, and choose tail using the dropdown.
    h3('Please choose score'),
    sliderInput("score", "It could only be between -4 and 4:",
                min = -4, max = 4, value = 0, step = 0.01),
    h3('Please choose lower tail or upper tail'),
    selectInput("lo", " ", choices = c("lower", "upper"), selected = "lower")
    ),
  mainPanel(# show a normal distribution with score indicated as a red vertical
    plotOutput("distPlot"),
    h3("The associated probability is"),
    verbatimTextOutput("pvalue"))))
```

- server.R

```
library(shiny)
shinyServer(
  function(input,output){
    output$distPlot <- renderPlot({# generate an rnorm distribution and plot it
      x <- seq(-4,4,by = 0.01)
      y <- dnorm(x,mean=mean(x),sd=sd(x))
      xl <- rep(input$score,301)
      yl <- seq(0,0.3,by = 0.001)
      plot(x, y, type = "l",col="black", lwd=2,ylab = "Pr(X=x)")
      lines(xl,yl,col="red",lwd=2)
    })
    lower <- reactive({switch(input$lo, "lower" = T, "upper" = F)})
    output$pvalue <- renderPrint({pnorm(input$score, 0,1, lower.tail = lower())
    })
```