An Introduction to R

3.2 A few more things you can do in R

Dan Navarro (daniel.navarro@adelaide.edu.au)
School of Psychology, University of Adelaide
ua.edu.au/ccs/people/dan
DSTO R Workshop, 29-Apr-2015

Writing custom functions

R doesn't have the function you want?

- Well, it might... there are 5000 or so package
- But even if it doesn't, you can write your own

- Example:
  - Write a function to quadruple a number
  - Call it quadruple() ???

Done.

```
quadruple <- function( x ) {
               y <- x*4
               return(y)
             }
```

```
> quadruple( 9 )
[1] 36

> quadruple( x=c(2,3,4) )
[1]  8 12 16
```

Basic programming constructs

If & else

```
amHappy <- function( day ) {

              if( day=="Thursday" ) {
                 happy <-TRUE
              } else {
                 happy <- FALSE
              }
              return( happy )

           }
```

```
> amHappy( "green" )
[1] FALSE
> amHappy( "Thursday" )
[1] TRUE
> amHappy( "THURSDAY" )
[1] FALSE
```

Loops

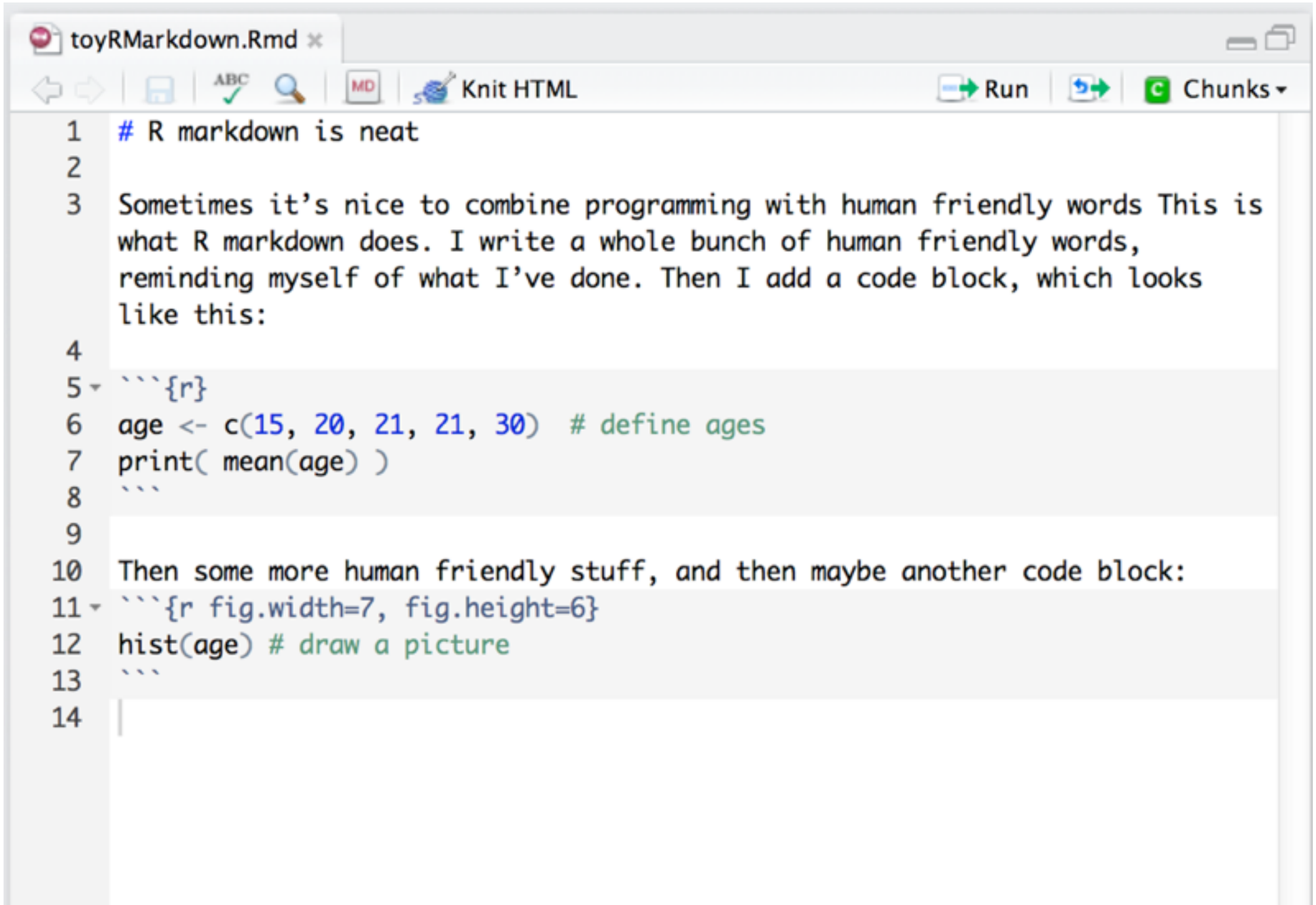
```
firstFivePowers <- function( x ) {
    y <- vector()
    for( i in 1:5 ) {
      y[i] <- x^i
    }
    return(y)
}
```

```
> firstFivePowers( 2 )
[1]  2  4  8 16 32

> firstFivePowers( 3 )
[1]   3   9  27  81 243
```

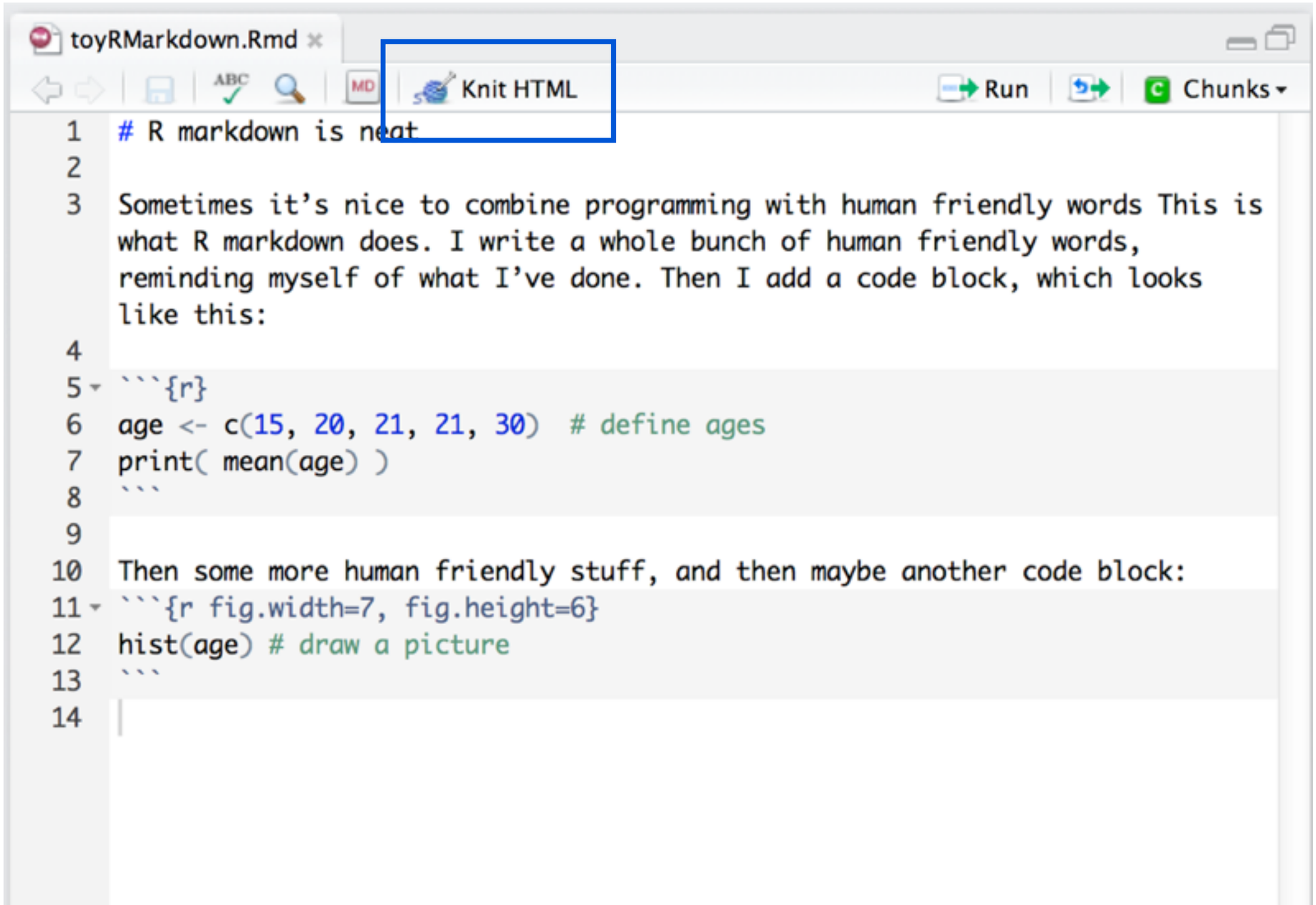
R Markdown

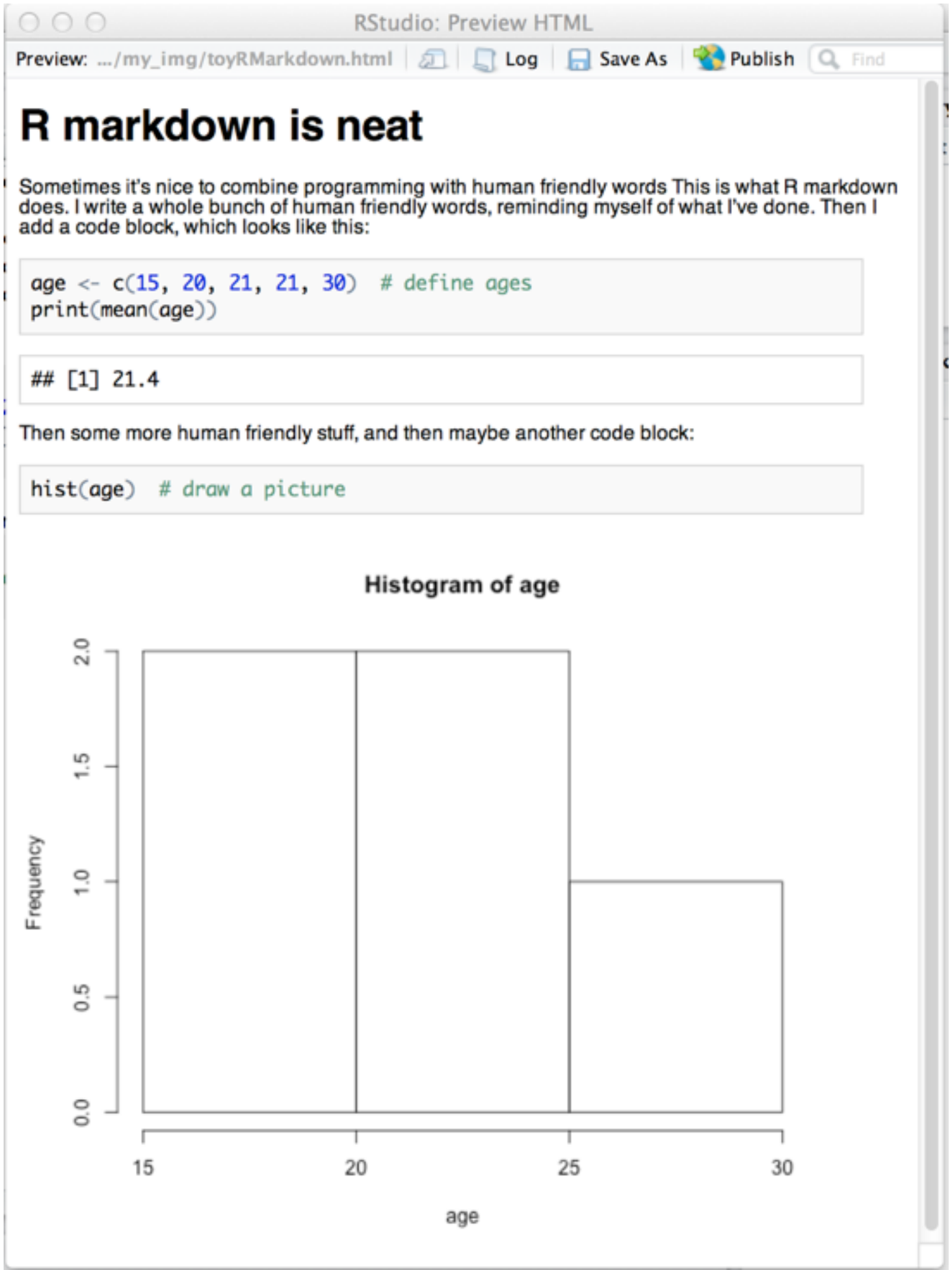
````
# R markdown is neat

Sometimes it’s nice to combine programming with
human friendly words This is what R markdown
does. I write a whole bunch of human friendly
words, reminding myself of what I’ve done. Then
I add a code block, which looks like this:

```{r}
age <- c(15, 20, 21, 21, 30)  # define ages
print( mean(age) )
```

Then some more human friendly stuff, and then
maybe another code block:
```{r fig.width=7, fig.height=6}
hist(age) # draw a picture
```
````

Here it is in Rstudio...

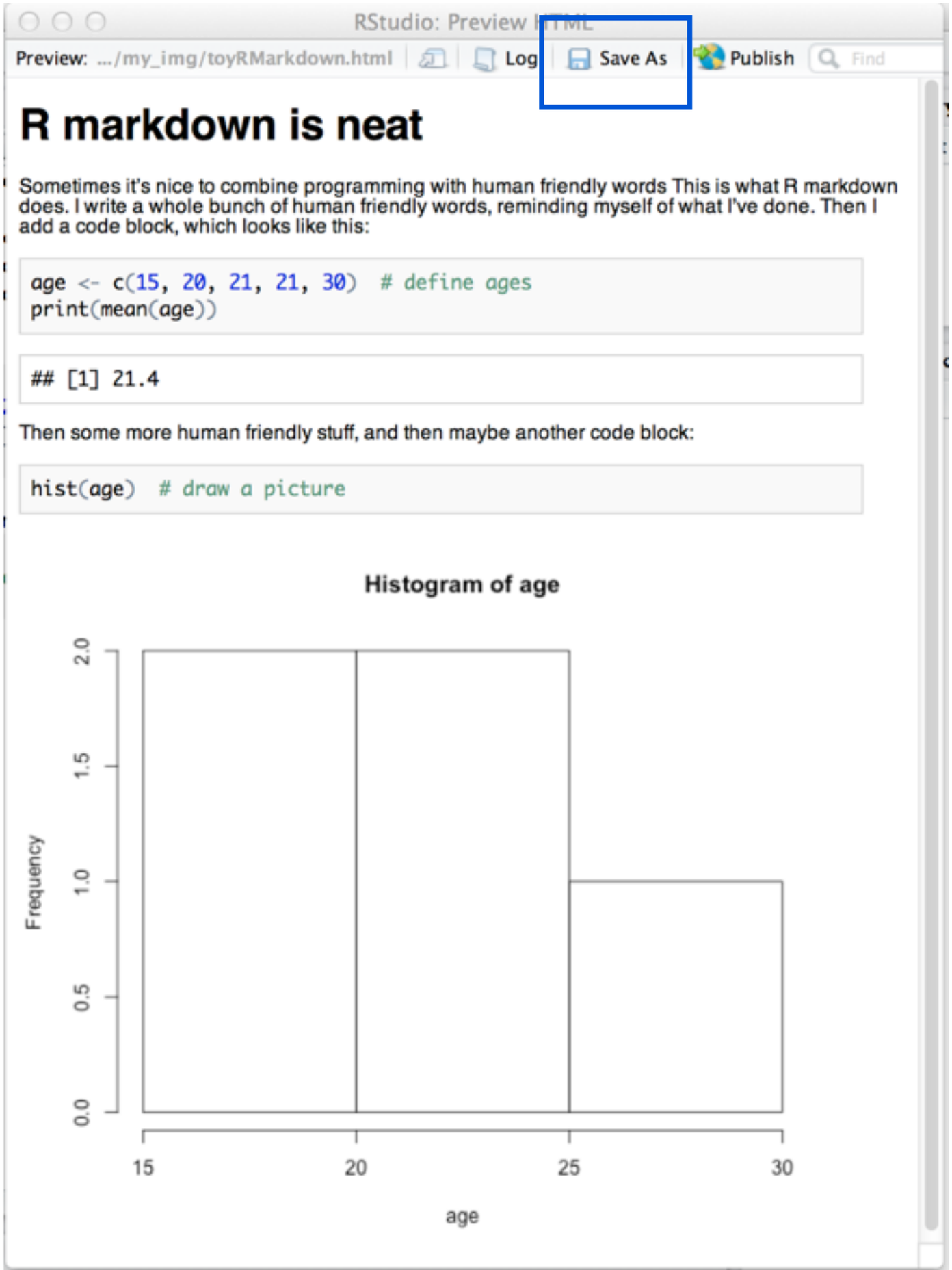


“Knit” it into an HTML file...

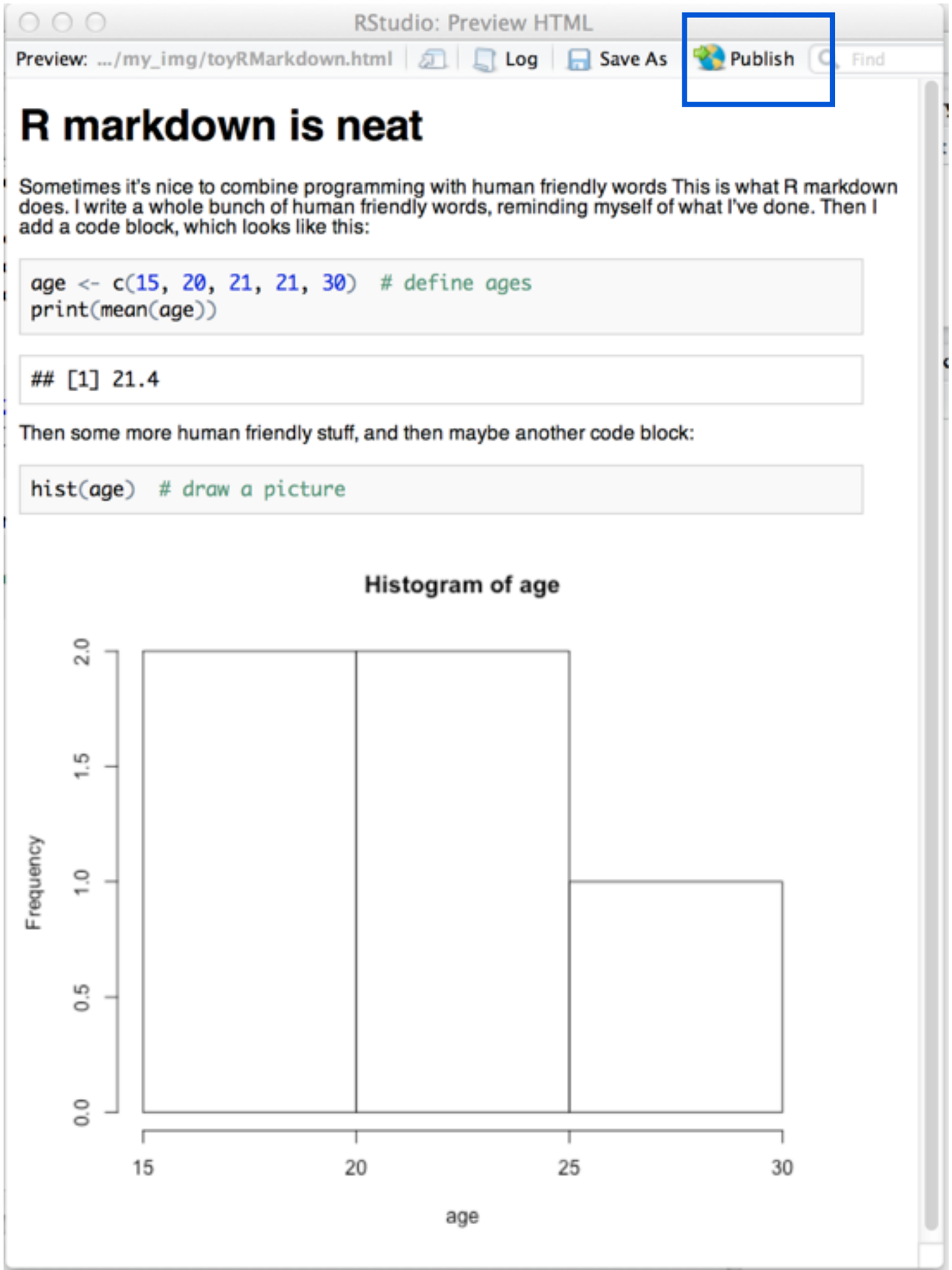




Rstudio embeds the R output directly in with your raw code and your nice human friendly description



You can save this HTML file if you want.



Or, you can publish the file to the open web, with the files hosted on servers owned by Rstudio.

(Haven't done this myself, but everyone tells me it's as simple as point and click stuff)

Working with text

Text processing

- Text data are pretty common, so it's nice to know how to manipulate text
- Variable names are text! Very handy for working with your data by name
- You can get by just fine without knowing any text processing, but once you've learned how to do it, you find uses for it everywhere

A few basics

```
> varNames <- names( expt )
> varNames
[1] "id"        "age"       "gender"    "treatment"
[5] "hormone"   "happy"     "sad"
```

names() pulls the variable names from the data frame and stores them in a vector

A few basics

```
> varNames <- names( expt )
> varNames
[1] "id"        "age"       "gender"    "treatment"
[5] "hormone"   "happy"     "sad"
```

```
> nchar( varNames )
[1] 2 3 6 9 7 5 3
```

nchar() counts the number of characters in each element of the vector

Concatenating strings

```
> paste( "AAA", varNames, "BBB", sep=".")

 [1] "AAA.id.BBB"          "AAA.age.BBB"
 [3] "AAA.gender.BBB"      "AAA.treatment.BBB"
 [5] "AAA.hormone.BBB"     "AAA.happy.BBB"
 [7] "AAA.sad.BBB"
```

paste() can be used to
concatenate strings together...

Changing case

```
> toupper( varNames )

[1] "ID"        "AGE"       "GENDER"    "TREATMENT"
[5] "HORMONE"   "HAPPY"     "SAD"
```

change case using toupper()
and tolower()

Trimming strings

```
> strtrim( varNames, 3 )
[1] "id"  "age" "gen" "tre" "hor" "hap" "sad"
```

use strtrim() to trim varNames
down to (no more than) the first
three characters

Trimming strings

```
> strtrim( varNames, 3 )
[1] "id"  "age" "gen" "tre" "hor" "hap" "sad"
```

```
> substring( varNames, first=2 )
[1] "d"        "ge"       "ender"    "reatment" "ormone"
[6] "appy"     "ad"
```

substring() is more flexible... by specifying first=2, the output "starts" at the 2nd character of varNames

Trimming strings

```
> strtrim( varNames, 3 )
[1] "id"  "age" "gen" "tre" "hor" "hap" "sad"
```

```
> substring( varNames, first=2 )
[1] "d"       "ge"      "ender"   "reatment" "ormone"
[6] "appy"    "ad"
```

```
> substring( varNames, first=2, last=4)
[1] "d"   "ge"  "end" "rea" "orm" "app" "ad"
```

... and if we specify last=4, then we only keep the 2nd through 4th characters

Capitalising the first three letters...

this command trims varNames to the first three characters, and converts to upper case...

```
toupper( strtrim( varNames,3 ))
```

Capitalising the first three letters...

this command trims varNames to the first three characters, and converts to upper case...

```
toupper( strtrim( varNames,3 ))
```

```
substring( varNames, first=4 )
```

and this one drops the first three characters (but doesn't change the case)

Capitalising the first three letters...

```
paste(
  toupper( strtrim( varNames,3 )),
  substring( varNames, first=4 ),
  sep=""
)
```

So... if we paste() the results of these two operations together...

Capitalising the first three letters...

```
> paste(
    toupper( strtrim( varNames,3 )),
    substring( varNames, first=4 ),
    sep=""
  )

[1] "ID"         "AGE"        "GENder"    "TREatment"
[5] "HORmone"    "HAPpy"      "SAD"
```

We get the first three letters capitalised!

Splitting strings

```
> strsplit( "it was the best of times", split=" ")

[[1]]
[1] "it"    "was"   "the"   "best"  "of"    "times"
```

strsplit() does what it says: it takes character data as input, and splits it up into separate variables, using the "split" argument to define the separator

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )
```

grep() is a "pattern matching" function. You specify a "pattern" that the character string needs to match, and it returns those elements that match it!

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )
```

the pattern here is the letter “a”

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )
```

it's a simple "fixed" pattern, not a fancy-pants "regular expression"

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )
```

the data correspond to the strings stored in varNames

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )
```

and R should output the actual strings that match the pattern, not just the indices of the matching strings

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )

[1] "age"       "treatment" "happy"     "sad"
```

yep, those are the variable names that contain the letter “a”

Finding strings that match a pattern

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE,
    value = TRUE
  )

[1] "age"       "treatment" "happy"     "sad"
```

```
> grep(
    pattern = "a",
    x = varNames,
    fixed = TRUE
  )

[1] 2 4 6 7
```

and those are the indices of the variable names that contain the letter “a”

```
> grep(
    pattern = "[aeiou]",
    x = varNames,
    value = TRUE
  )
```

"[aeiou]" is a regular expression that specifies the pattern "contains at least one vowel"

```
> grep(
    pattern = "[aeiou]",
    x = varNames,
    value = TRUE
  )

[1] "id"        "age"       "gender"    "treatment"
[5] "hormone"   "happy"     "sad"
```

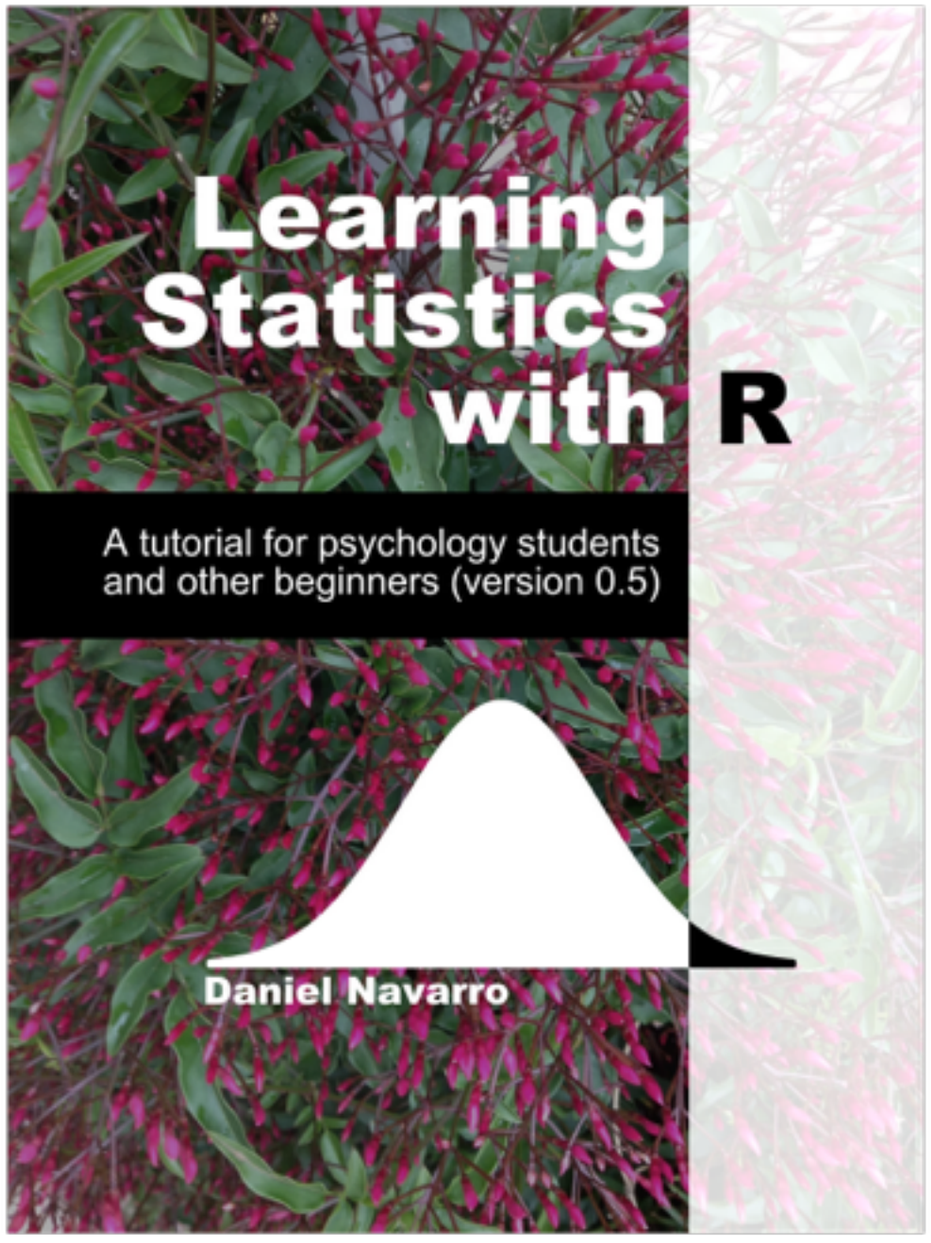
and these are the variable names
that contain at least one vowel (i.e.
all of them!)

```
> grep(
    pattern = "[aeiou]",
    x = varNames,
    value = TRUE
  )

[1] "id"        "age"       "gender"    "treatment"
[5] "hormone"   "happy"     "sad"
```

```
> gsub(
    pattern = "[aeiou]",
    replacement = ".",
    x = varNames
  )

[1] ".d"        ".g."       "g.nd.r"    "tr..tm.nt"
[5] "h.rm.n."   "h.ppy"     "s.d"
```

gsub() performs replacements on anything that matches the pattern... in this case we replace all the vowels with “.”

End of this section.
And the end of the whole thing.



The book

http://ua.edu.au/ccs/teaching/lsr/