Reusing tidyverse code

Wait!

Jenny Bryan, Tidy eval in context

You may better spend your time learning:

1. How to write functions
2. Domain-specific tooling (maps, time series, etc.)
3. Lists, list-columns, nesting, unnesting
4. Functional programming with purrr
5. Scoped dplyr verbs, e.g. `mutate_at()`

?subset, ?transform, ?with

```
subset(gapminder, country == "Chad", select = year:pop)

transform(gapminder, GDP = gdpPercap * pop)

with(gapminder, lifeExp[country == "Chad" & year < 1980])
```

?subset, ?transform, ?with

⚠️ Warning ⚠️

This is a convenience function intended for use interactively. For programming it is better to use the standard subsetting functions like `[`, and in particular the non-standard evaluation of argument `subset` can have unanticipated consequences.

Lionel Henry, UseR 2019

Reusing tidyverse code

Reusing Tidyverse code

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

Subsetting `.data`

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

Subsetting `.data`

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

```r
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.
```

Reusing Tidyverse code

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

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())
}
```

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())
}
```

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

```
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

Resources

- Tidy eval in context, rstudio::conf 2019, Jenny Bryan.
- Reusing tidyverse code, UseR 2019, Lionel Henry.
- Advanced R, ed. 2, ch. 17-21, Hadley Wickham.
- rlang package, Lionel Henry, Hadley Wickham.