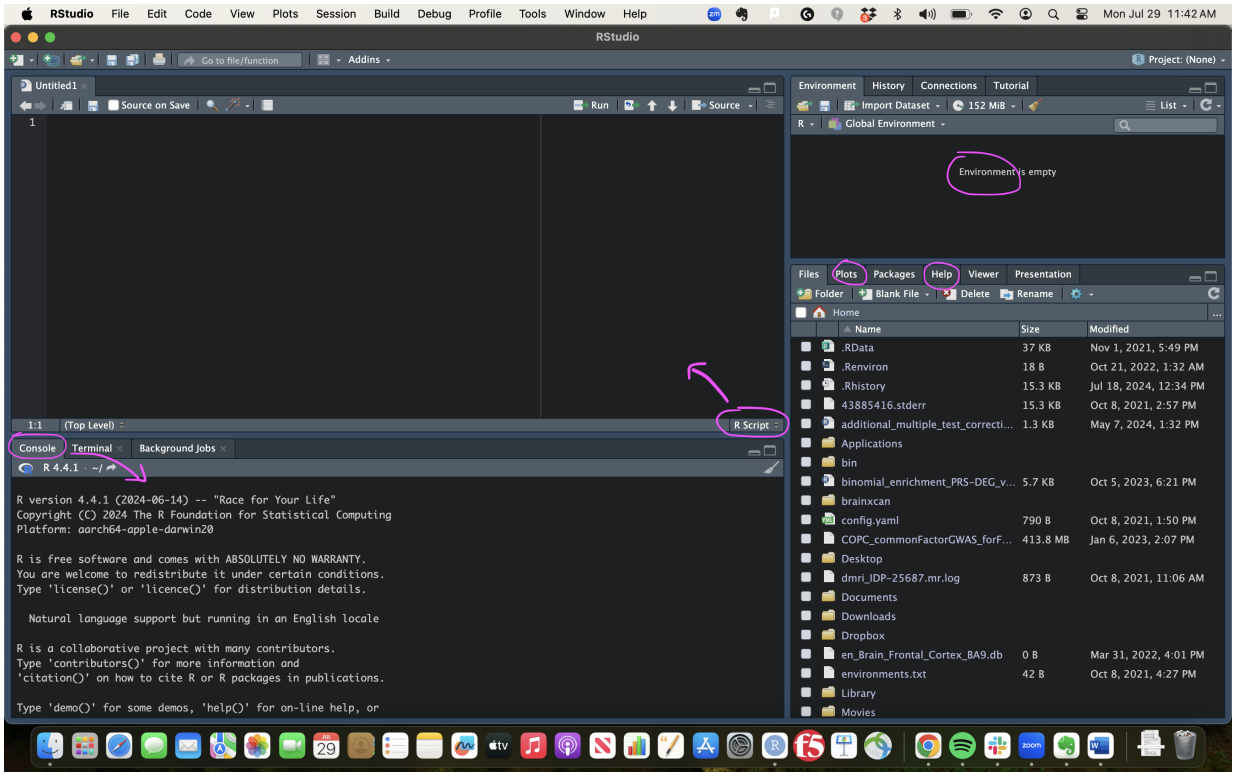
Basic R

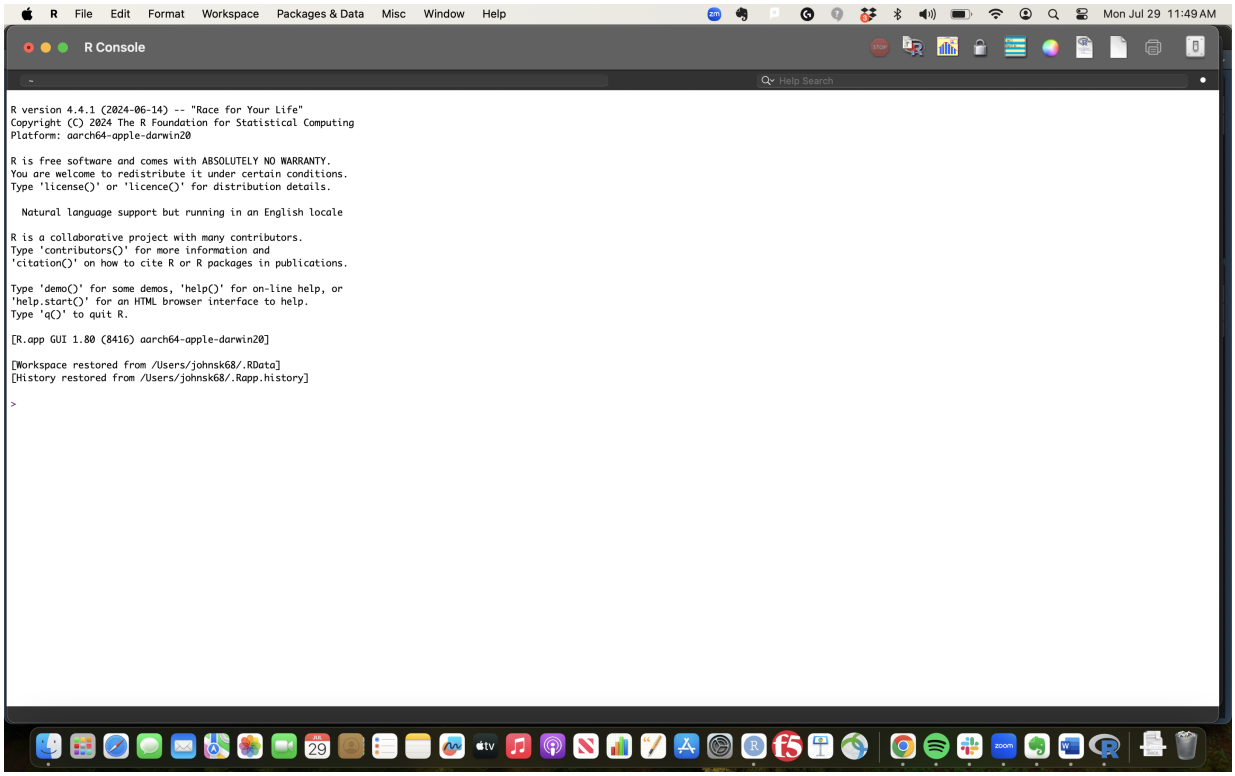
R Studio anatomy



- ?command - show help/manual page for command
- hit tab to help you complete commands
- put your cursor at a line in the R script, ctrl and enter to put it into the console (don't have to copy and paste)
- hit the up arrow when you (your cursor) are in the console, to scroll through your previously-entered lines of code/commands

R (no R Studio)

just the console, none of the helper features



Setting up

- typing in a new script, executing on the console
- cleaning up workspace
- set working directory
- working directory today: wherever you saved the GWAS file from last week

```
rm(list=ls())

setwd("/path/to/your/working/directory")

# setwd("~/Dropbox/Summer_Coding_Camp/")

getwd()

ls()
```

Some simple commands

- using R like a calculator, generating sequences of numbers etc
- logic statements

```
1:10
seq(1,10)
sum(1:10)
seq(1,10, by=2)
rep(5,10)
1+3
1*3
12/3
1>3
1==2/2

print("Hello World")

paste("yay","it is friday",sep=",")
paste0("yay","it is friday (squished)")
```

- ask R to print a sentence of your choosing
- print a sequence of numbers

that is all just printing to the console - what if i want to save/ store information?
first need to know what we are storing

data types in R

- numeric
- integer
- character
- factor
- logical (boolean)

numeric and integer are both numbers - the difference is in precision and size (integers can be a lot bigger) - also integers are always whole numbers
a number is numeric by default
they behave differently in certain situations

how a human reads != how a computer / R reads

```
is.numeric(1)
is.integer(1)
is.integer(1L)
is.numeric(1.5)
```

```
is.integer(1.5)
```

characters are strings - words (but can also represent numbers: "5")

```
is.character("Hello")
is.character("5")
is.character(TRUE)
is.character("TRUE")
```

factors - categorical variables with known set of values (levels)

```
is.factor(1)
is.factor("Hello")
```

logical - TRUE/FALSE

```
1>3
is.logical(1>3)
```

you can convert between data types in most cases, but the behaviour of this varies - always check youve done what you intended ESPECIALLY when converting factors to numeric

```
f <- factor(sample(runif(5), 20, replace = TRUE))
f
as.numeric(f)
# needs special handling
as.numeric(as.character(f))
# slightly faster option
as.numeric(levels(f))[f]
```

Structures of objects (things you store data in) in R

increasing order of complexity

- vectors
- matrices
- data frames, data tables, tibbles
- lists

```
# vectors - concatenation of elements of the same TYPE

x <- c(1,2,3,4,5)
```

```
y <- c("one","two","three","four","five")
z <- 10

x1<-c(1,"a","b","c")

# looking at them again

x

y

z

x1

# or

print(x)
print(y)
print(z)

# what TYPE of data in each vector

class(x)

class(y)

class(z)

str(x)
str(y)
str(z)
```

- what happens when you try to make a vector mixing data types?

```
# matrices - a special case of a vector, with 2 dimensions
# nb - do NOT have to be SQUARE

# empty matrix
```

```
mat<-matrix()
mat

mat<-matrix(nrow=5,ncol=5)
mat

# matrix with values
mat<-matrix(1:25,nrow=5,ncol=5)

# see how it's filled

mat

class(mat)
str(mat)

# data frames
# more general than matrices - data of different types can be stored in
the same tabular array
# made up of vectors of the same length
dat<-data.frame()

dat<-data.frame(cbind(x,y))
dat
class(dat)
str(dat)

# i can change a matrix into a data.frame, since a df is a special case
of a matrix

dat2<-data.frame(mat)
str(dat2)
dat2
```

- how would you go about making an empty 2 x 5 data frame ?

```
# lists
# here we can store collections of objects that are different data TYPES
and LENGTHS/ SIZES
```

```
dat_list<-list()
dat_list[[1]]<-dat
dat_list[[2]]<-mat

dat_list
class(dat_list)
str(dat_list)
```

data tables - data frames, with a few extra features, part of data.table package - some features make it faster/ cleaner to work with data tables, useful utility functions, data tables are a type of data frame (with extras), but not all data frames are data tables
tibbles - special case of data frame, used in the tidyverse - also has nice extras to speed up/ make for a smoother user experience (with tidyverse use)

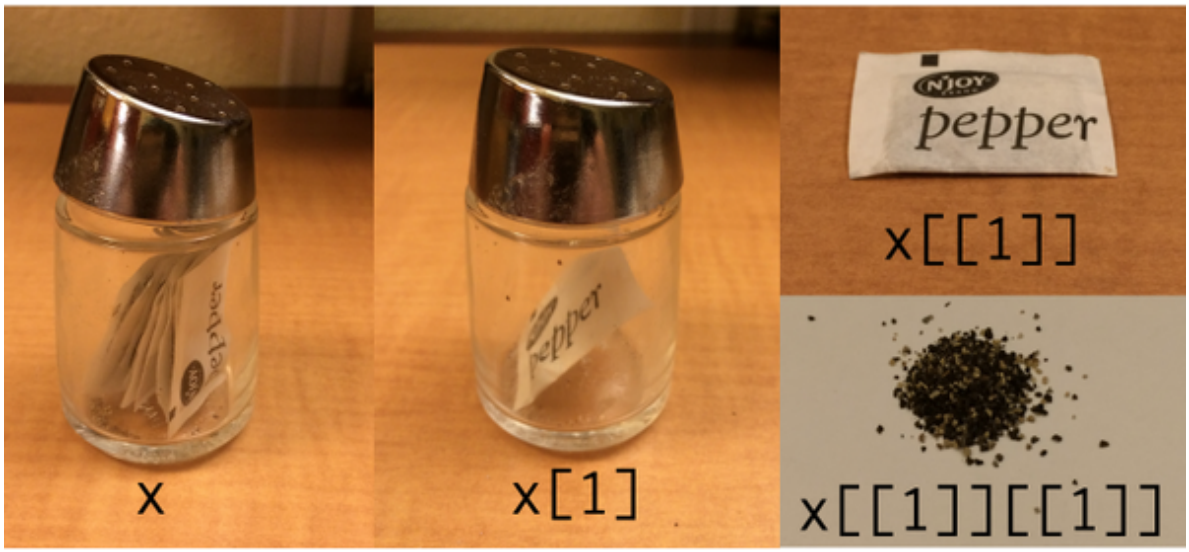
Subsetting data frames

- square brackets [row index, column index]
- double sq brackets
- $ operator
- which statements

```
dat
dat[1,2]
dat[1:2,]

# double sq brackets return the vector that makes up the Nth column

dat[[1]]
dat[[2]]

# single brackets return something that looks similar, but is still
'encased' in being a data frame

dat[1]

str(dat[1])
str(dat[[1]])

# $ operator - subset data frame when you know the column names
```

```
dat$x

# $ operator and sq brackets - subset to give an element in the column

dat$x[1]

# sq brackets and naming the column directly

dat[,"x"]
dat[,"y"]

# sq brackets, $ operator, and setting a condition

dat[which(dat$y=="three"),]

# subsetting lists

# double sq brackets allow us to access the 'bare' item stored in the
list
# single brackets allow access to a 'range', but if used for list will
not return what we are looking for

str(dat_list)
str(dat_list[1])
str(dat_list[[1]])
```

hadley wickham's example



- what is the element (value) in the 3rd row, 1st column of 'dat' ?
- what happens when you try to subset dat using square brackets without " " around the column name?
- how would i subset the dat_list object to give me the element (value) that is in the 1st row, 2nd column of the matrix object ('mat') ?

Reading data into R

the same SCZ GWAS from week 1
using base R
first - what do you plan to read into R? what file type is it? is it where you think it is?

```
getwd()
list.files()
list.files("../")
list.files("~/Downloads")
```

- how can i tell what the file is?
- how could i read in data if it was inside a sub-folder? somewhere else entirely?

```
rm(dat)
dat<-read.table("pgc.scz.2012-04/pgc.scz.full.2012-04.txt")

# what is this?
str(dat)
class(dat)
```

```
dim(dat)

head(dat)
```

- what is wrong with this data frame?

```
rm(dat)

dat<-read.table("pgc.scz.2012-04/pgc.scz.full.2012-04.txt", header=T)
head(dat)
```

using 'fread' function from package 'data.table'
how do we use packages that aren't in base R ?
see if it is already installed, by trying to load it into the environment

```
require(data.table)
# install.packages("data.table")

require(data.table)
library(data.table)

dat_fread_ver<-fread("pgc.scz.2012-04/pgc.scz.full.2012-04.txt")

str(dat_fread_ver)
head(dat_fread_ver)

dat_fread_ver<-data.frame(fread("pgc.scz.2012-04/pgc.scz.full.2012-
04.txt"))

str(dat_fread_ver)
```

nb 'require' and 'library' commands are functionally the same when you are just loading packages on your own machine, or interactively, they have some differences that become relevant in other situations

- what type of object does 'fread' create by default when reading in your file?

we can also use package 'readr'
many other functions: read_delim etc

```
dat_readr_ver<-read_table("pgc.scz.2012-04/pgc.scz.full.2012-
04.txt",col_names=T)
head(dat_readr_ver)
str(dat_readr_ver)
dat_readr_ver<-data.frame(read_table("pgc.scz.2012-04/pgc.scz.full.2012-
04.txt"),col_names=T))
head(dat_readr_ver)
str(dat_readr_ver)
```

saving space - what if we dont want to unzip, to read into R?

```
dat_unzip_ver <- data.frame(fread(cmd = 'unzip -pa pgc.scz.2012-
04/pgc.scz.2012-04.zip pgc.scz.full.2012-04.txt', header="auto"))

head(dat_unzip_ver)

# fixing the columns

unlist(strsplit(colnames(dat_unzip_ver),"[.]"))

colnames(dat_unzip_ver)<-unlist(strsplit(colnames(dat_unzip_ver),"[.]"))
[1:11]
```

Manipulating data frames

- negation operators (! and -)
- other operators (< > == %in%)
- attach
- na.omit
- split
- transposing
- adding new columns

```
# clean up the extra dataframes
ls()
which(ls()=="dat")
-which(ls()=="dat")
ls()[-which(ls()=="dat")]

rm(list=ls()[-which(ls()=="dat")])
```

```
ls()
#
head(dat)
colnames(dat)
str(dat)
summary(dat)
```

```
which(dat$pval < 5*10^-8)

dat[which(dat$pval < 5*10^-8),]
dat[-which(dat$pval < 5*10^-8),]

# without a which statement


dat[dat$pval<5*10^-8,]

# signpost to R, what the columns youre talking about are
# means you don't need to use $ and name the entire df every time
dat[pval<5*10^-8,]

attach(dat)

dat[pval<5*10^-8,]
```

- why can't I just use the column name by itself before i use 'attach'?
- what does the 'which' statement return?

```
# %in%
# make a random sample of SNPs
snps<-sample(dat$snpid)[1:100]

snps

dat$snpid %in% snps
snpid %in% snps
```

```
str(snpid %in% snps)

dat_subset<-dat[snpid %in% snps,]
str(dat_subset)

dat_subset<-dat[!snpid %in% snps,]
str(dat_subset)
```

- what is the snps object?
- when i ask if snpid column vals are %in% snps, what is the output?
- what happens when you try to use negation operator '-' with the output from an %in% query ?

Data cleaning exercise

```
# filtering on p val, maf etc

min(dat$CEUaf)

# reserve the allele freq for comparison

af_check<-dat$CEUaf

# solution 1 - change data type

dat$CEUaf_fix<-as.numeric(dat$CEUaf)

head(dat)
head(af_check)

min(dat$CEUaf_fix)
min(dat$CEUaf_fix,na.rm=T)
# whole-dataset involving solution - replace every "." with an NA

is.na(dat) <-dat=="."

min(dat$CEUaf)

summary(dat$CEUaf)
```

```
# filter by MAF and info
length(which(dat$info>=0.9))
length(dat$info >= 0.9)
summary(dat$CEUaf_fix)

# need to make a MINOR allele freq column

dat$MAF<-dat$CEUaf_fix
dat$MAF[which(dat$CEUaf_fix>0.5)]<- 1 -
dat$MAF[which(dat$CEUaf_fix>0.5)]

summary(dat$MAF)

# removing missing data
dat_na_fix<-na.omit(dat)
dat_maf_info<-dat_na_fix[dat_na_fix$info>=0.9 & dat_na_fix$MAF>0.01,]

dim(dat)
dim(dat_na_fix)
dim(dat_maf_info)

# removing columns i dont care about any more
dat$CEUaf<-NULL
dat[,"CEUaf_fix"]<-NULL

dat$extra<-NA

dat<-dat[,-12]


# split by chromosome
dat_chroms<-list()
dat_chroms[[1]]<-dat_maf_info[hg18chr==1,]
dat_chroms[[2]]<-dat_maf_info[hg18chr==2,]
dat_chroms[[3]]<-dat_maf_info[hg18chr==3,]
# ...and so on

# split function
dat_chroms<-split(dat_maf_info,dat_maf_info$hg18chr)
str(dat_chroms)
```

```
str(dat_chroms[[1]])
head(dat_chroms[[1]])

# transpose
dat_subset<-dat[snpid %in% snps,]
str(dat_subset)

dat_transpose<-t(dat_subset)

head(dat_transpose)
```

- why can't I get a useful min value from the CEUaf column as-is?
- what does the period represent in the CEUaf column?
- what is in the dat_chrom object?
- how are new columns added to an existing data frame?
- what would be one way to remove a column i know the name of, but the index position might vary without my knowledge?

Saving data

what do i want to do with this in future?
WHERE am i saving it?

```
getwd()
```

open in some other program/ on the cmd line at some point

```
write.table(pgc.scz.full.2012-
04_maf_info",row.names=F,col.names=T,quote=F,sep="\t")
```

never open outside of R

```
save(dat_chroms,file="pgc.scz.full.2012-04_maf_info_byChrom.RData")

# multiple items

note_for_later<-"hello"
```

```
save(dat_chroms,note_for_later,file="pgc.scz.full.2012-
04_maf_info_byChrom_extra.RData")
```

reloading things we saved

```
rm(list=ls())
load("pgc.scz.full.2012-04_maf_info_byChrom.RData")
ls()

rm(list=ls())
load("pgc.scz.full.2012-04_maf_info_byChrom_extra.RData")
ls()
```