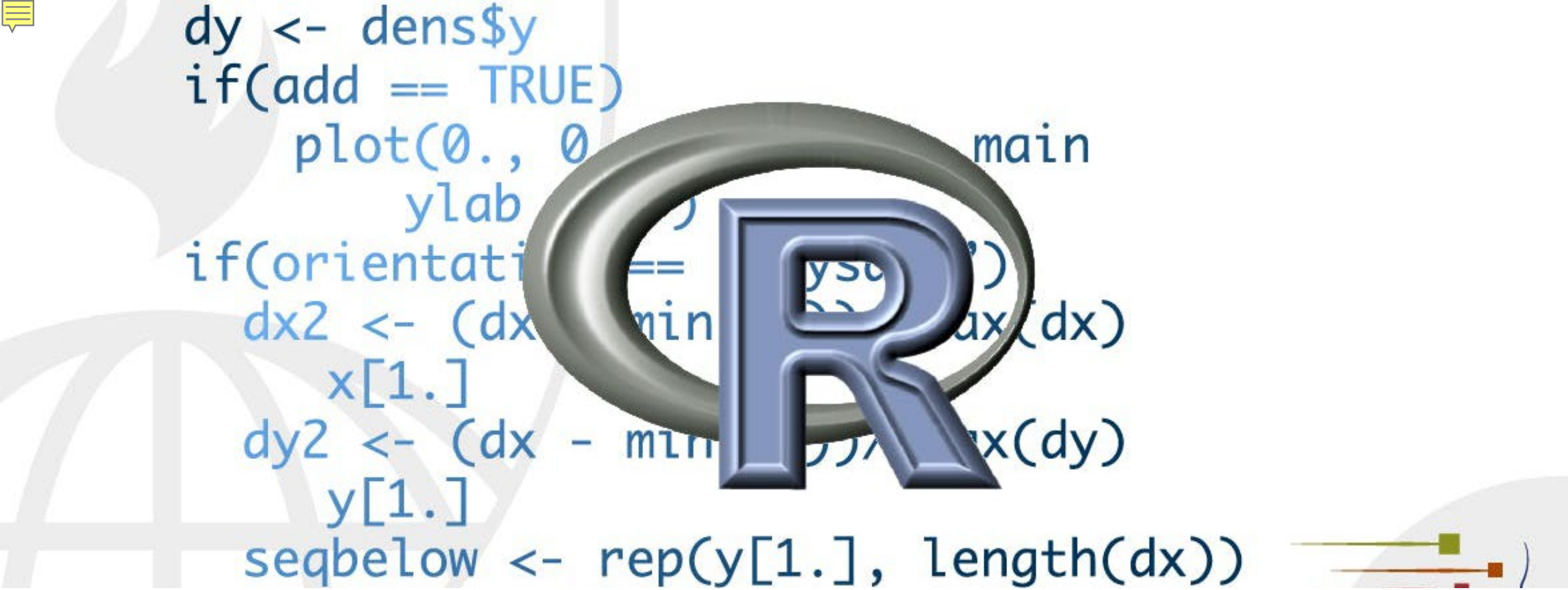
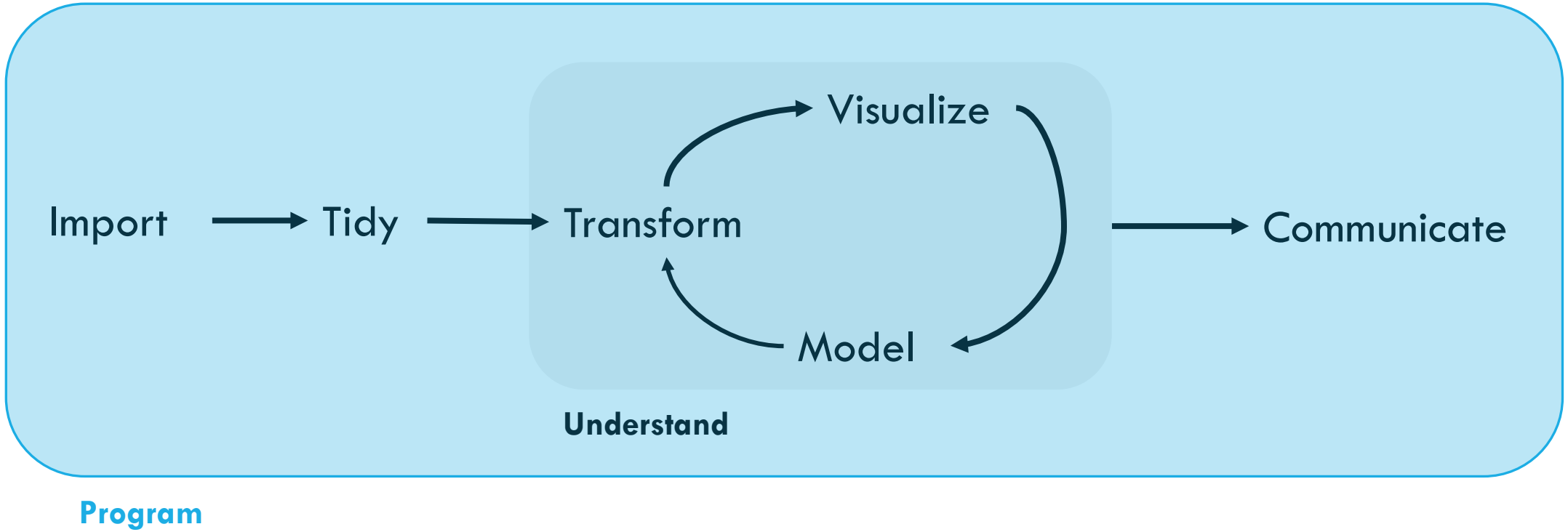


PROGRAMMING IN R

STA 4233 Introduction to Programming and Data Management in R

INTRODUCTION

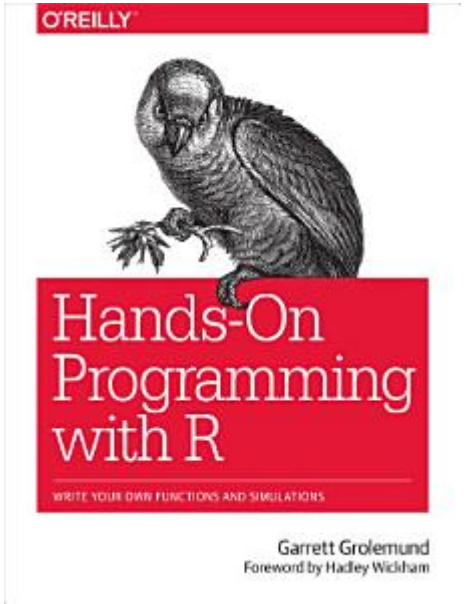


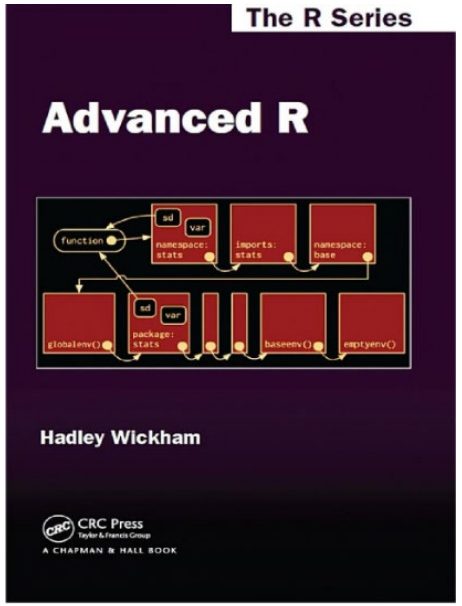
WHAT YOU'LL LEARN

1. dive deep into the pipe, %>%
2. write functions which let you extract out repeated code so that it can be easily reused.
3. R's data structures
4. iteration that let you do similar things again and again

LEARNING MORE

Hands on Programming with R, by Garrett Grolemund





Advanced R by Hadley Wickham

PIPES

```
library(magrittr)
```

PIPING ALTERNATIVES

```
Little bunny Foo Foo
Went hopping through the forest
Scooping up the field mice
And bopping them on the head
```

```
hop()
```

```
scoop()
```

```
bop()
```

```
foo_foo <- little_bunny()
```

1. Save each intermediate step as a new object.
2. Overwrite the original object many times.
3. Compose functions.
4. Use the pipe.

1. INTERMEDIATE STEPS

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

SO MANY COPIES OF DATA

```
diamonds <- ggplot2::diamonds
diamonds2 <- diamonds %>%
  dplyr::mutate(price_per_carat = price / carat)

pryr::object_size(diamonds)
#> Registered S3 method overwritten by 'pryr':
#>   method      from
#>   print.bytes Rcpp
#> 3.46 MB

pryr::object_size(diamonds2)
#> 3.89 MB
pryr::object_size(diamonds, diamonds2)
#> 3.89 MB
```

COLLECTIVE SIZE INCREASES

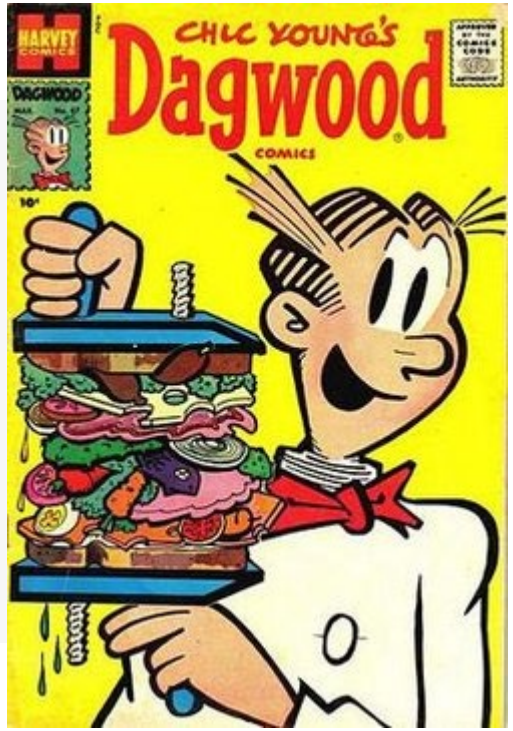
```
diamonds$carat[1] <- NA
pryr::object_size(diamonds)
#> 3.46 MB
pryr::object_size(diamonds2)
#> 3.89 MB
pryr::object_size(diamonds, diamonds2)
#> 4.32 MB
```

2. OVERWRITE THE ORIGINAL

```
foo_foo <- hop(foo_foo, through = forest)
foo_foo <- scoop(foo_foo, up = field_mice)
foo_foo <- bop(foo_foo, on = head)
```

3. FUNCTION COMPOSITION

```
bop(
      scoop(
            hop(fo_foo, through = forest),
            up =  eld_mice
      ),
      on = head
)
```



4. USE THE PIPE

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mice) %>%
  bop(on = head)
```

LEXICAL TRANSFORMATION

```
my_pipe <- function(.) {
  . <- hop(., through = forest)
  . <- scoop(., up = field_mice)
  bop(., on = head)
}
my_pipe(foo_foo)
```

PIPE DOESN'T WORK

Functions that use the current environment.

```
env <- environment()
"x" %>% assign(100, envir = env)
x
#> [1] 100
"x" %>% assign(100)
x
#> [1] 10
```

`get()` `load()`

Functions that use lazy evaluation

```
tryCatch(stop("!"), error = function(e) "An error")
#> [1] "An error"

stop("!") %>%
  tryCatch(error = function(e) "An error")
#> Error in eval(lhs, parent, parent): !
```

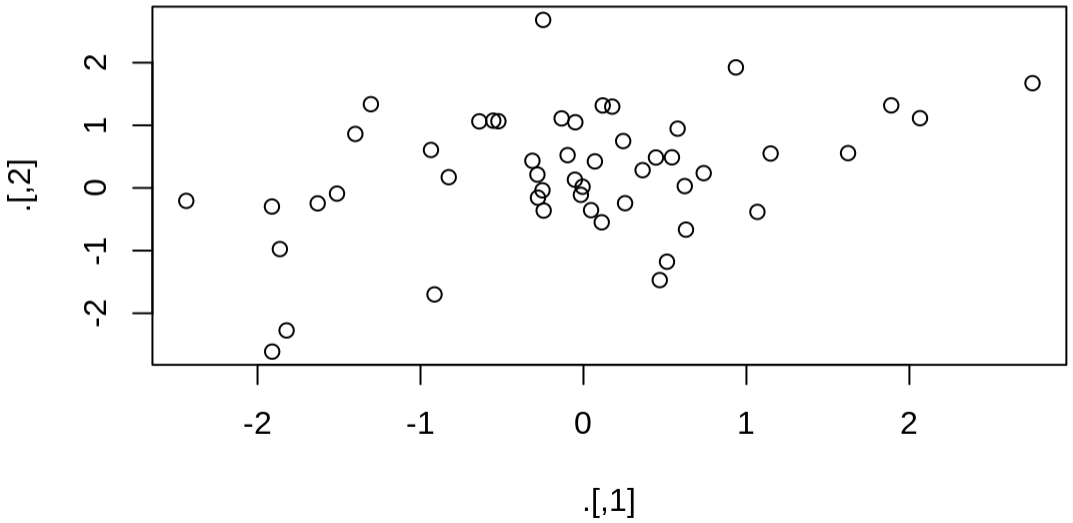
`try()` `suppressMessages()` `suppressWarnings()`

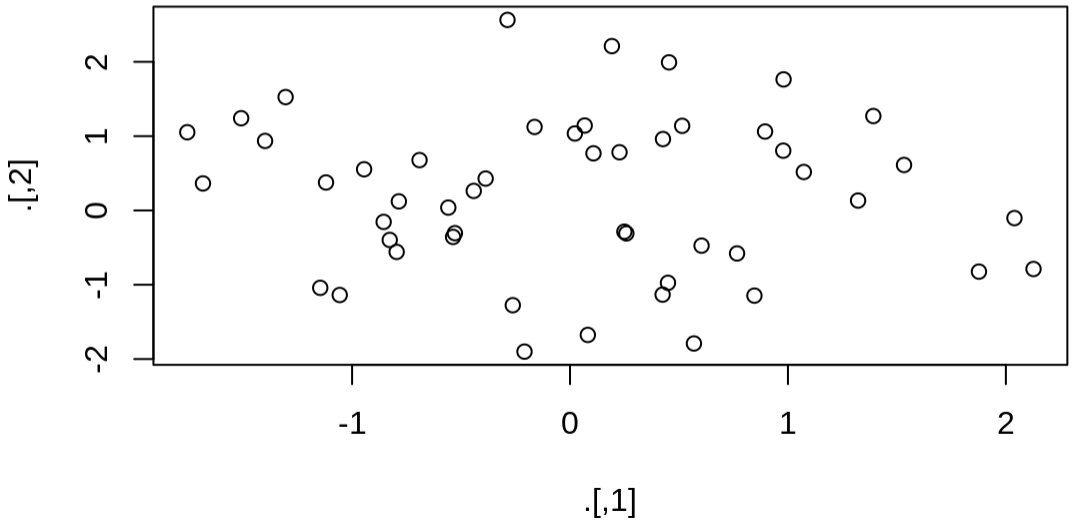
WHEN NOT TO USE THE PIPE

- Your pipes are longer than (say) ten steps
- You have multiple inputs or outputs
- You are starting to think about a directed graph with a complex dependency structure

OTHER TOOLS FROM MAGRITTR %T>%


```
rnorm(100) %>%
  matrix(ncol = 2) %>%
  plot() %>%
```





OTHER TOOLS FROM MAGRITTR %$%

```
mtcars %$%
  cor(disp, mpg)
#> [1] -0.848
```

OTHER TOOLS FROM MAGRITTR %<>%

```
mtcars <- mtcars %>%
  transform(cyl = cyl * 2)
```

with

```
mtcars %<>%
  transform(cyl = cyl * 2)
```