Managing many models

February 2016

Hadley Wickham
@hadleywickham
Chief Scientist, RStudio
There are 7 key components of data science

1. Import
2. Tidy
3. Transform
4. Visualise
5. Model
6. Communicate
7. Understand

Automate
Today I want to focus on understanding

Exploratory data analysis

- Import
- Tidy
- Transform
- Visualise
- Model
- Communicate
- Automate
Gapminder data
One way to handle is to fit a model to each country year. Here's an example for New Zealand:

<table>
<thead>
<tr>
<th>Year</th>
<th>Life Expectancy (lifeEx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1952</td>
<td>69.4</td>
</tr>
<tr>
<td>1957</td>
<td>70.3</td>
</tr>
<tr>
<td>1962</td>
<td>71.2</td>
</tr>
<tr>
<td>1967</td>
<td>71.5</td>
</tr>
</tbody>
</table>

The model is fitted as:

```
lm(lifeExp ~ year, data = nz)
```

The $R^2$ is 0.95.

- **Intercept**: -307.7
- **Slope**: 0.19

The residuals for the years are:

<table>
<thead>
<tr>
<th>Year</th>
<th>Residual (resid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1952</td>
<td>0.70</td>
</tr>
<tr>
<td>1957</td>
<td>0.61</td>
</tr>
<tr>
<td>1962</td>
<td>0.63</td>
</tr>
<tr>
<td>1967</td>
<td>-0.05</td>
</tr>
</tbody>
</table>

*Broom*, by David Robinson, makes this easy!
To do that for many countries, we need a list of data frames

<table>
<thead>
<tr>
<th>Country</th>
<th>Year</th>
<th>LifeEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1952</td>
<td>28.9</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>1957</td>
<td>30.3</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Albania</td>
<td>1952</td>
<td>55.2</td>
</tr>
<tr>
<td>Albania</td>
<td>1957</td>
<td>59.3</td>
</tr>
<tr>
<td>Albania</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Algeria</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A nested data frame has one row per group.
We can use `purrr::map()` to fit each model

```r
map(by_country$data, ~ lm(year1950 ~ year, data = .))
```

<table>
<thead>
<tr>
<th>Country</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>Albania</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>Algeria</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;data&gt;</td>
</tr>
</tbody>
</table>

```r
lm(lifeExp ~ year1950, data = afghanistan)
```

```r
lm(lifeExp1950 ~ year, data = albania)
```
Why for loops are bad

An digression with cupcakes
Why for loops are suboptimal

An digression with cupcakes
Vanilla cupcakes

1 cup flour
a scant ¾ cup sugar
1 ½ t baking powder
3 T unsalted butter
½ cup whole milk
1 egg
¼ t pure vanilla extract

Preheat oven to 350°F.

Put the flour, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.

Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Chocolate cupcakes

$\frac{3}{4}$ cup + 2T flour
2 ½ T cocoa powder
a scant $\frac{3}{4}$ cup sugar
1 ½ t baking powder
3 T unsalted butter
½ cup whole milk
1 egg
¼ t pure vanilla extract

Preheat oven to 350°F.

Put the flour, cocoa, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps. Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Chocolate cupcakes

3/4 cup + 2T flour
2 1/2 T cocoa powder
a scant 3/4 cup sugar
1 1/2 t baking powder
3 T unsalted butter
1/2 cup whole milk
1 egg
1/4 t pure vanilla extract

Preheat oven to 350°F.
Put the flour, cocoa, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.
Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.
Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.
Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
df <- data.frame(...)
means <- double(ncol(df))
for(i in seq_along(df)) {
    means[[i]] <- mean(x[[i]], na.rm = TRUE)
}

medians <- double(ncol(df))
for(i in seq_along(df)) {
    median[[i]] <- median(x[[i]], na.rm = TRUE)
}
df <- data.frame(...)  
means <- double(ncol(df)) 
for(i in seq_along(df)) { 
  means[[i]] <- mean(x[[i]], na.rm = TRUE) 
} 

medians <- double(ncol(df))  
for(i in seq_along(df)) { 
  median[[i]] <- median(x[[i]], na.rm = TRUE) 
}
<table>
<thead>
<tr>
<th>Ingredient</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cup flour</td>
<td></td>
</tr>
<tr>
<td>a scant ¾ cup sugar</td>
<td></td>
</tr>
<tr>
<td>1 ½ t baking powder</td>
<td></td>
</tr>
<tr>
<td>3 T unsalted butter</td>
<td></td>
</tr>
<tr>
<td>½ cup whole milk</td>
<td></td>
</tr>
<tr>
<td>1 egg</td>
<td></td>
</tr>
<tr>
<td>¼ t pure vanilla extract</td>
<td></td>
</tr>
</tbody>
</table>

Preheat oven to 350°F.

Put the flour, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps. Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Vanilla cupcakes

120g flour
140g sugar
1.5 t baking powder
40g unsalted butter
120ml milk
1 egg
0.25 t pure vanilla extract

Preheat oven to 170°C.

Put the flour, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.

Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.

1. Convert units
Vanilla cupcakes

120g flour
140g sugar
1.5 t baking powder
40g butter
120ml milk
1 egg
0.25 t vanilla

Beat flour, sugar, baking powder, salt, and butter until sandy. Whisk milk, egg, and vanilla. Mix half into flour mixture until smooth (use high speed). Beat in remaining half. Mix until smooth.

Bake 20-25 min at 170°C.
For loops emphasise the data

df <- data.frame(...

means <- double(ncol(df))
for(i in seq_along(df)) {
    means[[i]] <- mean(x[[i]], na.rm = TRUE)
}

medians <- double(ncol(df))
for(i in seq_along(df)) {
    median[[i]] <- median(x[[i]], na.rm = TRUE)
}
Purrr emphasises the action

```r
library(purrr)
means <- map_dbl(df, mean)
medians <- map_dbl(df, median)
```
Vanilla cupcakes

120g flour
140g sugar
1.5 t baking powder
40g butter
120ml milk
1 egg
0.25 t vanilla

Beat dry ingredients + butter until sandy.
Whisk together wet ingredients. Mix half into dry until smooth (use high speed). Beat in remaining half. Mix until smooth.
Bake 20-25 min at 170°C.
Cupcakes

Beat dry ingredients + butter until sandy.
Whisk together wet ingredients. Mix half into dry until smooth (use high speed). Beat in remaining half. Mix until smooth.
Bake 20-25 min at 170°C.

Vanilla
120g flour
140g sugar
1.5t baking powder
40g butter
120ml milk
1 egg
0.25 t vanilla

Chocolate
100g flour
20g cocoa
140g sugar
1.5t baking powder
40g butter
120ml milk
1 egg
0.25 t vanilla
Similarly, purrr lets you create more complex recipes

```
df <- data.frame(...)  

col_sum <- function(df, f) {  
  df %>%  
    keep(is_numeric) %>%  
    map_dbl(f)  
}  

means <- col_sum(df, mean)  
medians <- col_sum(df, median)  ```
Similarly, purrrr lets you create more complex recipes

df <- data.frame(...)  

col_sum <- function(df, f) {  
  map_dbl(keep(df, is_numeric), f)  
}  

means <- col_sum(df, mean)  
medians <- col_sum(df, median)
## Cupcakes

<table>
<thead>
<tr>
<th></th>
<th>Flour</th>
<th>Baking powder</th>
<th>Sugar</th>
<th>Butter</th>
<th>Egg</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vanilla</strong></td>
<td>120</td>
<td>1.5</td>
<td>140</td>
<td>40</td>
<td>1</td>
<td>0.25t vanilla</td>
</tr>
<tr>
<td><strong>Chocolate</strong></td>
<td>100</td>
<td>1.5</td>
<td>140</td>
<td>40</td>
<td>1</td>
<td>20g cocoa • 0.25t vanilla</td>
</tr>
<tr>
<td><strong>Lemon</strong></td>
<td>120</td>
<td>1.5</td>
<td>140</td>
<td>40</td>
<td>1</td>
<td>2T lemon zest</td>
</tr>
<tr>
<td><strong>Red velvet</strong></td>
<td>150</td>
<td>0</td>
<td>150</td>
<td>60</td>
<td>1</td>
<td>10g cocoa • 20ml red colouring • 1.5t vinegar • 0.5 t baking soda</td>
</tr>
</tbody>
</table>

5. Store as data
In R, we can store functions in lists

```r
funs <- list(
    mean = mean,
    median = median,
    sd = sd
)

map(funs, col_sum, df = df)
```
Back to gapminder
We can use `purrr::map()` to fit each model

```r
map(by_country$data, ~ lm(year1950 ~ year, data = .))
```

<table>
<thead>
<tr>
<th>Country</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>Albania</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>Algeria</td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;data&gt;</td>
</tr>
</tbody>
</table>

```r
lm(lifeExp ~ year1950, data = afghanistan)
lm(lifeExp1950 ~ year, data = albania)
```
map(by_country$data, ~ lm(year1950 ~ year, data = .))

# same as

global <- vector("list", length(by_country$data))
for (i in seq_along(by_country$data)) {
  df <- by_country$data[[i]]
  out[[i]] <- lm(year1950 ~ year, data = df)
}
Multiple lists make it easy to lose context

So use a data frame!
Unnesting is the reverse of nesting.

<table>
<thead>
<tr>
<th>Data</th>
<th>Year</th>
<th>LifeEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1952</td>
<td>28.9</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>1957</td>
<td>30.3</td>
</tr>
<tr>
<td>Afghanistan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albania</td>
<td>1952</td>
<td>55.2</td>
</tr>
<tr>
<td>Albania</td>
<td>1957</td>
<td>59.3</td>
</tr>
<tr>
<td>Albania</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algeria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nest() function takes data and nests it into a structured format.

unnest() function takes the nested data and un-nests it back into its original form.
Cross-validation
<table>
<thead>
<tr>
<th>Test</th>
<th>Training</th>
<th>Model</th>
<th>Prediction</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>df</td>
<td>df</td>
<td>lm</td>
<td>vector</td>
</tr>
<tr>
<td>2</td>
<td>df</td>
<td>df</td>
<td>lm</td>
<td>vector</td>
</tr>
<tr>
<td>3</td>
<td>df</td>
<td>df</td>
<td>lm</td>
<td>vector</td>
</tr>
<tr>
<td>4</td>
<td>df</td>
<td>df</td>
<td>lm</td>
<td>vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
crossv <- partition(mtcars, 100, c(
    test = 0.2,
    training = 0.8
))

crossv <- crossv %>% mutate(
  # Fit the models
  model = map(training, ~ lm(mpg ~ wt, data = .)),
  # Make predictions on test data
  pred = map2(model, test, predict),
  # Evaluate difference between predicted
  diff = map2_dbl(pred, test %>% map("mpg"), msd)
)
Conclusion
1. Store related objects in list-columns.

2. Learn FP so you can focus on verbs, not objects.

3. Use broom to convert models to tidy data.
This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License.

To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/us/