**R Essentials**

Joyce 学statistics,做Data science

Some R recources

Basic: Computing for data analysis:

https://www.youtube.com/watch?v=EiKxy5IecUw&list=PLycnP7USbo1XGmTeFAAzr79e8V

7zr7Ccx

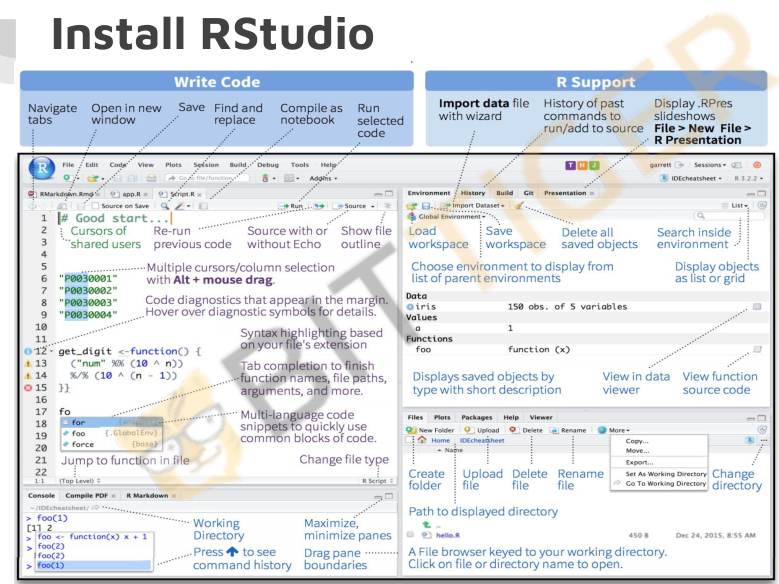
Project:

● R-bloggers: https://www.r-bloggers.com

● Kaggle: https://www.kaggle.com/

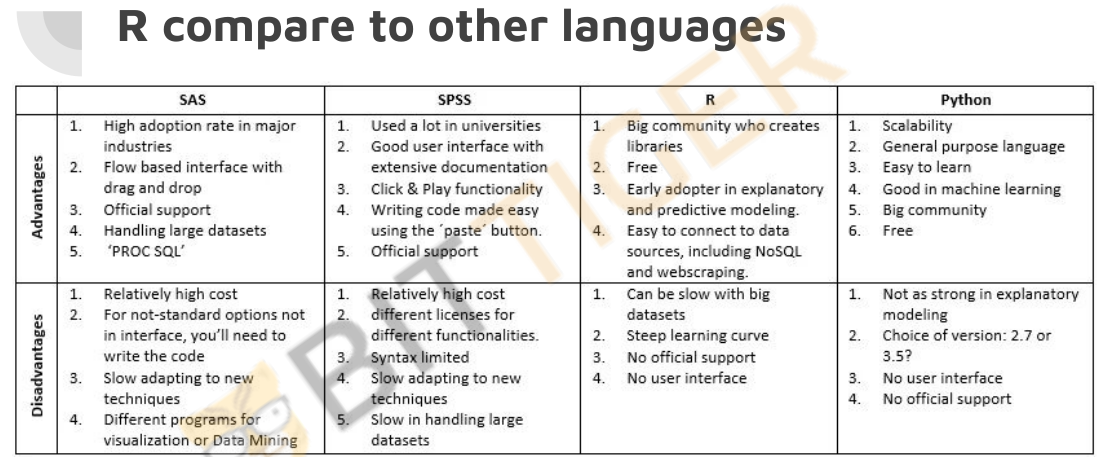
R for machine learning: http://appliedpredictivemodeling.com/

Rsudio是一个编辑界面，类似UI，让你能更方便的使用R。



**R in practice & interview**

同时用R和python的人比较少，一般两个里面选一个就可以了。R比python还简单好学，稍微会点以防兼容问题，（别的同事都用R你也可以用）但不能处理大数据而且bug多因为没人维护。



面试时可能会用到，经常会问你会什么language，会R会python会sas，面试官可能会问你什么时候用R什么时候用python或者会问你觉得r和python的比较是什么样的？比较常见比较casual的问题，不一定有完全正确的答案但是如果说不出来点东西还是会red flag觉得你这两个语言其实都不熟。

**SAS**最大的好处：因为是付钱的 所以有很好的support，所有的function都是经过公司的人反复验证，documentaton都非常清楚，一般在一些对**regulation比较严格的industry**比如**银行，clinical trial-biostets，保险公司**用。银行每年都要向美联储银监会提交报告做年审，在做年审时所有的模型都要解释的非常清楚，哪里是怎么做的，每一个function用的是什么。SAS因为**documentation特别清楚**，所有industry统一用，不太容易被challenge。（如果你用R，用一个自己写的function，很容易被人challenge你的function是不是写错了之类的）如果想进金融行业要会SAS

坏处：licence贵，一年几百上千刀，一般初创小公司并不会买。**金融行业里面进去了以后一定会用SAS。**

SPSS在social science里面用的相对较多，工作中几乎用不到，越来越less popular，它比较像SAS，也要licence，但是没有SAS好用。

R最大的好处就是免费，start-up可能用的多；flexible，可以自己写很多function，比SAS好用很多。

坏处是处理大数据的能力非常有限，当数据规模大的时候必须要换到别的language。R适用于做prototype（原型，雏形，蓝本），industry中在自己电脑上先去做一个试验，develop一个model。但一旦这个model要进production（叫Uber一辆车的背后不停有模型在计算如何降低等待时间、怎么计算价格，怎么match，怎么估计arrival time等，所有这些都是模型在跑，这些都叫production system，因为是实时的）

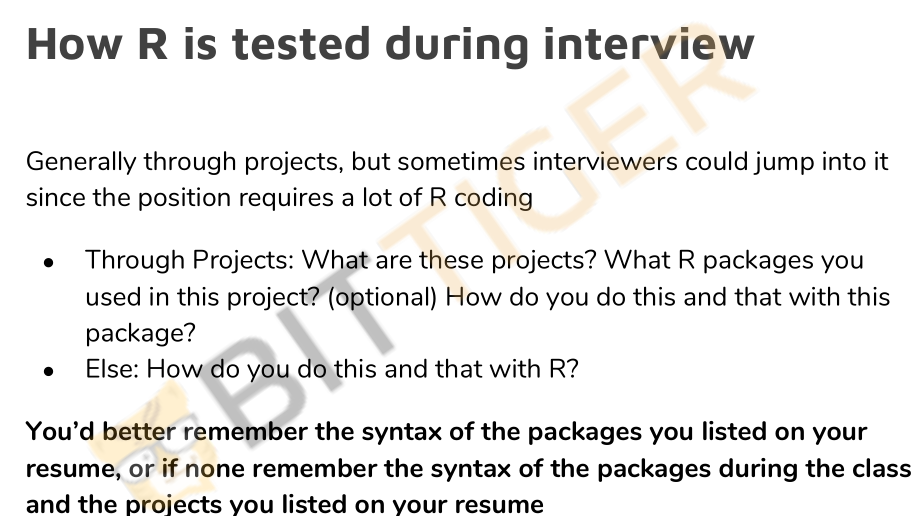
一般prototype都是off-line的，我现在拿上个月所有uber的trips data来做一个模型，这个模型可以看准确度如何，跟实际的data差多少——这些都叫prototype。当我发现我的模型建的差不多了，有信心让它实时去跑的时候就叫“build into production”/“productionalize”

Python-跟R差不多，免费，flexible，对比R最大的好处就是它是一个真正的programming language，它对于production system和production的接口是非常好的，python和C和C++的接口很多，跑的很快，当你要处理real time data和大规模的data时都用python，比如面试时问你在uber中算一个matching 的model在哪里写，不能说在R里写，R中写无法进production。

SAS考证——锦上添花不是雪中送炭，考证的过程作为学习的过程就行。SAS advance考证过分追求syntax细节，考这个的时间做proj学会如何用这个软件作分析可能更有用。

面试时一般的说辞是“我用python很熟，但是我也接触过R的syntax，我觉得他们非常相似。我可以很快pick up”

编程语言在面试中怎么考察？可能会让你提交一个做过的proj；或者做take home proj，交code和report；可能问你一个问题。



Proj真正做过的话有很多细节的东西可以问，package等。

可能会问一句syntax，比如让你写一行code，**如何在r中做t-test**？答案就是T.Test 有时会懵逼，但又说自己对R熟，就。。。有时候会和一些逻辑类的问题结合问，比如K-means（clustering的方法），有时会让你**写一个k-means的方法**，**可以选R可以选python**。

自己真正hands-on的用过的语言和package可以写在简历上，如果没用过只是听说过千万不要写。写上去就容易被问到，被问到不会就很糟糕。如果大概用过一点点，比如只用R做过一个proj可以在proj里写或者“experience in R”

在proj中写清楚用了R中的哪些package或者在R中做了什么事情。

面试之前refresh memory，review一下简历里写过的package里的syntax，可能在面试中被问到。

R on resume

Mentioned the package name if JD requires and you are confident; OR you believe the package is very popular and should mention it。（项目中写清楚在R中做了啥就够了）

高频题（型）

1. R的经验

a. Projects on resume

b. Package on resume (what do you use it for)； 如果问，一般来说因为公司也在用这个package

c. 什么时候用R，什么时候用其他（考查你工具之间切换的能力）

i. SQL处理大量较整齐干净的数据：速度最快

ii. R一般用于（一次性）统计，模型，画图

iii. Python常用于建数据的pipeline

2. R的使用

a. 基本概念：数据类型；常识 ——> 背知识点&刷题！（网上的面试题，常会问到的领域）

b. 数据处理：project过程中痛苦积累来的经验；刷题有点用。（只看别人的code很难提高，要自己做）

**R和Python**和SAS可以只会一个，它们都是做**statistical analysis**,visualization,summary,function等

**SQL**是和它们不同类型的语言，必须会，是用来**快速提取数据，对数据做简单的aggregation**，看下数据什么样等等，并不能visualization，不能做model这种事。对大量数据做简单分析，只是count，mean这种可以用SQL。

R during interview

准备好段子（Project，why project，why this package）

In the end, if you don’t know the syntax of a certain package, it is fine, as long as you could get the things done

**R basic syntax**

**R – Operators**

● Assignment Operator : **<- or =**

x <- 1

print(x)

[1] 1

● Arithmetic Operators: +, -, \*, /

x <- 1

Y <- 2

X + y

1. 3

● Relational Operator : >, <, **==**, <=, >=, **!=** 要比较两个数是否相等要写两个等号 == ~~和python不同~~

x <- 1

print(x >= 3)

[1] False

● Logical Operators: **&（point wised AND，每个element进行AND）**, |（or）, !（非）, **&&**, **||（适用于vector，一列）**

**Printing**

> x <- 1:20

> x

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

[16] 16 17 18 19 20

The : operator is used to create integer sequences.

可以直接把变量打出来来print它或者print(x)

**R Data Types & Structures**

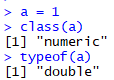
**Atomic data types** ——R has 6 atomic vector types单元素的数据类型有6种

● character字符串: "a", ”hahaha"

● numeric数字，包括整数和浮点数: 2, 15.5

● integer: 2L (**the L tells R to store this as an integer**)

● logical: TRUE, FALSE

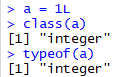
● complex复数: 1+4i (complex numbers with real and imaginary parts

R provides many functions to examine features of vectors and other objects, for example

● class( ) - what kind of object is it (high-level)?

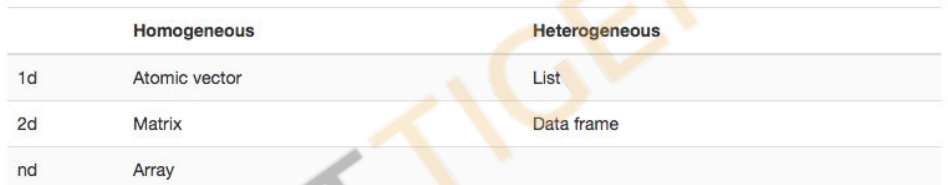
● typeof( ) - what is the object’s data type (low-level)?看它里面每个元素的data type

● length( ) - how long is it? What about two dimensional objects?

● attributes( ) - does it have any metadata元数据?

Double是和integer对应的：

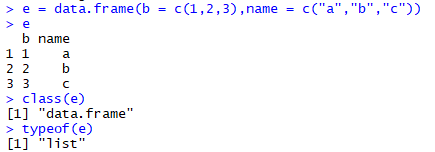
**Data Structure**



Dimensions

Homogeneous: **all columns** must be of the same type

Heterogeneous: columns can be of different types



Class看的是整体的类型，typeof看的是里面的元素是什么类型

**运行的快捷键：ctrl+enter**

R中按**向上箭头**可以调出之前写过的代码。

**Creating Vector**

* The **c() function** can be used to **create vectors of objects**. c是combine的意思

> x <- c(0.5, 0.6) #create a float/double vector

> x <- c(TRUE, FALSE) #create a logical vector

> x <- c(T, F)

> x <- c("a", "b", "c") ## character

> x <- 9:29 ## integer

**The function c() can also be used to add elements to a vector**.

> x <- c(0.5, 0.6)

> x <- c(x, 0.7)

* create vectors as **a sequence of numbers**.

x <- 1**:**10

X <- **seq**(10)

**Mixing Objects**

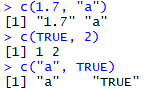
y <- c(1.7, "a") ## character

> y <- c(TRUE, 2) ## numeric

> y <- c("a", TRUE) ## character

When different objects are mixed in a vector, coercion强迫 occurs so that every element in the vector is of the same class.

当数据类型在一个vector中不一样的时候，会自动做数据的转换，转换成同一类型。一般会转换到“不能转换的那个类型”，比如字母没法看做数字，只能把1.7看成字符；true和2：没法把2看成true还是false，会把true转换成1；a和true会把true转换成文字



**Explicit Coercion**

在R中不一定要做自动转换，可以手动转换类型，强制转换

Objects can be explicitly coerced from one class to another using the **as.\*functions**.

> x <- 0:6

> class(x)

[1] "integer"

> **as.numeric(x)**

[1] 0 1 2 3 4 5 6

> **as.logical(x)**

[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE

> **as.character(x)**

[1] "0" "1" "2" "3" "4" "5" "6"

> **as.complex(x)**

1. 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i

**有的时候会有没法完成的强制类型转换 Nonsensical coercion results in NAs.**

> x <- c("a", "b", "c")

> as.numeric(x)

[1] NA NA NA

Warning message: NAs introduced by coercion

> as.logical(x)

1. NA NA NA

**Matrices**

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 **(nrow, ncol)**

> m <- **matrix**(nrow = 2, ncol = 3)

>m

[,1] [,2] [,3]

[1,] NA NA NA

[2,] NA NA NA

> dim(m)

1. 2 3

Matrix的syntax：**matrix(values,nrow,ncol)**，这个matrix要取哪些值，有多少行多少列

**Lists**

Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R and you should get to know them well.

> x <- **list**(1, "a", TRUE, 1 + 4i)

>x

[[1]]

[1] 1

[[2]] [1] "a"

[[3]]

[1] TRUE

[[4]]

1. 1+4i

R中直接有list这个function，对应python中的[ ].

List会把每个元素带两层括号，想要取出单个元素要加两层括号比如x[[1]]

**Factors**

Factors are used to represent categorical data. Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label.

● Factors are treated specially by **modelling functions like lm() and glm()**

● Using factors with labels is better than using integers because factors are self-describing; having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.

> x <- factor(c("yes", "yes", "no", "yes", "no"))

>x

[1] yes yes no yes no

Levels: no yes

> table(x)

x

no yes

2 3

The order of the levels can be set using the **levels argument** to factor(). This can be important in linear modelling because the first level is used as the baseline level.

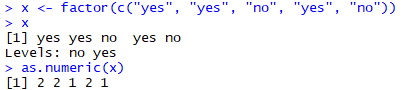
> x <- factor(c("yes", "yes", "no", "yes", "no")),

**levels =** c("yes", "no"))

>x

1. yes yes no yes no Levels: yes no

如果把factor转换成数字类型，就会按照level的顺序，把相应元素转换成1，2....:



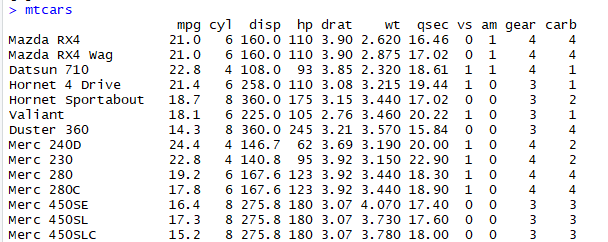
这个特点有时会让factor的类型转换后不make sense比如本来factor是（3,4,5）转换为数字类型就会变成（1,2,3）

**Data Frames**

Data frames are used to store tabular表格的 data

工作中用的最多，很像SQL的database里面的数据，几行几列，可以对每列操作。

例如





平时读入csv files自动默认是dataframe

● They are represented as a special type of list where every element of the list has to have the same length

● Each element of the list can be thought of as a column and the length of each element of the list is the number of rows

● Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class

● Data frames also have a special attribute called **row.names**

● Data frames are usually created by calling **read.table() or read.csv()** Can be converted to a matrix by calling **data.matrix()**

**如何创建dataframe？函数**data.frame

> x <- **data.frame**(foo = 1:4, bar = c(T, T, F, F)) 下划线处为列名，创建df时函数内给出每一列的名字 = 这列的data

>x

foo bar

1 1 TRUE

2 2 TRUE

3 3 FALSE

4 4 FALSE

> nrow(x)

[1] 4

> ncol(x)

1. 2

**Names**

R objects can also have names, which is very useful for writing readable code and self-describing objects.

> x <- 1:3

> names(x)

NULL

> names(x) <- c("foo", "bar", "norf")

>x

foo bar norf

1 2 3

> names(x)

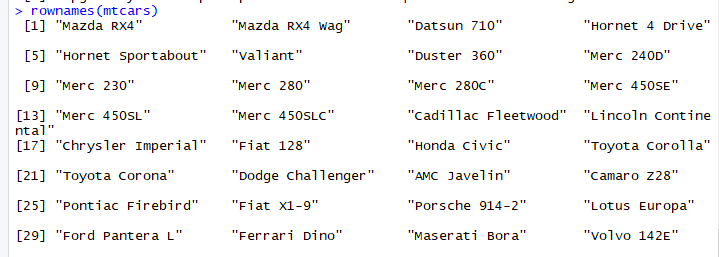
1. "foo" "bar" "norf"

R中如果行或列有名字，可以用names来看，比如names(mtcars),返回了每一列的名字



等同于colnames(mtcars)





这些都是dataframe可以用的functions，来看data

**Missing Values**

**Missing values** are denoted by **NA or NaN** for **undefined mathematical operations**. **is.na()** is used to test objects if they are NA

**is.nan()** is used to test for NaN——判断missing value的两个function

**NA values have a class** also, so there are **integer NA, character NA, etc.**

**A NaN value is also NA but the converse is not true（NA不是NaN）**

NA是空值

NaN就是0/0的结果，也是空值（stands for **Not a Number**） Inf是infinity

“null”是python中missing value的表示，R中是NA；SQL中是is null

Misssing values-NA

Impossible values- NaN

> x <- c(1, 2, NA, 10, 3)

> is.na(x)

[1] FALSE FALSE TRUE FALSE FALSE

> is.nan(x)

[1] FALSE FALSE FALSE FALSE FALSE

> x <- c(1, 2, NaN, NA, 4)

> is.na(x)

[1] FALSE FALSE TRUE TRUE FALSE

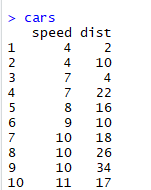
> is.nan(x)

1. FALSE FALSE TRUE FALSE FALSE

**Access Data Subset**

**Overview**

There are three main ways for addressing data inside R objects.提取R objects中数据的三种方法

● By index (slicing)

● By logical vector

● By name (columns only)

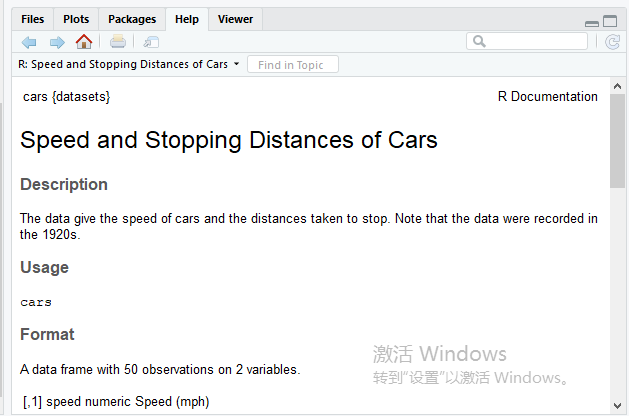
**Load a built-in dataset**

**> data(cars)**

？Cars可以在Rstudio右侧看cars是一个什么样的dataset

用data(cars)就会把这个dataset load 进environment里





By Index （从1数起）

——当我们要access某个data点，by index是最straight forward的方法

加个中括号

**use [row\_numbers,column\_numbers] to access data**

Cars[1, 1]

**Access multiple indexes with : or c( )**

Cars[1:10, 1]

Cars[c(1,3,5), 1]

**: without index means select all. Or you can leave blank**

Cars[:, 1]

Cars[, 1]

**Negative numbers means to select all except this index**

Cars[-1, ]——选择从第二行开始的所有列

Cars[-1:-30,]——去掉前30行，从第31行开始的所有列

这里和python中很不一样，Python中-1是取最后一个index。R中如果想取最后的：cars[1:30, **ncol(cars)**]

By Name

We usually use the**$ operator to address a column by name**

Cars$speed——$ like a pointer in the dataframe, pointing to a column

We can also use ‘column\_name’ and mix with index

Cars[1:10, ‘speed’] ——结果和Cars[1:10, 1]一样

Cars[c(1,3,5), ‘speed’]

By Logical Index

A logical vector contains only the special values TRUE & FALSE. Logical vectors can be created using relational operators e.g. <, >, ==, !=, %in%

Slice with a logical vector will return subset of data with logic = TRUE

x <- c(1, 2, 3, 11, 12, 13)

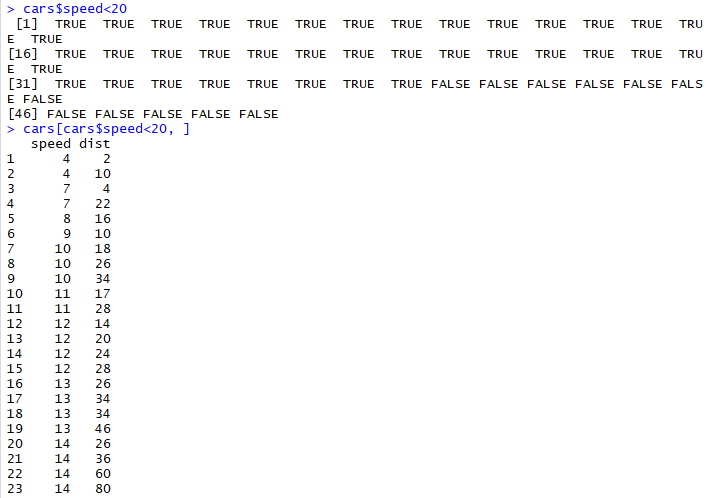
x < 10

[1] TRUE TRUE TRUE FALSE FALSE FALSE

x[x<10]

c(1,2,3)

**Cars[cars$speed < 20, ]**



**Programming in R**

**If statement**

**记住格式，尤其是和python的不同就行**

可以不加else后面的，代表如果不满足condition，就没有action

If ( condition ) {

action1

} else {

action2

}

**For loop**

for ( i in sequence ) {

do something

}

Example

For ( i in 1:10) {

y[i] = i \* (i-1)

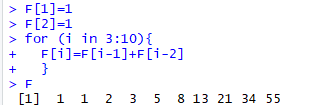
}

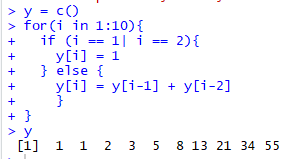
Practice: write a Fibonacci sequence in R

R中没有逗号，必须要换行写多条赋值语句

R中的for loop中不用写i = i+1来增加i，for i in xxx中i是自动加1的（遍历）





注意R中大小写敏感，写成For就一直报错，应该是for

for里面的必须写i **in** ...，不能写for(i **=** 1:10)

一开始可以空出一个**空vector**作为F，用来存放循环出来的值

**F = c()**

**或**

**F = rep(0,20)——rep(x, ...) replicates the values in x. repeat 0 20次**

You can also nest for loops

for(i in 1:3) {

for(j in 1:6){

mymat[i,j] = i\*j

}

}

**While loop**

while (condition) {

do something

}

Condition is evaluated before do something

只要满足这个condition就一直do something

For ( i in 1:10) {

y[i] = i \* (i-1)

}

for中每个循环会把i自动加1；while中不会自动加1，需要强制stop，否则会无限循环，不停的计算同一个i。

所以在while中要写i = i+1

i=1

While (i <=10) {

y[i] = i \* (i-1)

i = i+1

}

**Apply function**

R中一个很好的类似loop的function——apply function

A for loop is used to **apply the same function calls to a collection of objects**. R has a family of functions, **the apply family**, which can be used in much the same way

• apply - apply **over the margins of an array** (e.g. the rows or columns of a matrix)（对matrix或更高维array的margins做aggregation）

• **l**apply - apply over an object and **return list 返回list（**针对每个元素的计算&返回list**）**

• **s**apply - apply over an object and return **a simplified object** (an array) if possible 用的比较多（针对每个元素的操作&返回simplified object like vector）

• **v**apply - similar to sapply **but you specify the type of object returned** by the iterations 返回什么格式的你要自己指定

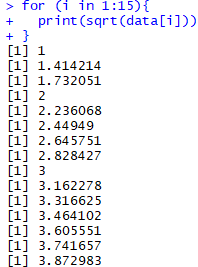
每个具体怎么用敲 ?xapply 来查documentation

比如

data = seq(15)

sapply( data, FUN = sqrt)

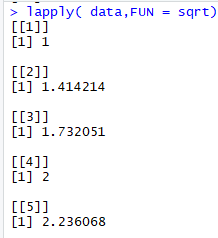




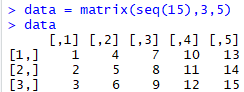
Sapply:

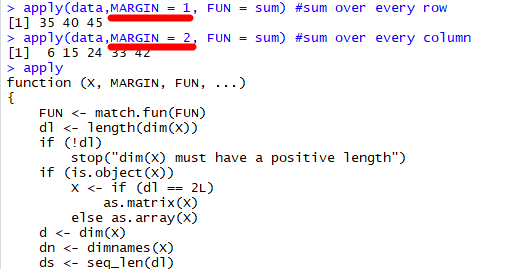


Lapply:



apply：





**Create Function**

You can **define a function** with **function( )** ——创建define自己的function

Your\_function\_name <- function(parameter1, parameter2..){

do something

return(your\_output)

}

Example:

F\_to\_C <- function(temp\_F) {

temp\_C <- ((temp\_F - 32) \* (5 / 9)) + 273.15

return(temp\_C)

}

**Data Loading / Writing in R**

**Load Data**

• Prepare workspace 找到data在哪里，在哪一个文件夹里

○ Getwd( )知道你现在的文件夹在哪, setwd( )：set你的working directory



• Load in data

○ **.txt file**: **read.table( )**

○ .csv file: **read.csv**(file = ‘your\_file\_path/file\_name.csv', header =FALSE) header = T会把第一列当成名字，如果header = F会把第一列当成数据。

○ .excel file: **readWorksheetFromFile from library**

○ Json file, XML file, HTML file

○ other stats software files

○ Relational database and non-relational database

**Write Data** 把R中的output写到一个table/文件里

• Output **.txt file**

**write.table**(dataset,‘output\_name’ ,row.names=F,col.names=F)

• Output csv file

**Write.csv**(dataset, ‘output\_name’)

**Data Exploration in R**

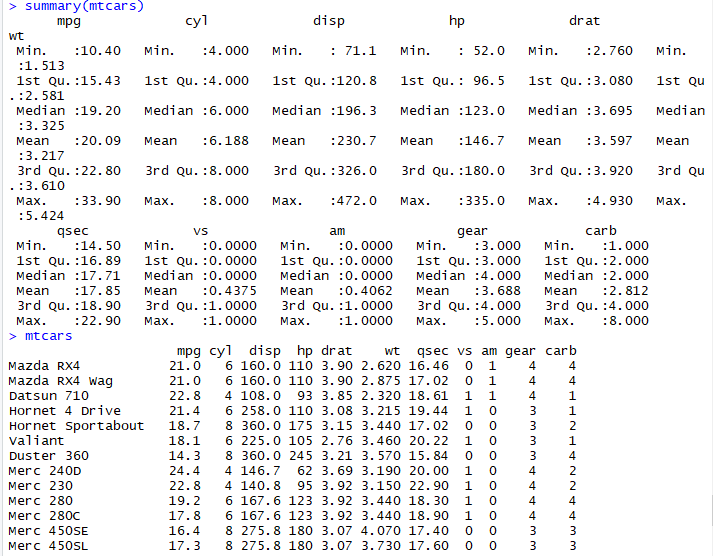
R非常适合做data exploration，因为它有很多非常好用的built-in functions

**Getting to know the data**

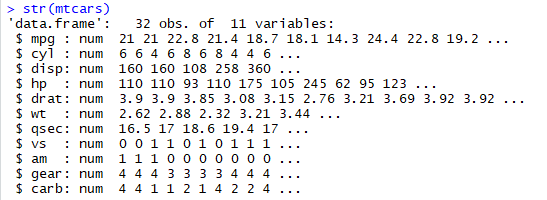
● Metadata

○ summary( ), str( ), dim( ), head( ), colnames( )/rownames( ), length( ), unique( )

Summary: 会给出这个dataframe的summary statistics，会告诉我这个data有哪些行有哪些列，每一列的最小值，最大值，平均值等



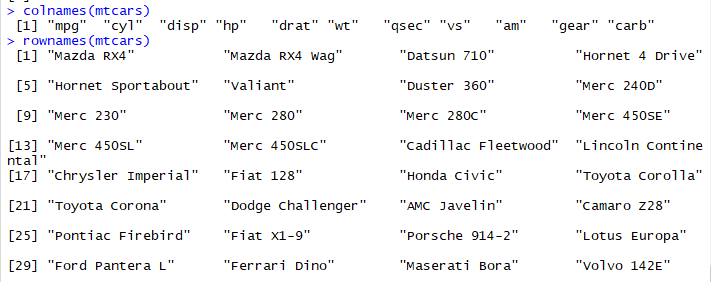
Str( )会给我一个table的structure，会告诉我这是一个dataframe（还是别的啥structure），一共多少个observation（行）多少个variable（列），每一列（每一个variable）的数据类型以及数据举例



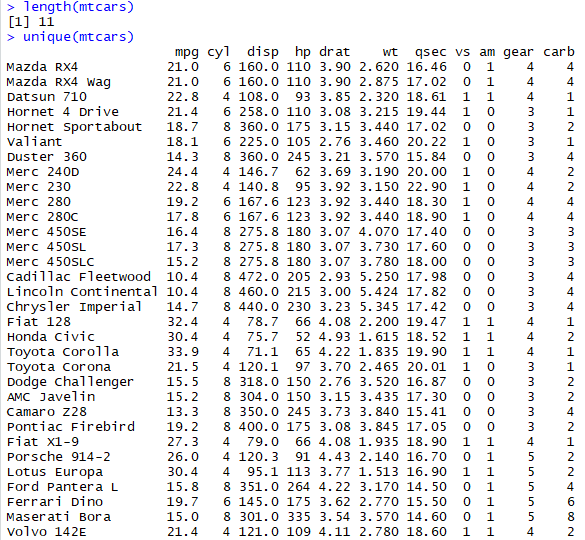
Dim告诉我们的是dimension



行列名：



Length是多少列（每一列算一个元素），unique是给出所有的unique values，mtcars这个df都是unique的所以32行11列都在



● Categorical variable

○ table( ), barplot( ),pie( )——画图

● Continuous variable or ordinal categorical

○ by( ), apply( )...

○ mean( ), median( ), sd( ), quantiles( ), density( ), boxplot( )

○ plot( ), lines( ) to visualize results

**How to understand data**

○ Numeric variables

■ mean(), sd(), quantile(), boxplot(), density()...

○ Categorical variables

■ sort(), table(), barplot()...

○ Numeric variable with numerical response:你有两个variables——x and y，都是数字

■ Cor, library(corrplot)... 可以看一下correlation

○ Caregorical variable with numerical response

■ boxplot(), **by**(), apply(), library(lattice)...

by()是分组的意思。比如体重 by gender：可以分组来看数据

○ Numeric variable with categorical response

■ boxplot(), library(tabplot)...

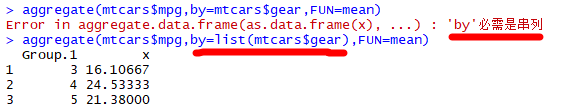
○ Categorical variable with categorial response

**Aggregation & Sorting**

Aggregate function 很常用

**Aggregate(data$age,by=list(data$group),FUN=mean)** 按照group来aggregate年龄，计算每个group的平均年龄

比如说看一下mtcars中每个gear（倒数第二列）的平均mpg是多少：



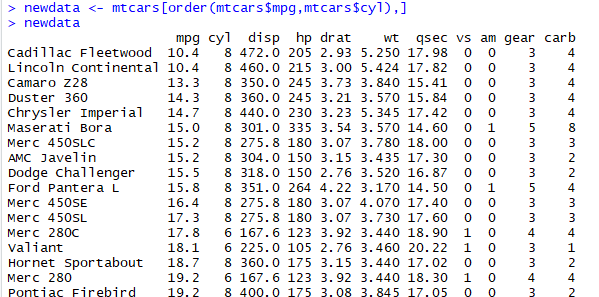
Library(dplyr)——一个很好用的package：deploy R，专门做这样的数据处理

grp<- group\_by(data,group) 这个package中有一个专门的function，group\_by：can group data by group

Summarise(grp,mean=mean(age),sd=sd(age))

**Sorting**——将数据按照某列进行排序

Example: newdata <- mtcars[**order**(mpg,cyl),]——将mtcars按照mpg和cyl排序——如果mpg重复了就按照cyl来



**Merging**

Merge function

Example: merge two data frames by ID

Total <- merge(data frame A, data frame B, by = “ID”)

~~grp <- group\_by(data,group)~~

~~Summarise(grp,mean=mean(age),sd=sd(age))~~

**cbind-ing and rbind-ing**

**Matrices** can be created by **column-binding or row-binding** with **cbind() and rbind().**

> x <- 1:3

> y <- 10:12

> cbind(x, y)——变成两列

x y

[1,] 1 10

[2,] 2 11

[3,] 3 12

> rbind(x, y)——变成两行

[,1] [,2] [,3]

X 1 2 3

y 10 11 12

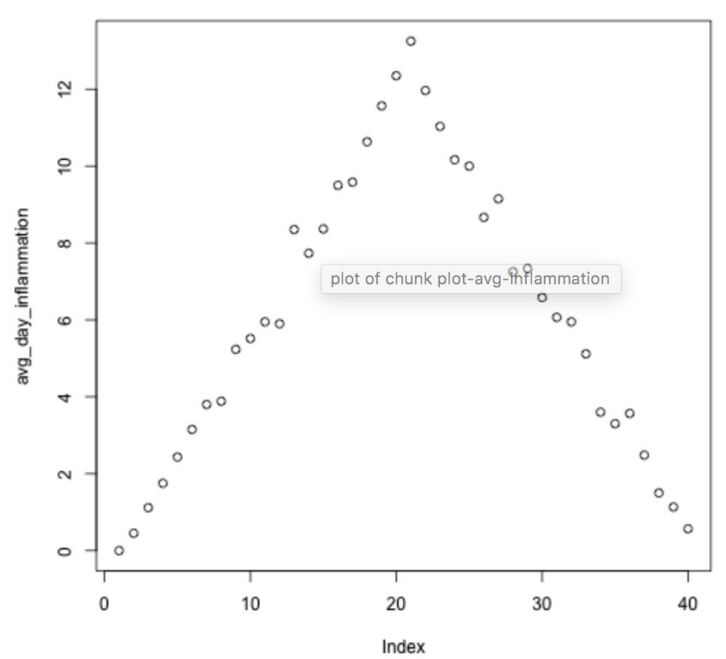
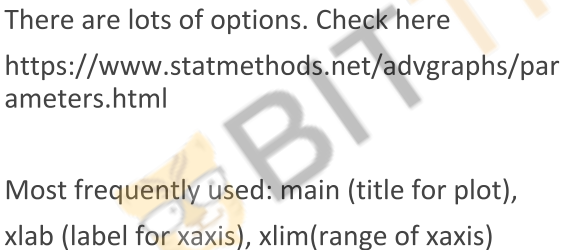
**Plot in R**

不光是R里，BA/DS in general 在什么情况下选择什么样的图？面试中take home project肯定要画图。

Evaluate画图的质量的关键点：（1）**label是否清楚，well labeled**——标题是否标明这是个什么图？X，y轴是否标清了是什么？写清楚明确的定义，不用下划线这些（直接copy列名）（2）正确的运用**axis的range**，数据能完整覆盖但不要太detail，看起来波动特别大，会misleading audience。（3）是否**选用了正确的图表**，很好的表达你的意思，Support your statement。知道什么情况下用什么图表最有效。

**Scatter Plot, line chart**

一般用来看两个variables之间的关系，看相关性（正负相关？）和pattern（有无clusters？）

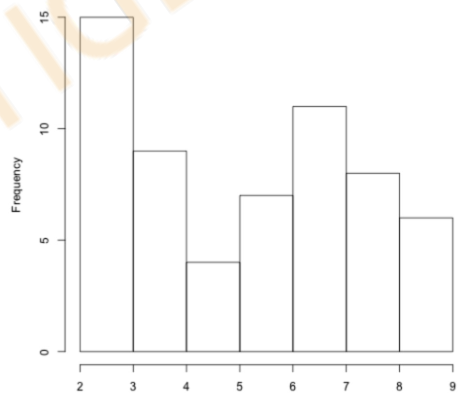
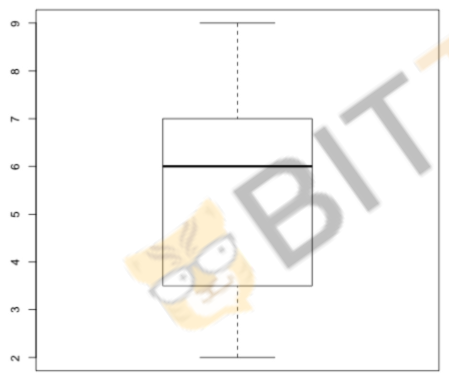


**图的选项**很多：点的颜色、label实心空心、方块圆形、x轴y轴的label，range等等——**paremeters**都可以在以上网址查

Plot(x, y, data = dataset\_name)

**Boxplot, line chart**

https://swcarpentry.github.io/r-novice-inflammation/04-cond/



boxplot(x, data = dataset\_name) hist(x,data = dataset\_name)

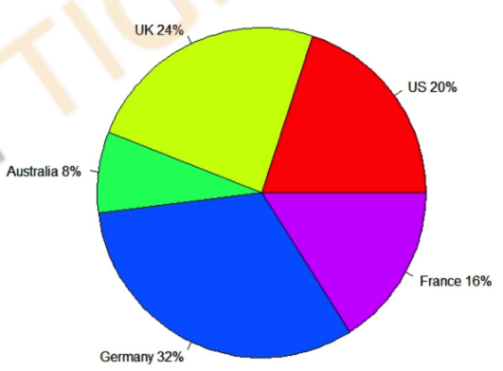
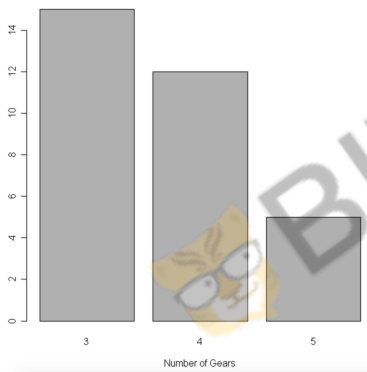
Boxplot中间**黑色的线表示它的中位数**，也有的boxplot会有两条黑线，一条是平均值一条是中位数。**框的上线是Q3**,the third quantile(every 25% is a quantile, the third quantile is 75%). **框的下线是Q1**，25% percentile. 最上面和最下面的**两条须分别表示最大值和最小值**，也有的是显示90%和10%（R中是最大最小值，不同的软件可能有不同定义画的时候具体看一下）

**Boxplot**可以用来看一下data大概是什么分布，看一下中位数均值，25%，75%，最大最小值。看一个**variable的分布**，**有多集中，最大最小值在哪**。另一种用例是**比较几个组**之间有没有区别。比如gender两个组的体重情况有什么区别，可以画**两个boxplots, side by side**，可以很**清楚的看到它们的均值差多少，Q1,Q3的差异**等等。

**Histogram**，**频率图**。x轴是某一个取值，比如每个月看书的数目，y就是统计有多少个人（frequency）在各个catogory里面。有时y轴不是count，会画百分比，某个类里面的个体数量占全体的百分之多少。

**Bar plot, pie chart**

barplot(data) Pie(data)



Bar plot 和histogram长得很像，x轴是几个不同的值/组类，y轴可以是frequency，加和，也可以是平均值。

它们最大的区别：1.**barplot的y-axis可以是任何的值**，不一定是count，**不一定是frequency**，只要是可以画成bar的形状就可以是barplot。2.**barplot可以是任何的data的子集**，**可以显示所有情况**，也**可以只显示几个值**。Histgrom必须展示所有的data的所有分布，比如x的取值是2到8，histogram显示的是所有的情况。可以理解为histogram是barplot的特例——histogram特指是看频率的，看所有的x的情况的barplot。

Piechart是饼状图，专门用来看百分比构成的，用的较少（当category较多时看的不是很清楚明了）。但是是visualize百分比很好的方法。

—————————————————以上都是R中的基本包，base里就能画出来的——————————————

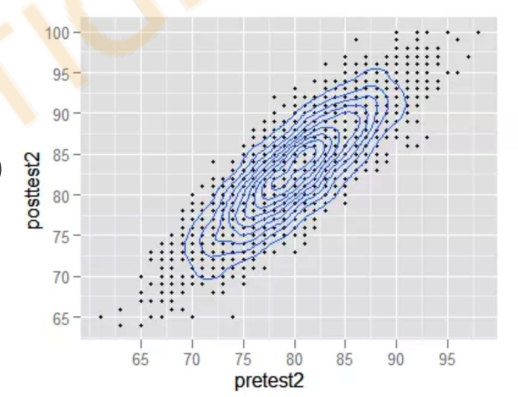
**ggplot2**--R中常用的画图扩展包

ggplot is a **powerful package for plotting** in R. You can make **fancy plots** by **adding more layers**.(like photoshop)

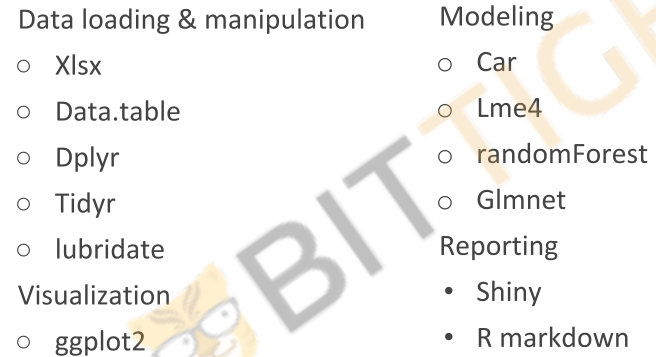
Example: ggplot(temp,aes(x=pretest2,y=posttest2))**+**geom\_point(size=1)**+**geom\_density2d()

基本dataset +散点图 +density图

把图中的元素用 **加号** 不停的加上去，就能画出很多fancy的图。



**R Packages**-Frequently used packages



一般用R时，知道要做的task，要先去google一下怎么完成这个task，google就会告诉你用哪个package里的哪个function。