

# Principles for modelling packages

*TBD*

*2018-08-08*



# Contents

<b>1</b>	<b>Intro</b>	<b>5</b>
<b>2</b>	<b>Conceptual overview of modelling</b>	<b>7</b>
<b>3</b>	<b>Getting started on a modelling package</b>	<b>9</b>
<b>4</b>	<b>Model objects</b>	<b>11</b>
<b>5</b>	<b>Data Specification</b>	<b>13</b>
5.1	Formulas . . . . .	13
<b>6</b>	<b>Functional programming principles</b>	<b>15</b>
<b>7</b>	<b>Data</b>	<b>17</b>
<b>8</b>	<b>Documentation</b>	<b>19</b>
<b>9</b>	<b>Testing</b>	<b>21</b>
<b>10</b>	<b>Workflow</b>	<b>23</b>
10.1	Prediction . . . . .	23
10.2	Inference . . . . .	23
<b>11</b>	<b>Interface</b>	<b>25</b>
<b>12</b>	<b>Low and high level interfaces</b>	<b>27</b>
<b>13</b>	<b>Interactive modelling</b>	<b>29</b>
<b>14</b>	<b>Programmatic modelling</b>	<b>31</b>
<b>15</b>	<b>Vocabulary</b>	<b>33</b>
15.1	model frame stuff . . . . .	33
15.2	na.action stuff . . . . .	33
15.3	quoting operators . . . . .	33
<b>16</b>	<b>Naming things</b>	<b>35</b>
16.1	How to name function arguments . . . . .	35
16.2	How to name model components . . . . .	35
<b>17</b>	<b>Danger Zone</b>	<b>37</b>
17.1	Warnings / places to use care . . . . .	37
17.2	Anti-patterns . . . . .	37
17.3	Things to be aware of . . . . .	38



# Chapter 1

## Intro

**Rule 1:** Always spell it *modelling*, never *modeling*.

This document is targetted at R developers writing new packages for modelling.

For users interesting in learning more about using R for modelling in practice, we recommend X and Y



## Chapter 2

# Conceptual overview of modelling

- what is a model: models, estimands, estimators and model specifications
- what do we do with models
- how do fit models
- once we have a fit model, how do we predict or do inference
- the difference between working with a single fit vs a set of fits. LASSO example: wanting to use the coefficients for prediction vs wanting to see the order in which features enter the model





## Chapter 3

# Getting started on a modelling package

General dos:

- Export the `predict()` method
- Document the `predict()` method
- Use `match.arg()` for categorical arguments
- Validate the arguments to all your functions, especially our data

General dont's

-



## Chapter 4

# Model objects

- some explanation of why and how to save the function call
- generally what kinds of things should go into a model object, giving model objects a class so other people can extend them
- S3 object creation and validation for model building a la Advanced R
- Model classes beyond lists. when is S4 worth it? when is R6?
- Every modeling function should include its package version in its data object I will now save my models as a list of three objects: model, data, and `sessioninfo::session_info()`



# Chapter 5

## Data Specification

- formulas, model.frame, term objects, etc
- data / design matrix specification - **recipes**

habit: get the df right, then  $y \sim .$  in the formula. would be nice to still see the features in the call?

- ask users to use data.frames and tibbles, not matrices.

### 5.1 Formulas

#### 5.1.1 Testing formulas

[https://github.com/alexpghayes/formulize/blob/master/tests/testthat/test\\_formula.R](https://github.com/alexpghayes/formulize/blob/master/tests/testthat/test_formula.R)

minimum set of formula tests (based on mtcars example dataset):

- using `as.factor()` inline `mpg ~ as.factor(hp)`
- using `as.character()` inline `mpg ~ as.character(hp)`
- intercept only `mpg ~ 1`
- no intercept `mpg ~ disp + hp + drat - 1`
- implicit intercept `mpg ~ disp + hp + drat`
- explicit intercept `mpg ~ disp + hp + drat + 1`
- polynomials with 1 term `mpg ~ disp + hp + poly(drat, 1)`
- polynomials with multiple terms `mpg ~ disp + hp + poly(drat, 3)`
- natural splines with 1 term `mpg ~ disp + hp + ns(drat, 1)`
- natural splines with multiple terms `mpg ~ disp + hp + ns(drat, 3)`
- explicit interactions `mpg ~ drat + hp + drat:hp`
- dot `mpg ~ .`
- star `mpg ~ hp * drat`
- as.is `mpg ~ hp + I(drat^2)`
- multiple response `cbind`
- multiple responses as matrix `y ~ x` where `y` is a matrix
- multiple predictors as matrix `y ~ x` where `x` is a matrix
- multiple predictors and responses together `y ~ x`, both `x`, `y` matrices
- transformed response `log(mpg) ~ hp + drat`

optional / to: - `survival::Surv` and `survival::strata` objects



## Chapter 6

# Functional programming principles

calls to fit should be pure: i.e. no side effects like plotting, and especially no plotting with invisible object  
return - side effects: useful in interactive mode, irritating in programmatic mode

- type safety, particularly of returned objects
- type safety with respect to single fits vs sets of fits





# Chapter 7

## Data

specification, exported data, and data sets used internally in a package

- using data from the package in tests
- using data from *other* packages in tests



## Chapter 8

# Documentation

- vignette should include not only the coefficients as output in an example, but also those coefficients written up as a general latex model and as a latex model with those specific coefficients substituted in
- **show** your example data in the README so users immediately see the structure

function to write out model form and fitted model in latex for sanity checking: some sort of `model_report` / `model_form` generic. think `report` generic or `write.model` may be coming to `fable/forecast` soon.

it's a bad idea to expect users to learn the *math* for your model from function level documentation, or math presented in ascii or unicode or poorly rendered latex.

show write out the math in a nicely formatted vignette, and then clearly describe the connection between code objects and math objects there as well

documenting arguments:

- **data**: super important to document acceptable **types** and formats, and highly recommend provided a dataset in `data/` with this format so the user can see exactly what they need to provide.

bad doc: The dataset to fit on the model on better doc: A data.frame or tibble with one row per observation and one column per features. For example, `mtcars` is in this format, but `messy_data` is not. It is okay to specify a matrix so long as it can coerced to a tibble. etc etc



## Chapter 9

# Testing

- testing against existing software - say a Matlab implementation
- saving long running models in `R/sysdata.rda` with `usethis::use_data(model_obj, internal = TRUE)`



# Chapter 10

## Workflow

### 10.1 Prediction

1. feature engineering
2. ML wizardry
3. more feature engineering
4. ???
5. predictions

### 10.2 Inference

1. Clean data
2. Specify model
3. Fit model
4. Check that model fitting process converged / worked
5. Check statistical assumptions of model

KEY part that always gets left out: working with multiple modellings





# Chapter 11

## Interface

- user friendly interfaces

good and bad existing idioms

- methods to implement
- examples of tried and true workflows

methods to implement - note on plotting: Should be easy to get the values plotted so others can make their own plots

TWO DISTINCT ISSUES THAT GET RESOLVED IN FORMULAE:

design matrix specification

model specification. (a la `fGarch::garchFit(~arma(1, 1) + garch(1, 0))`)



## Chapter 12

# Low and high level interfaces

- high level versus low level interface
- programmatic versus interactive use

when you should use which

examples: - high level: keras, brms - low level: tensorflow, stan



## Chapter 13

# Interactive modelling

what most people do different because there's a person looking at stuff as opposed to programmatic model when it's just code interacting with the model with no human involved

this is a chapter mostly to remind us to think of differences between the two and how they might be important in terms of interface



## Chapter 14

# Programmatic modelling

i.e. interacting with models programmatically

examples: - packages that export a model from someone to use a la `botornot` - models sitting behind a Plumber API - etc





# Chapter 15

## Vocabulary

useful functions that all modelling package developers should be aware of  
`all.names(terms_object)` `all.vars(terms_object)`

### 15.1 model frame stuff

`model.frame` `mode`

### 15.2 na.action stuff

### 15.3 quoting operators



## Chapter 16

# Naming things

### 16.1 How to name function arguments

### 16.2 How to name model components

some standard names

currently lots of work happening in this realm in `broom` and the Stan community

<https://github.com/tidymodels/broom/issues/452>



# Chapter 17

## Danger Zone

little things to include somewhere: - the danger of misspecified arguments disappearing into ...

don't give your model `lm` class and count on other stuff to magically work. you should implement methods specifically for your class, and if that is just wrapping `lm` internals, great, but take explicit control

provide an example of extensive formula tests: include a log term in tests to catch formula edge cases

### 17.1 Warnings / places to use care

`model.frame()` is not the be all end all

double evaluation issue in

```
fit <- lm(hp ~ log(mpg), mtcars)
predict(fit, newdata = model.frame(fit))
```

```
## Error in eval(predvars, data, env): object 'mpg' not found
```

don't do `predict(object, newdata = model.frame(object))` since this will breaaaaaaaak

### 17.2 Anti-patterns

#### 17.2.1 Using the default method of a generic

i.e. funneling everything into `confint.default`. Default methods for new generics should throw an error.

What do to about existing generics?

Key principle here: want to *guarantee* to the user that they are getting the right numbers

don't funnel everything through `augment_columns` and add special cases slowly - v hard to maintain. much better to have individualized S3 methods with *consistent behavior* abstracted out into small helpers.

related idea to the belong: have enough classes! using inheritance appropriately.

### 17.2.2 the documentation that isn't documentation and doesn't feature an actual use case

- *cough* ?MASS::predict.rlm *cough*
- misisng doc ?MASS::predict.polr()

### 17.2.3 Never use missing arguments

because people have to write more code to pass the objects they want

predict(m) and predict(m, newdata = NULL) should do the same thing you should test this

### 17.2.4 special casing everything through one workhorse function instead of using S3 methods

basically `augment_columns`

don't funnel everything through `augment_columns` and add special cases slowly - v hard to maintain. much better to have individualized S3 methods with *consistent behavior* abstracted out into small helpers.

## 17.3 Things to be aware of

If you call things `data` or `df` and users fail to specify data arguments, R will try and perform dataframe operations on the *functions* `data` and `df`. The resulting error messages for this can be cryptic. In this case you may wish to write an informative error message with a hint:

```
Error: Can't *do data frame thing* on a function.  
Are you sure you passed a tibble to *argument*?
```

## Chapter 18

# References

- bdr's model fitting functions in r