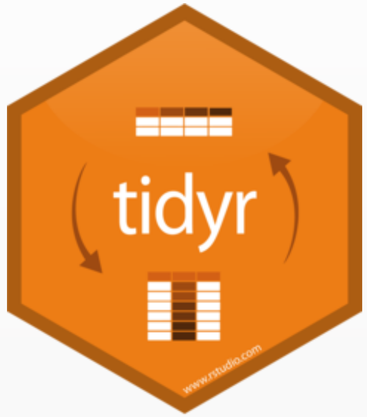
Reusing Tidyverse code

Tidyverse



Data wrangling / visualisation

- Domain oriented
- Language-like interface
- Data is the important scope

- Domain oriented
- Language-like interface
- Data is the important scope

Set of verbs for data manipulation

- `select()`
- `filter()`
- `arrange()`
- `mutate()`
- `group_by()`
- `summarise()`

flights


```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
# … with 336,772 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, …
```


```
flights %>%
  filter(month == 10, day == 10)
```

```
# A tibble: 687 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013    10     5      453            500        -7      624
2  2013    10     5      525            515        10      747
3  2013    10     5      541            545        -4      827
4  2013    10     5      542            545        -3      813
# … with 683 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, …
```


```
flights %>%
  mutate(
    gain = arr_delay - dep_delay,
    gain_per_hour = gain / (air_time / 60)
  )
```

```
# A tibble: 336,776 x 21
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
# … with 336,772 more rows, and 14 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, …
```


```r
flights %>%
  group_by(month) %>%
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE))
```

```
# A tibble: 12 x 2
   month      avg
   <int>    <dbl>
 1     1   6.13
 2     2   5.61
 3     3   5.81
 4     4  11.2
 5     5   3.52
 6     6  16.5
 7     7  16.7
 8     8   6.04
 9     9  -4.02
10    10  -0.167
11    11   0.461
12    12  14.9
```

- **`group_by()`** only affects future computations
- **`summarise()`** makes one summary per level

- Domain oriented
- Language-like interface
- Data is the important scope

```
starwars %>%
  filter(
    height < 200,
    gender == "male"
  )
```

Change context of computation

Translate computation to a SQL query

```
starwars %>%
  filter(
    height < 200,
    gender == "male"
  )
```

```
<SQL>
SELECT *
FROM `starwars`
WHERE ((`height` < 200.0) AND
       (`gender` = 'male'))
```

Transport computation inside a data frame

```
starwars %>%
  filter(
    height < 200,
    gender == "male"
  )
```

```
starwars[starwars$height < 200 &
           starwars$gender ==  "male", ]
```

Data masking


```
data %>%
  fill(year) %>%
  spread(key, count)
```


```
starwars %>%
  filter(
    height < 200,
    gender == "male"
  )
```


```
starwars %>%
  ggplot(aes(height, mass)) +
  geom_point() +
  facet_wrap(vars(hair_color))
```

Data masking

In base R too!

- Inspiration for dplyr
- By R core member Peter Dalgaard

```
starwars %>%
  base::subset(height < 150, name:mass) %>%
  base::transform(height = height / 100)

starwars %>%
  stats::lm(formula = mass ~ height)
```

Data masking


```
library(data.table)

as.data.table(starwars) [
  height < 150,   # rows
  name:mass       # columns
]
```

Data masking built into
the subsetting operator

Creating functions

- Data masking optimised for interactivity and scripts
  → Single-usage pipelines
- Still need to **reuse code** (**D**on't **R**epeat **Y**ourself)

```r
flights %>%
  group_by(month) %>%
  summarise(average = mean(arr_delay, na.rm = TRUE))
```

```r
diamonds %>%
  group_by(cut) %>%
  summarise(average = mean(price, na.rm = TRUE))
```

```r
starwars %>%
  group_by(hair_color) %>%
  summarise(average = mean(height, na.rm = TRUE))
```

```r
flights %>%
  group_by(month) %>%
  summarise(average = mean(arr_delay, na.rm = TRUE))
```

```r
diamonds %>%
  group_by(cut) %>%
  summarise(average = mean(price, na.rm = TRUE))
```

```r
starwars %>%
  group_by(hair_color) %>%
  summarise(average = mean(height, na.rm = TRUE))
```

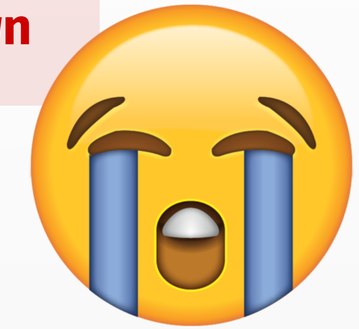
```r
flights %>%
  group_by(month) %>%
  summarise(average = mean(arr_delay, na.rm = TRUE))
```

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by(by) %>%
    summarise(average = mean(var, na.rm = TRUE))
}
```

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by(by) %>%
    summarise(average = mean(var, na.rm = TRUE))
}
```

```r
flights %>% group_mean(arr_delay, by = month)
```

```
Error: Column `by` is unknown
```



How do you Data Mask?

```
list(
  height < 200,
  gender == "male"
)
```

```
Error: object 'height' not found
```

```
starwars %>%
  filter(
    height < 200,
    gender == "male"
  )
```

- Compute as soon as needed
- Compute in the workspace

- Capture blueprints of computations
- Compute in the data mask

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by(by) %>%
    summarise(average = mean(var))
}
```

We got the wrong blueprint!

- We'd like to transport **month**
- We transported **by** instead

```r
flights %>% group_mean(arr_delay, by = month)
```

```
Error: Column `by` is unknown
```

Data masking

- Unique feature of R
- Great for reading/writing data analysis code
- Focus on your data not the data structure

- Creating functions is harder

Reusing Tidyverse code

Tidy eval

- Powers data masking from the rlang package
- Flexible and robust programming

- Strange syntax: `!!` and `!!!`, `enquo()`
- New concepts: Quasiquotation, quosures

Tidy eval

RStudio Community

provocative question: Will tidyeval kill the tidyverse?

tidyverse tidyeval

ehagen 2 Jan 14

Split from it's original thread; Interesting tidy eval use cases 19

Should tidyeval be abandoned?

tidyverse

pavopax 2017-10-25

I'm being provocative on purpose, but I have a point, all in the spirit of the "tenth man rule" .

Long-time dplyr user/advocate here.

I'm struggling with understanding quosures, quasiquotation, !!!

Oct 2017

1 / 57
Oct 2017

Tidy eval

- Documentation efforts to highlight easier patterns

- New embracing operator `{{ arg }}`
  Makes it easy to create tidy eval functions

Reusing Tidyverse code

1. Subset `.data`
2. Pass the dots
3. Embrace args

Reusing Tidyverse code

1. Subset `.data`
2. Pass the dots
3. Embrace args

Data masking

```
diamonds %>% summarise(avg = mean(price))
```

Subsetting .data with $

```
diamonds %>% summarise(avg = mean(.data$price))
```

Subsetting .data with [[

```
var <- "price"
diamonds %>% summarise(avg = mean(.data[[var]]))
```

Subsetting `.data`

```
diamonds %>%
  group_by(cut) %>%
  summarise(avg = mean(price, na.rm = TRUE))
```

Subsetting `.data`

Take column names and pass to `.data[[`

```
group_mean <- function(data, var, by) {
  data %>%
    group_by(.data[[by]]) %>%
    summarise(avg = mean(.data[[var]], na.rm = TRUE))
}
```

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by(.data[[by]]) %>%
    summarise(average = mean(.data[[var]], na.rm = TRUE))
}
```

```r
diamonds %>% group_mean("price", by = "cut")
#> # A tibble: 5 x 2
#>   cut       average
#>   <ord>       <dbl>
#> 1 Fair        4359.
#> 2 Good        3929.
#> 3 Very Good   3982.
#> 4 Premium     4584.
#> 5 Ideal       3458.
```

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by(.data[[by]]) %>%
    summarise(average = mean(.data[[var]], na.rm = TRUE))
}

by <- "cut"
diamonds %>% group_mean("price", by = by)
#> # A tibble: 5 x 2
#>   cut       average
#>   <ord>       <dbl>
#> 1 Fair        4359.
#> 2 Good        3929.
#> 3 Very Good   3982.
#> 4 Premium     4584.
#> 5 Ideal       3458.
```

Reusing Tidyverse code

1. Subset `.data`
2. Pass the dots
3. Embrace args

Taking group counts

```
diamonds %>%
  group_by(cut) %>%
  summarise(count = n())
```

```
# A tibble: 5 x 2
  cut       count
  <ord>     <int>
1 Fair       1610
2 Good       4906
3 Very Good 12082
4 Premium   13791
5 Ideal     21551
```

```
flights %>%
  group_by(month) %>%
  summarise(count = n())
```

```
diamonds %>%
  group_by(cut) %>%
  summarise(count = n())
```

```
starwars %>%
  group_by(hair_color) %>%
  summarise(count = n())
```

Pass the dots

```
starwars %>%
  group_by(hair_color) %>%
  summarise(count = n())
```

Passing the dots

```r
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

Passing the dots

1. Recipient of dots interprets inputs
   - Behaviour of recipient function is **inherited**
   - Automatically masks data
2. **Names** can be overridden
3. Can pass multiple inputs

1. Inherited behaviour

```
diamonds %>% group_count(cut)
```

```
# A tibble: 5 x 2
  cut       count
  <ord>     <int>
1 Fair       1610
2 Good       4906
3 Very Good 12082
4 Premium   13791
5 Ideal     21551
```

```
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

1. Inherited behaviour

```
diamonds %>% group_count(cut(carat, 3))
```

```
# A tibble: 3 x 2
  `cut(carat, 3)` count
  <fct>           <int>
1 (0.2,1.8]       51666
2 (1.8,3.4]        2264
3 (3.4,5]            10
```

```
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

2. Override names

```
diamonds %>% group_count(cut(carat, 3))
```

```
# A tibble: 3 x 2
  `cut(carat, 3)` count
  <fct>           <int>
1 (0.2,1.8]       51666
2 (1.8,3.4]        2264
3 (3.4,5]            10
```

Suboptimal default name?

```
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

2. Override names

```
diamonds %>% group_count(carat = cut(carat, 3))
```

```
# A tibble: 3 x 2
  carat       count
  <fct>       <int>
1 (0.2,1.8] 51666
2 (1.8,3.4] 2264
3 (3.4,5]   10
```

Suboptimal default name?
Just override it!

```
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

3. Multiple inputs

```
diamonds %>% group_count(cut, color, carat = cut(carat, 3))
```

```
# A tibble: 76 x 4
# Groups:   cut, color [35]
   cut   color carat      count
   <ord> <ord> <fct>      <int>
 1 Fair  D     (0.2,1.8]   157
 2 Fair  D     (1.8,3.4]     6
 3 Fair  E     (0.2,1.8]   218
 4 Fair  E     (1.8,3.4]     6
 5 Fair  F     (0.2,1.8]   296
# … with 71 more rows
```

```
group_count <- function(data, ...) {
  data %>%
    group_by(...) %>%
    summarise(count = n())
}
```

Reusing Tidyverse code

1. Subset `.data`
2. Pass the dots
3. Embrace args

Embrace arguments

New syntax: Substitution with `{{ arg }}`

Inspired by the *glue* package:

```
string <- "FOOBAR"
glue::glue("Let's substitute this { string } right here")
```

```
[1] "Let's substitute this FOOBAR right here"
```

Embrace arguments

```
diamonds %>%
  group_by(cut) %>%
  summarise(avg = mean(price, na.rm = TRUE))
```

Embrace arguments

Substitute function arguments with {{

```
group_mean <- function(data, var, by) {
  data %>%
    group_by({{ by }}) %>%
    summarise(avg = mean({{ var }}, na.rm = TRUE))
}
```

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by({{ by }}) %>%
    summarise(average = mean({{ var }}, na.rm = TRUE))
}
```

```r
diamonds %>% group_mean(price, by = cut)
```

```
# A tibble: 5 x 2
  cut       average
  <ord>       <dbl>
1 Fair        4359.
2 Good        3929.
3 Very Good   3982.
4 Premium     4584.
5 Ideal       3458.
```

- Full data masking
- Create vectors on the fly

```r
group_mean <- function(data, var, by) {
  data %>%
    group_by({{ by }}) %>%
    summarise(average = mean({{ var }}, na.rm = TRUE))
}
```

```r
diamonds %>% group_mean(price / 1000, by = cut(carat, 3))
```

```
# A tibble: 5 x 2
  `cut(carat, 3)` average
  <fct>             <dbl>
1 (0.2,1.8]          3.46
2 (1.8,3.4]         14.7
3 (3.4,5]           15.9
```

- Full data masking
- Create vectors on the fly

Embrace arguments

- New syntax — Needs last version of rlang
- Shortcut for `!!enquo(var)`
- `{{ var }}` easier and more intuitive

- **Data masking** is a unique R feature
  - Great for data analysis
  - Harder to program with
- **Easy** techniques for creating functions
  - Subset `.data`
  - Pass the dots
  - Embrace arguments
- Harder techniques still relevant
  - Flexibility and robustness
  - **https://tidyeval.tidyverse.org** (WIP)

https://speakerdeck.com/lionelhenry/reusing-tidyverse-code

Unquote names

!! on the left hand side of :=

```
group_mean <- function(data, var, by, var_name = "avg") {
  data %>%
    group_by({{ by }}) %>%
    summarise(!!var_name := mean({{ var }}, na.rm = TRUE))
}
```

```r
group_mean <- function(data, var, by, var_name = "avg") {
  data %>%
    group_by({{ by }}) %>%
    summarise(!!var_name := mean({{ var }}, na.rm = TRUE))
}
```

```r
diamonds %>% group_mean(price, by = cut)
```

```
# A tibble: 5 x 2
  cut         avg
  <ord>     <dbl>
1 Fair      4359.
2 Good      3929.
3 Very Good 3982.
4 Premium   4584.
5 Ideal     3458.
```

```r
group_mean <- function(data, var, by, var_name = "avg") {
  data %>%
    group_by({{ by }}) %>%
    summarise(!!var_name := mean({{ var }}, na.rm = TRUE))
}
```

```r
diamonds %>% group_mean(price, by = cut, var_name = "price")
```

```
# A tibble: 5 x 2
  cut       price
  <ord>     <dbl>
1 Fair      4359.
2 Good      3929.
3 Very Good 3982.
4 Premium   4584.
5 Ideal     3458.
```

Multiple custom inputs

```
group_mean <- function(data, ..., by) {
  dots <- enquos(..., .named = TRUE)
  dots <- lapply(dots, function(dot) expr(mean(!!dot, na.rm = TRUE)))

  data %>%
    group_by({{ by }}) %>%
    summarise(!!!dots)
}
```

```r
group_mean <- function(data, ..., by) {
  dots <- enquos(..., .named = TRUE)
  dots <- lapply(dots, function(dot) expr(mean(!!dot, na.rm = TRUE)))

  data %>%
    group_by({{ by }}) %>%
    summarise(!!!dots)
}
```

```r
diamonds %>% group_mean(price, depth, by = cut)
```

```
# A tibble: 5 x 3
  cut       price depth
  <ord>     <dbl> <dbl>
1 Fair      4359.  64.0
2 Good      3929.  62.4
3 Very Good 3982.  61.8
4 Premium   4584.  61.3
5 Ideal     3458.  61.7
```

```r
group_mean <- function(data, ..., by) {
  data %>%
    group_by({{ by }}) %>%
    summarise({{{ mean(..., na.rm = TRUE) }}} )
}
```

A terrible idea??

```r
diamonds %>% group_mean(price, depth, by = cut)
```

```
# A tibble: 5 x 3
  cut       price depth
  <ord>     <dbl> <dbl>
1 Fair      4359.  64.0
2 Good      3929.  62.4
3 Very Good 3982.  61.8
4 Premium   4584.  61.3
5 Ideal     3458.  61.7
```

```r
group_mean <- function(data, var, by) {
  data <- as.data.table(data)
  data[, mean(.SD[[var]], na.rm = TRUE), by = by]
}
```

```r
diamonds %>% group_mean("price", by = "cut")
#>           cut       V1
#> 1:      Ideal 3457.542
#> 2:    Premium 4584.258
#> 3:       Good 3928.864
#> 4: Very Good 3981.760
#> 5:       Fair 4358.758
```

`.SD` pronoun in data.table works similarly

```r
group_mean <- function(data, var, by) {
  var <- data[[var]]
  by <- data[[by]]
  aggregate(var, mean, by = list(by), na.rm = TRUE)
}
```

```r
diamonds %>% group_mean("price", by = "cut")
#>           cut       V1
#> 1:      Ideal 3457.542
#> 2:    Premium 4584.258
#> 3:       Good 3928.864
#> 4: Very Good 3981.760
#> 5:       Fair 4358.758
```