Building a modeling package

useR - July 2019

RStudio®

What are the pieces?

How does hardhat help?

Implement!

What are the pieces?

How does hardhat help?

Implement!

logistic_regression()

The design of a modeling 📦

Model Fitting

✓ High level user interface

✓ Low level fit implementation

✓ Bridge

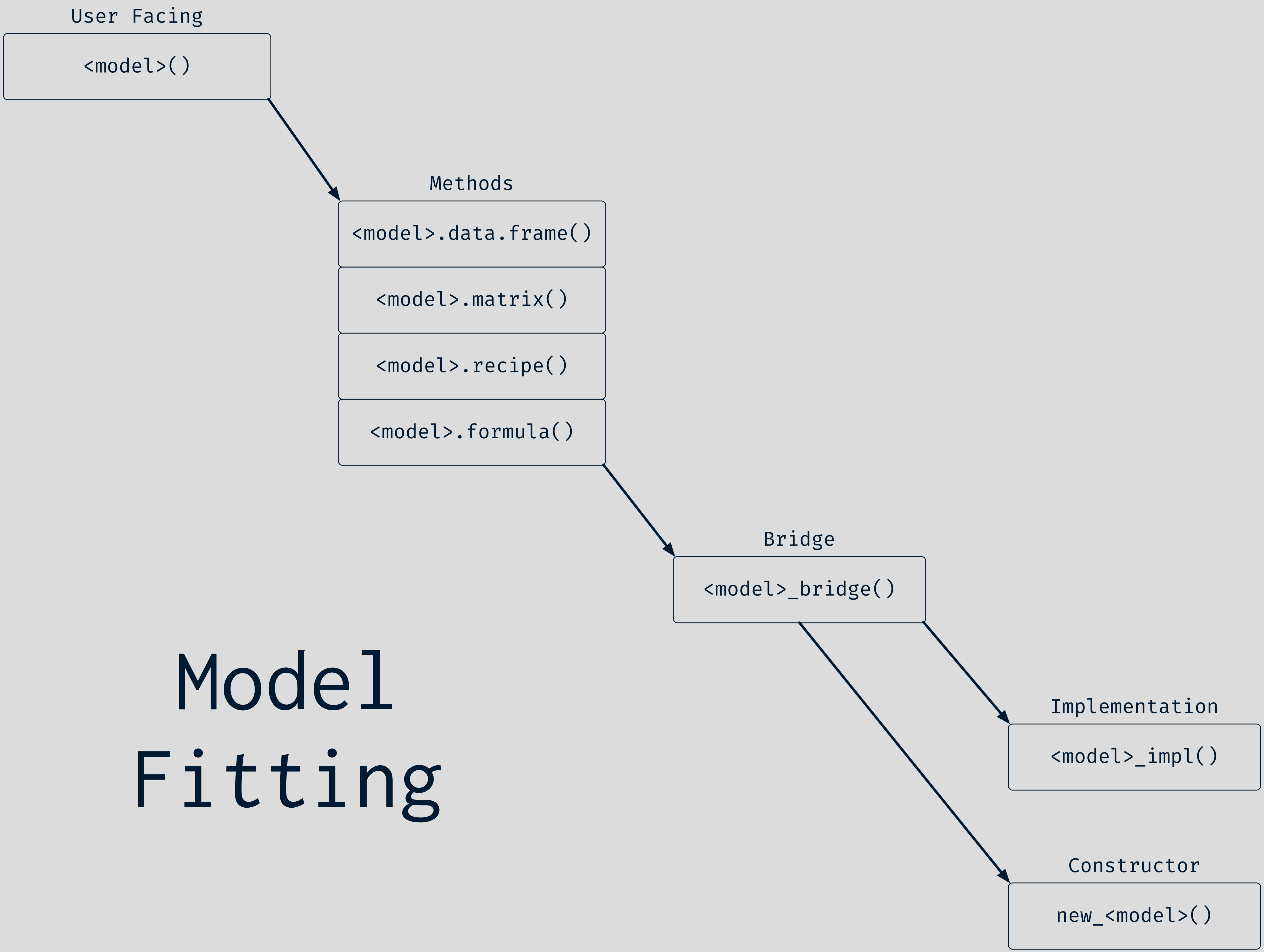
✓ Model constructor

Model Prediction

✓ High level predict() method

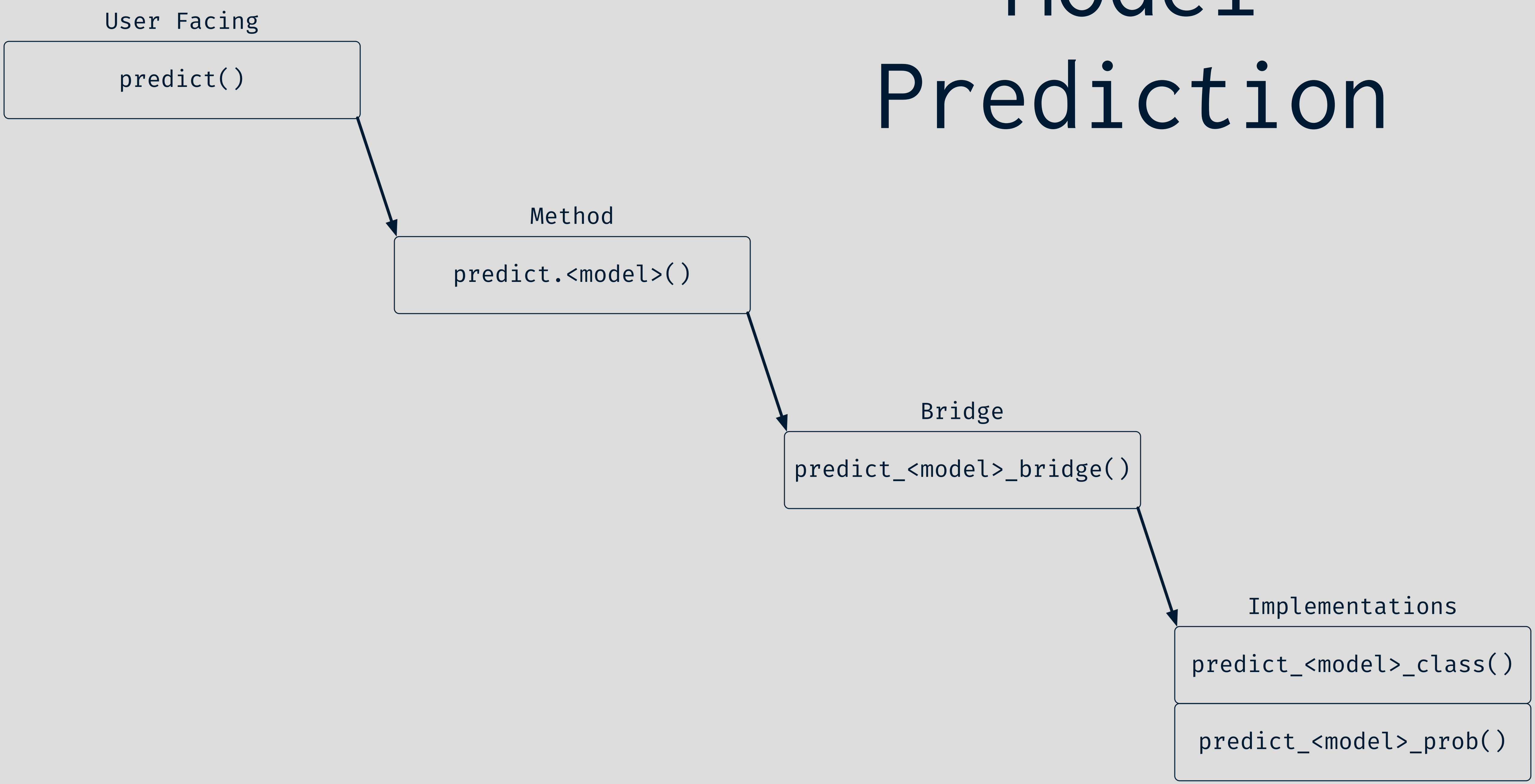
✓ Low level type specific predict implementations

✓ Bridge



Model Fitting

Model Prediction



Model Fitting - Low level implementation

Example

`lm.fit(x, y)`

If you're feeling dangerous: `.lm.fit()`

Recommendations

Don't export it!

Low level inputs: matrix / vector

Assume preprocessing has been done

Return value: A named list for your constructor

ranger

```
# Top level function
ranger::ranger(formula, data)

ranger <- function(formula, data, ...stuff...) {
  # preprocessing

  result <- rangerCpp(...stuff..., data.final, ...stuff...)

  # finalize `result` and add the “ranger” class

  result
}
```

ranger

```
# Top level function
ranger::ranger(formula, data)

ranger <- function(formula, data, ...stuff...) {
  # preprocessing

  result <- rangerCpp(...stuff..., data.final, ...stuff...)

  # finalize `result` and add the “ranger” class

  result
}
```

Raw processed
low level data goodness

ranger

```r
# Top level function
ranger::ranger(formula, data)

ranger <- function(formula, data, ...stuff...) {
  # preprocessing

  result <- rangerCpp(...stuff..., data.final, ...stuff...)

  # finalize `result` and add the "ranger" class

  result
}
```

Named list of fit specific info:
num.trees, forest

Raw processed
low level data goodness

How does hardhat help?

How does hardhat help?

It doesn’t 😬

How does hardhat help?

It doesn’t 😬

This one’s all you

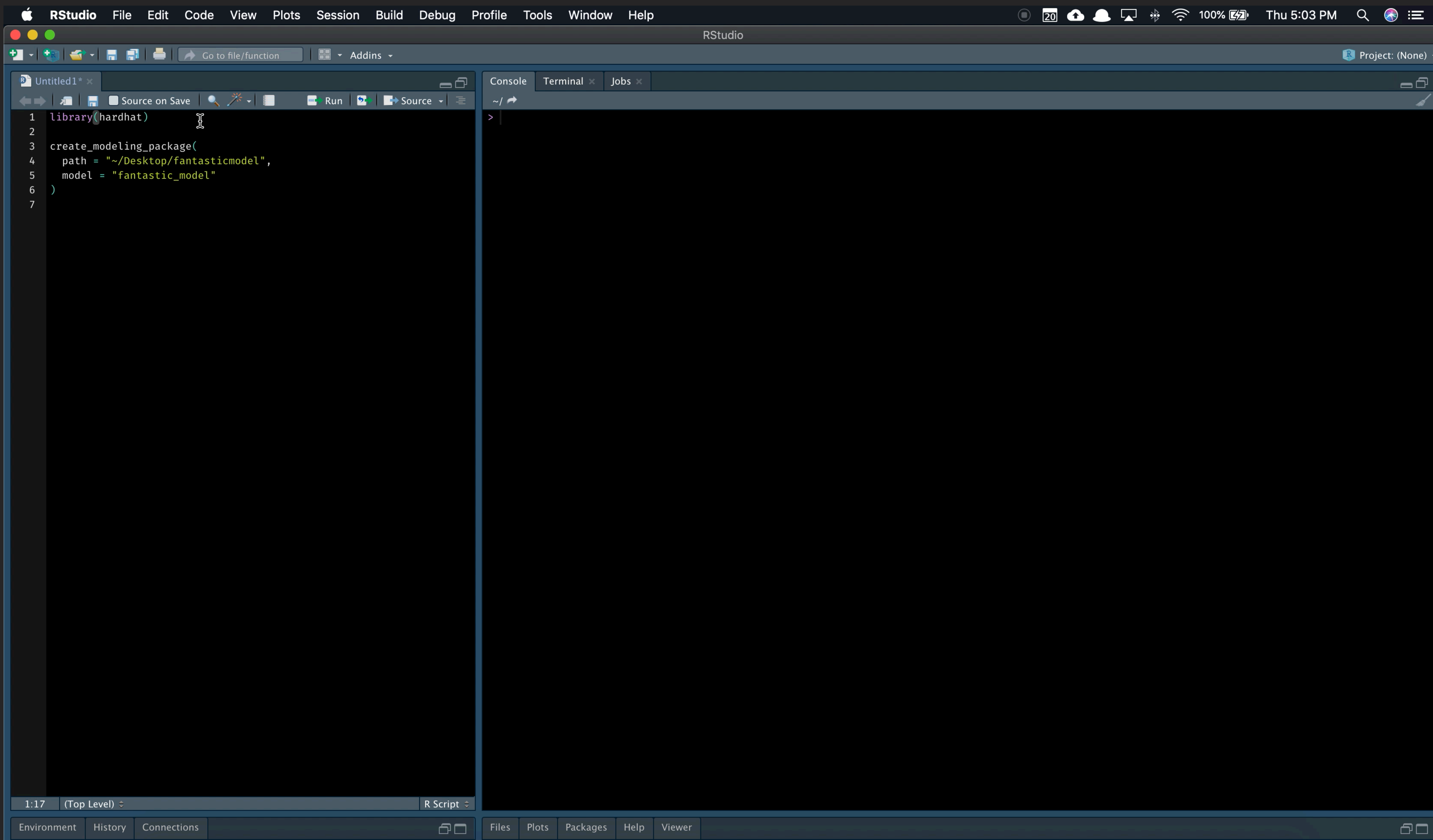
How does hardhat help?

It doesn’t 😬

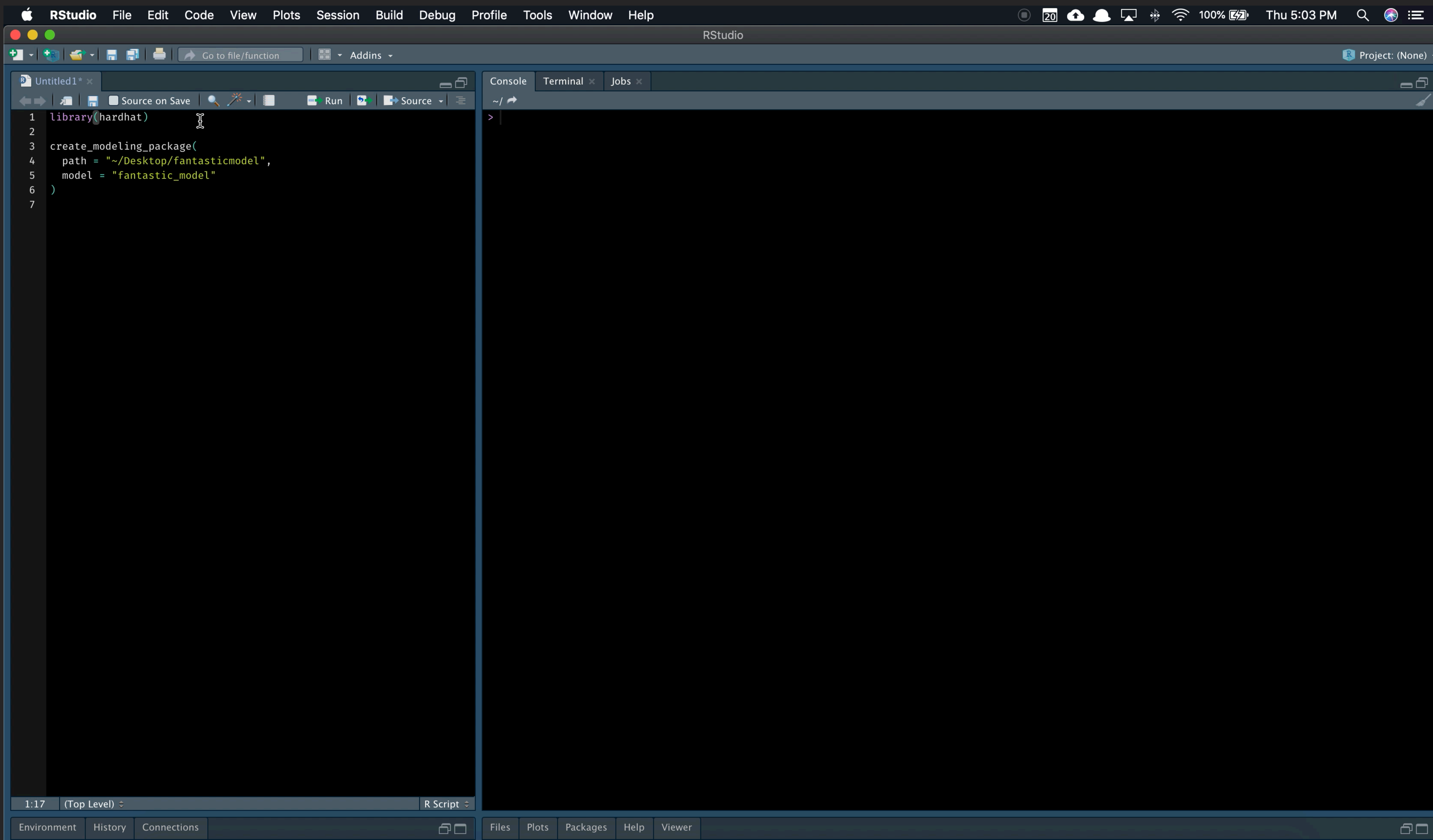
This one’s all you

But...

create_modeling_package()



create_modeling_package()



Demo

Model Fitting - High level interface

Example

```
earth::earth(x, y)
earth::earth(formula, data)
```

Recommendations

S3 methods: data.frame / matrix / formula / recipe

y: A vector or 1 column data frame / matrix

Consistent output: A model object!

1) Preprocess and standardize here!

2) Pass off to bridge function

Model Fitting - High level interface

Example

```
earth::earth(x, y)
earth::earth(formula, data)
```

Multivariate Adaptive
Regression Splines

Recommendations

S3 methods: data.frame / matrix / formula / recipe

y: A vector or 1 column data frame / matrix

Consistent output: A model object!

1) Preprocess and standardize here!

2) Pass off to bridge function

S3 Crash Course

```r
logistic_regression ← function(x, ... ) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix ← function(x, y, ... ) {
  # matrix method
}

logistic_regression.data.frame ← function(x, y, ... ) {
  # data frame method
}

logistic_regression.recipe ← function(x, data, ... ) {
  # recipe method
}

logistic_regression.formula ← function(formula, data, ... ) {
  # formula method
}
```

```r
logistic_regression <- function(x, ... ) {
  UseMethod("logistic_regression")
}
```

Generic

```r
logistic_regression.matrix <- function(x, y, ... ) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ... ) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ... ) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ... ) {
  # formula method
}
```

```
logistic_regression ← function(x, ... ) {
  UseMethod("logistic_regression")
}
```

Generic

```
logistic_regression.matrix ← function(x, y, ... ) {
  # matrix method
}

logistic_regression.data.frame ← function(x, y, ... ) {
  # data frame method
}

logistic_regression.recipe ← function(x, data, ... ) {
  # recipe method
}

logistic_regression.formula ← function(formula, data, ... ) {
  # formula method
}
```

Methods

```r
logistic_regression(<data.frame>, <factor>)
```

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

logistic_regression(<data.frame>, <factor>)

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```
logistic_regression(<data.frame>, <factor>)
```

1

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```
logistic_regression(<data.frame>, <factor>)
```

1

```
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```
logistic_regression(<data.frame>, <factor>)
```

1

```
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```r
logistic_regression(<data.frame>, <factor>)
```

1

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```
logistic_regression(<data.frame>, <factor>)
```

1

```
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}
```

```
logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}
```

2

```
logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

```
logistic_regression(<data.frame>, <factor>)
```

1

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}
```

```r
logistic_regression.matrix <- function(x, y, ...) {
  # matrix method
}
```

2

```r
logistic_regression.data.frame <- function(x, y, ...) {
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  # formula method
}
```

Fun fact

https://github.com/wch/r-source/blob/588b4c31ca3eb6dda585875514824737O7c03689/src/library/tools/R/QC.R#L2422

How does hardhat help?

data.frames, formulas,
and recipes, oh my!

mold()

```
predictors <- select(iris, -Species)
outcome <- pull(iris, Species)

processed <- mold(predictors, outcome)

processed$predictors
#> # A tibble: 150 x 4
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width
#>           <dbl>       <dbl>        <dbl>       <dbl>
#>  1          5.1         3.5          1.4         0.2
#>  2          4.9         3            1.4         0.2
#>  3          4.7         3.2          1.3         0.2
#>  4          4.6         3.1          1.5         0.2
#>  5          5           3.6          1.4         0.2
#>  6          5.4         3.9          1.7         0.4
#>  7          4.6         3.4          1.4         0.3
#>  8          5           3.4          1.5         0.2
#>  9          4.4         2.9          1.4         0.2
#> 10          4.9         3.1          1.5         0.1
#> # … with 140 more rows
```

mold()

```
predictors ← select(iris, -Species)
outcome ← pull(iris, Species)

processed ← mold(predictors, outcome)

processed$outcomes
#> # A tibble: 150 x 1
#>    .outcome
#>    <fct>
#>  1 setosa
#>  2 setosa
#>  3 setosa
#>  4 setosa
#>  5 setosa
#>  6 setosa
#>  7 setosa
#>  8 setosa
#>  9 setosa
#> 10 setosa
#> # … with 140 more rows
```

mold()

```
predictors ← select(iris, -Species)
outcome ← pull(iris, Species)

processed ← mold(predictors, outcome)

processed$blueprint
#> XY blueprint:
#>
#> # Predictors: 4
#>   # Outcomes: 1
#>    Intercept: FALSE
```

mold()

```r
processed <- mold(Species ~ Sepal.Width, iris)
```

```r
processed$predictors
#> # A tibble: 150 x 1
#>    Sepal.Width
#>          <dbl>
#>  1         3.5
#>  2         3  
#>  3         3.2
#>  4         3.1
#>  5         3.6
#>  6         3.9
#>  7         3.4
#>  8         3.4
#>  9         2.9
#> 10         3.1
#> # … with 140 more rows
```

```r
processed$outcomes
#> # A tibble: 150 x 1
#>    Species
#>    <fct>  
#>  1 setosa 
#>  2 setosa 
#>  3 setosa 
#>  4 setosa 
#>  5 setosa 
#>  6 setosa 
#>  7 setosa 
#>  8 setosa 
#>  9 setosa 
#> 10 setosa 
#> # … with 140 more rows
```

ok, cool

Preprocessing

What?

Can preprocess inline transforms:

```
Species ~ log(Sepal.Width)
```

Multivariate outcomes:

```
mpg + cyl ~  wt + gear
```

prep()s recipes for you!

How?

The blueprint!

blueprint

A blueprint defines how to preprocess data

```
mold(x, …, blueprint = NULL)
```

```
default_formula_blueprint()
default_xy_blueprint()
default_recipe_blueprint()
```

Tweaking defaults

```r
processed <- mold(Sepal.Width ~ Sepal.Length, iris)
```

```r
processed$predictors
#> # A tibble: 150 x 1
#>    Sepal.Length
#>           <dbl>
#>  1          5.1
#>  2          4.9
#>  3          4.7
#>  4          4.6
#>  5          5
#>  6          5.4
#>  7          4.6
#>  8          5
#>  9          4.4
#> 10          4.9
#> # … with 140 more rows
```

Tweaking defaults

```r
bp <- default_formula_blueprint(intercept = TRUE)
processed <- mold(Sepal.Width ~ Sepal.Length, iris, blueprint = bp)
```

```r
processed$predictors
#> # A tibble: 150 x 2
#>    `(Intercept)` Sepal.Length
#>            <dbl>        <dbl>
#>  1             1          5.1
#>  2             1          4.9
#>  3             1          4.7
#>  4             1          4.6
#>  5             1          5  
#>  6             1          5.4
#>  7             1          4.6
#>  8             1          5  
#>  9             1          4.4
#> 10             1          4.9
#> # … with 140 more rows
```

Tweaking defaults

```
bp <- default_formula_blueprint(intercept = TRUE)
processed <- mold(Sepal.Width ~ Species, iris, blueprint = bp)
```

```
processed$predictors
#> # A tibble: 150 x 3
#>    `(Intercept)` Speciesversicolor Speciesvirginica
#>            <dbl>             <dbl>            <dbl>
#>  1             1                 0                0
#>  2             1                 0                0
#>  3             1                 0                0
#>  4             1                 0                0
#>  5             1                 0                0
#>  6             1                 0                0
#>  7             1                 0                0
#>  8             1                 0                0
#>  9             1                 0                0
#> 10             1                 0                0
#> # … with 140 more rows
```

Tweaking defaults

```r
bp <- default_formula_blueprint(intercept = TRUE, indicators = FALSE)
processed <- mold(Sepal.Width ~ Species, iris, blueprint = bp)
```

```r
processed$predictors
#> # A tibble: 150 x 2
#>    `(Intercept)` Species
#>            <dbl> <fct>
#>  1             1 setosa
#>  2             1 setosa
#>  3             1 setosa
#>  4             1 setosa
#>  5             1 setosa
#>  6             1 setosa
#>  7             1 setosa
#>  8             1 setosa
#>  9             1 setosa
#> 10             1 setosa
#> # … with 140 more rows
```

mold() and you

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  processed <- mold(x, y)
  # matrix method
}

logistic_regression.data.frame <- function(x, y, ...) {
  processed <- mold(x, y)
  # data frame method
}

logistic_regression.recipe <- function(x, data, ...) {
  processed <- mold(x, data)
  # recipe method
}

logistic_regression.formula <- function(formula, data, ...) {
  processed <- mold(formula, data)
  # formula method
}
```

Demo

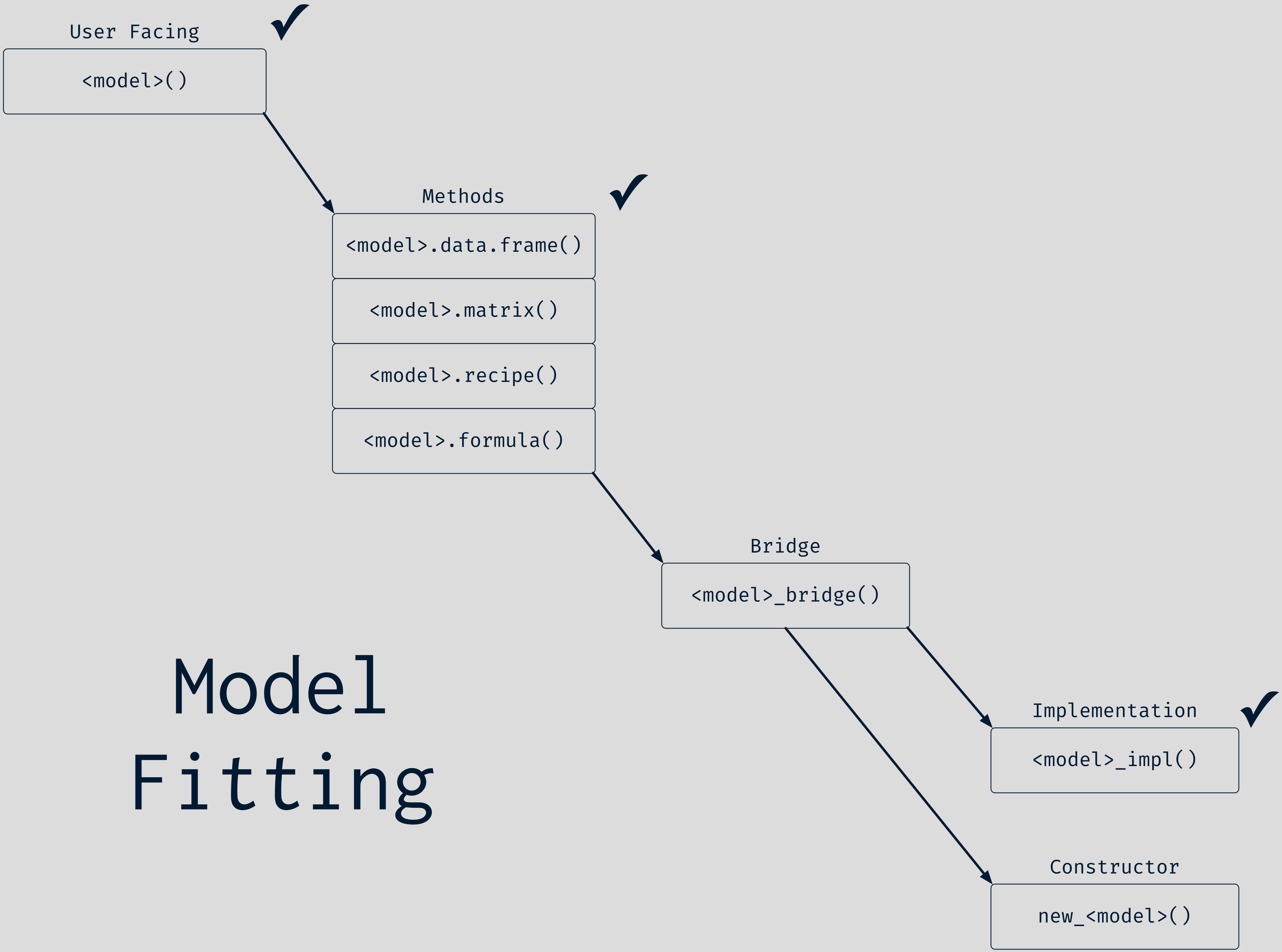
Why separate the high from the low?

Different interfaces = Different input types

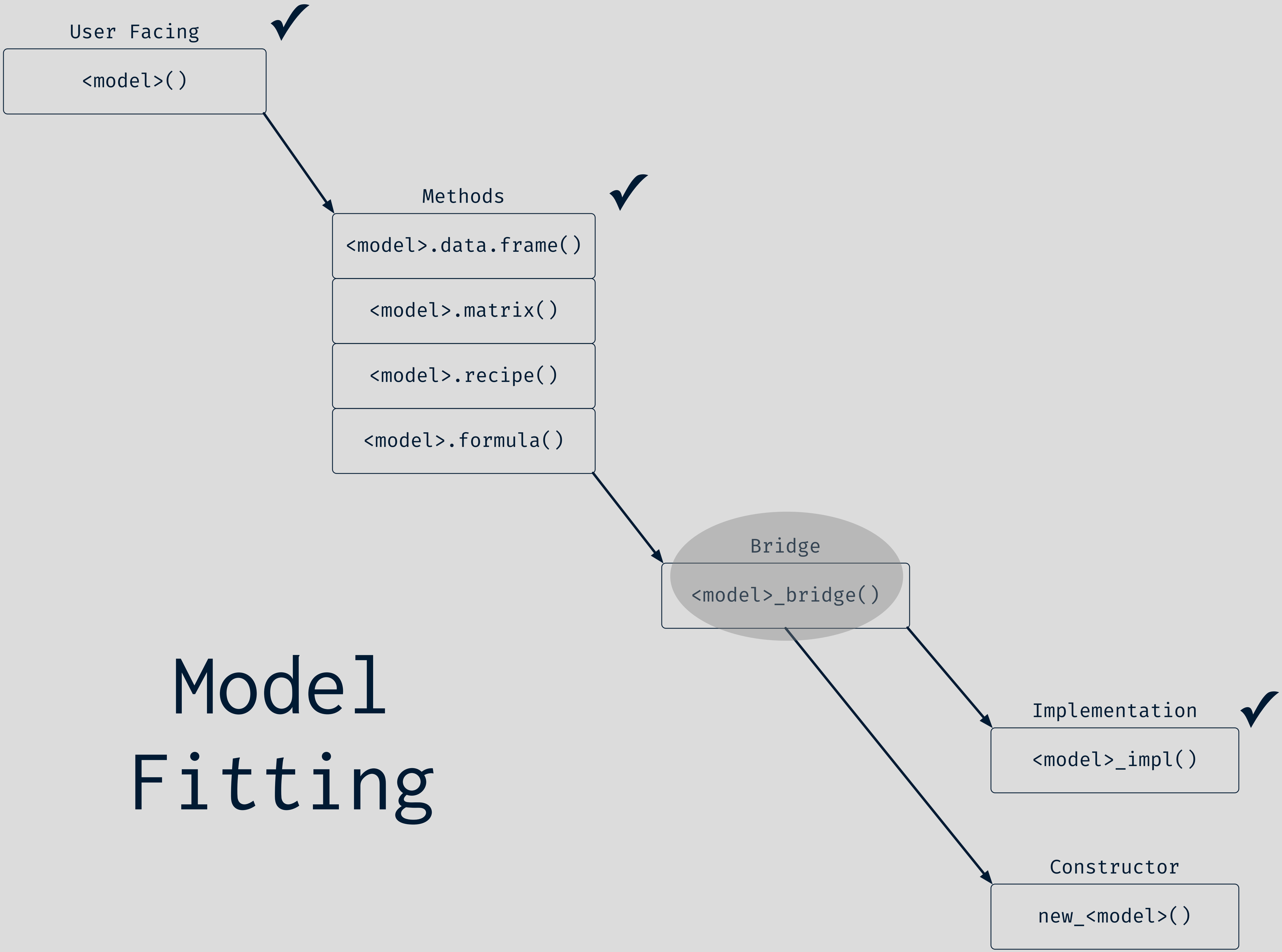
logistic_regression.default()?

Test implementation separate from interface

Model Fitting



Model Fitting



Model Fitting - Bridging the gap

Why?

High level interface <--> Low level implementation

Input types still need to be validated

Recommendations

Inputs: Preprocessed mold() output

Validation

Converts to low level matrices / vectors

Calls implementation function

Returns model object using the constructor

```r
logistic_regression_bridge <- function(processed, ...) {

  # Validate and process predictors
  predictors <- processed$predictors
  hardhat::validate_predictors_are_numeric(predictors)
  predictors <- as.matrix(predictors)

  # Validate and process outcomes
  outcome <- processed$outcomes
  hardhat::validate_outcomes_are_factors(outcome)
  hardhat::validate_outcomes_are_binary(outcome)
  hardhat::validate_outcomes_is_univariate(outcome)
  outcome <- outcome[[1]]

  # Fit the model
  fit <- logistic_regression_impl(predictors, outcome)

  # Constructor stuff
}
```

```r
logistic_regression_bridge ← function(processed, ...) {

  # Validate and process predictors
  predictors ← processed$predictors
  hardhat::validate_predictors_are_numeric(predictors)
  predictors ← as.matrix(predictors)

  # Validate and process outcomes
  outcome ← processed$outcomes
  hardhat::validate_outcomes_are_factors(outcome)
  hardhat::validate_outcomes_are_binary(outcome)
  hardhat::validate_outcomes_is_univariate(outcome)
  outcome ← outcome[[1]]

  # Fit the model
  fit ← logistic_regression_impl(predictors, outcome)

  # Constructor stuff
}
```

```r
logistic_regression_bridge <- function(processed, ...) {

  # Validate and process predictors
  predictors <- processed$predictors
  hardhat::validate_predictors_are_numeric(predictors)
  predictors <- as.matrix(predictors)

  # Validate and process outcomes
  outcome <- processed$outcomes
  hardhat::validate_outcomes_are_factors(outcome)
  hardhat::validate_outcomes_are_binary(outcome)
  hardhat::validate_outcomes_is_univariate(outcome)
  outcome <- outcome[[1]]

  # Fit the model
  fit <- logistic_regression_impl(predictors, outcome)

  # Constructor stuff
}
```

```r
logistic_regression <- function(x, ...) {
  UseMethod("logistic_regression")
}

logistic_regression.matrix <- function(x, y, ...) {
  processed <- mold(x, y)
  logistic_regression_bridge(processed, ...)
}

logistic_regression.data.frame <- function(x, y, ...) {
  processed <- mold(x, y)
  logistic_regression_bridge(processed, ...)
}

logistic_regression.recipe <- function(x, data, ...) {
  processed <- mold(x, data)
  logistic_regression_bridge(processed, ...)
}

logistic_regression.formula <- function(formula, data, ...) {
  processed <- mold(formula, data)
  logistic_regression_bridge(processed, ...)
}
```

Demo

Model Fitting - Constructors

Model Fitting - Constructors

Wat

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

Raw input

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Validate just
the type

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret ← function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Add attributes
to an object
structure(.Data, …)

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Add a class

Model Fitting - Constructors

Just a function!

Formalizes an S3 class

```
new_secret ← function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

Constructors - Vectors

```r
new_secret <- function(x = double()) {
  stopifnot(is.double(x))

  structure(
    x,
    class = "secret"
  )
}
```

```r
new_secret(55)
#> [1] 55
#> attr(,"class")
#> [1] "secret"
```

Constructors - Attributes

```r
new_secret <- function(x = double(),
                       name = character()) {
  stopifnot(is.double(x))
  stopifnot(is.character(name))

  structure(
    x,
    name = name,
    class = "secret"
  )
}
```

```r
new_secret(55, "bob")
#> [1] 55
#> attr(,"name")
#> [1] "bob"
#> attr(,"class")
#> [1] "secret"
```

Constructors - Inheritance

```r
new_secret <- function(x = double(),
                       name = character(),
                       ... ,
                       class = character()) {
  stopifnot(is.double(x))
  stopifnot(is.character(name))

  structure(
    x,
    name = name,
    ... ,
    class = c(class, “secret")
  )
}

new_secret(55, "bob", class = "super-secret")
#> [1] 55
#> attr(,"name")
#> [1] "bob"
#> attr(,"class")
#> [1] "super-secret" "secret"
```

Constructors - Scalars

```
new_lm <- function(coef = double(),
                   resid = double(),
                   ... ,
                   class = character()) {

  stopifnot(is.double(coef))
  stopifnot(is.double(resid))

  fields <- list(
    coef = coef,
    resid = resid,
    ...
  )

  structure(
    fields,
    class = c(class, “lm”)
  )
}
```

Constructors - Scalars

```
new_lm <- function(coef = double(),
                   resid = double(),
                   ... ,
                   class = character()) {

  stopifnot(is.double(coef))
  stopifnot(is.double(resid))

  hardhat::new_model(
    coef = coef,
    resid = resid,
    class = c(class, "lm")
  )
}
```

Constructors - Scalars

```
new_lm ← function(coef = double(),
                  resid = double(),
                  ... ,
                  blueprint = NULL,
                  class = character()) {

  stopifnot(is.double(coef))
  stopifnot(is.double(resid))

  hardhat::new_model(
    coef = coef,
    resid = resid,
    blueprint = blueprint,
    class = c(class, “lm”)
  )
}
```

Model Fitting - Constructors

Recommendations

Name: new_<model_object>()

Pass through the blueprint!

Inputs: Consider post fit methods - coef() / plot()

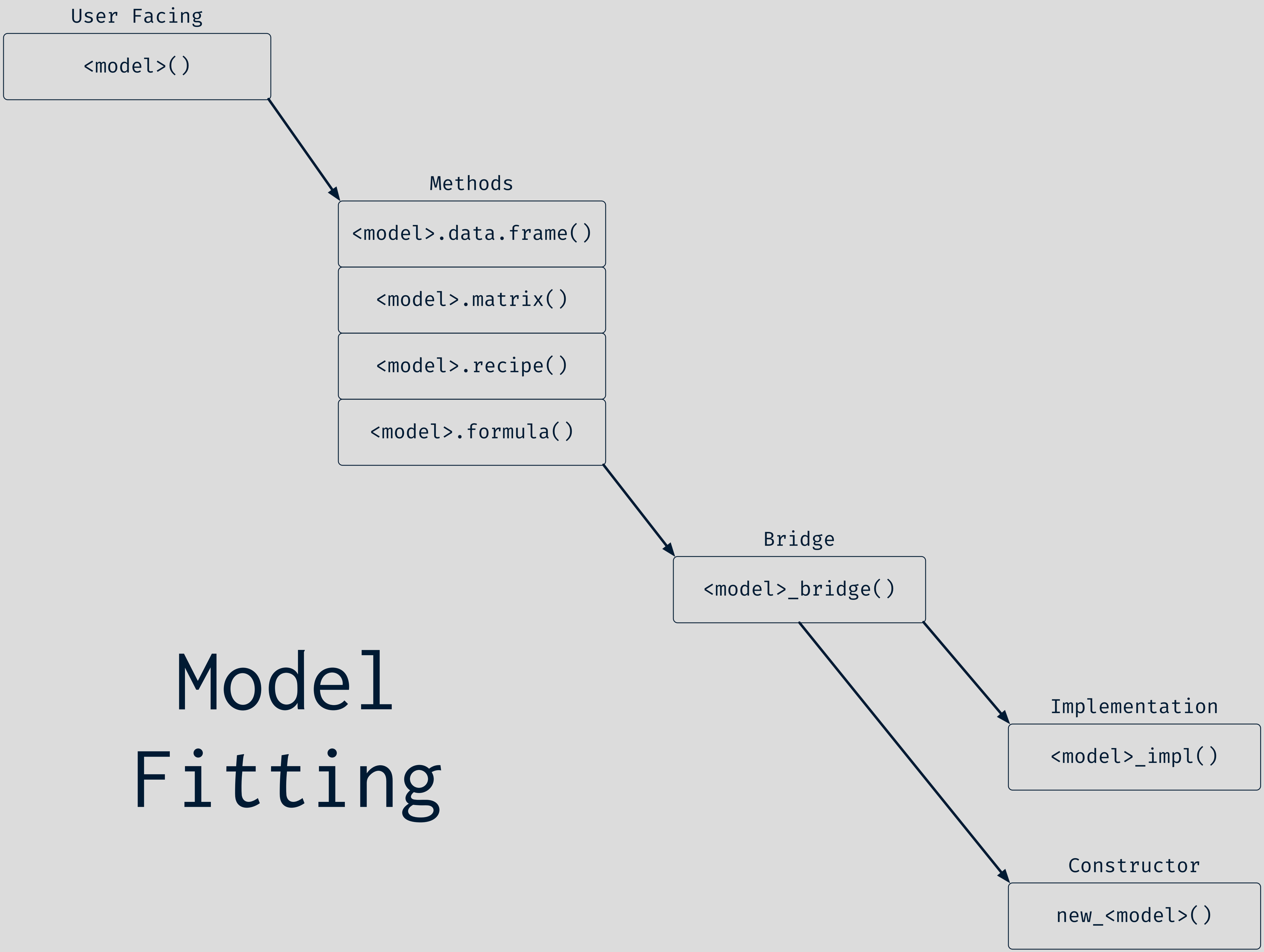
Generally favor recomputing > memory usage

🔥 Try not to store the terms

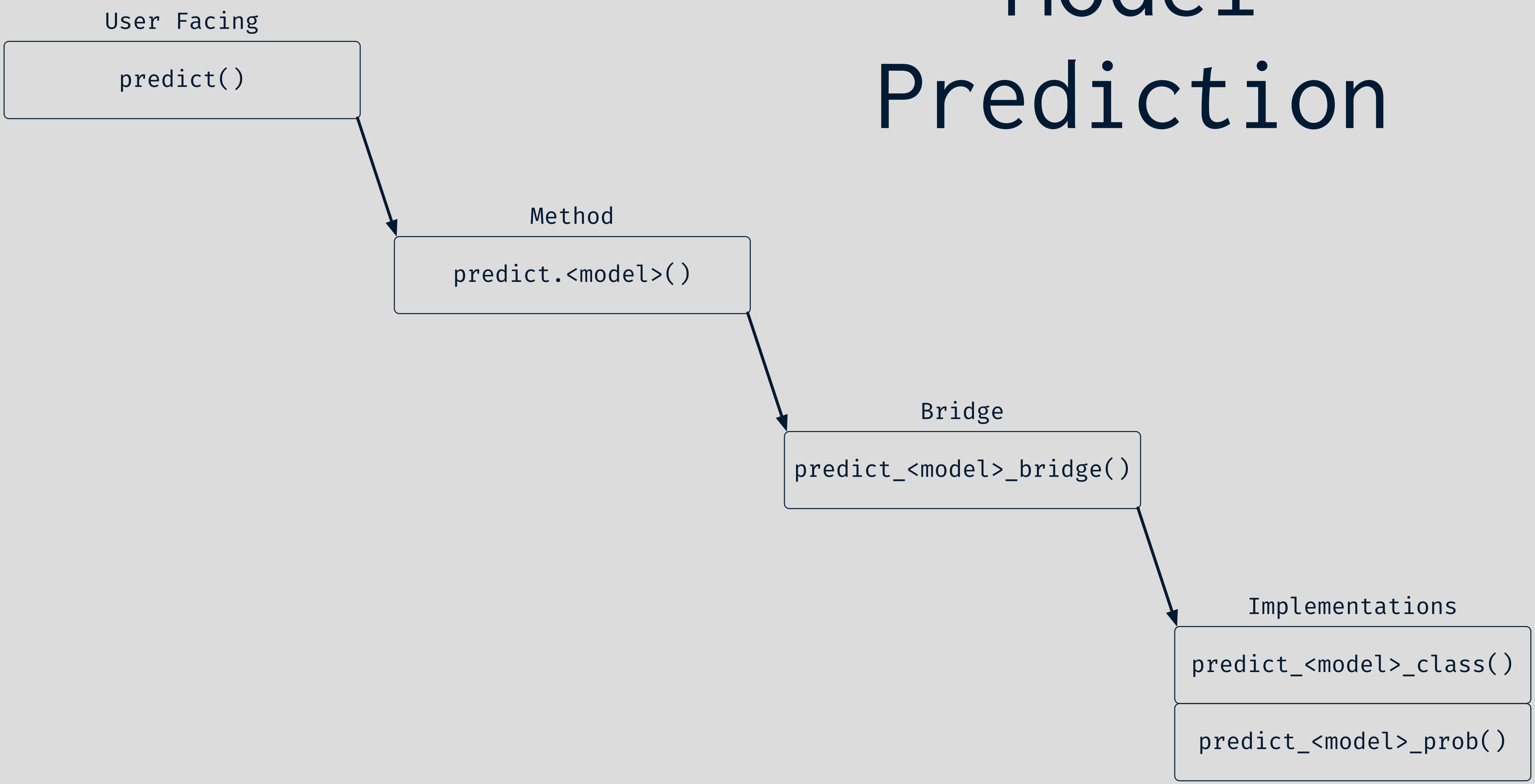
🔥 Try not to store the call

Most importantly: Do not store the training set

Demo


User Facing
<model>()
Methods
<model>.data.frame()
<model>.matrix()
<model>.recipe()
<model>.formula()
Bridge
<model>_bridge()
Implementation
<model>_impl()
Constructor
new_<model>()
Model
Fitting

Model Prediction



Model Prediction - High level interface

Example

```
earth::predict.earth(object, newdata, type)
```

object = earth model

newdata = data frame, matrix, NULL

```
type = c("link", "response", "earth",
         "class", "terms")
```

Model Prediction - High level interface

Example

```
earth::predict.earth(object, newdata, type)
```

object = earth model

Returns stored fitted values

newdata = data frame, matrix, NULL

type = c("link", "response", "earth",
"class", "terms")

Model Prediction - High level interface

Example

```
earth::predict.earth(object, newdata, type)
```

object = earth model

newdata = data frame, matrix, NULL

type = c("link", "response", "earth",
         "class", "terms")

Model Prediction - High level interface

Example

```
earth::predict.earth(object, newdata, type)
```

object = earth model

newdata = data frame, matrix, NULL

type = c("link", "response", "earth",
         "class", "terms")

Return vectors or matrices

Model Prediction - High level interface

Example

```
earth::predict.earth(object, newdata, type)
```

object = earth model

newdata = data frame, matrix, NULL

type = c("link", "response", "earth",
         "class", "terms")

Model Prediction - High level interface

Recommendations

Convention: new_data

Convention: Standardized type arguments

https://tidymodels.github.io/model-implementation-principles/model-predictions.html

Input: new_data can be a data frame / matrix

Output: A data frame (implementation detail)

Model Prediction - High level interface

Job

(Hint: Same as the fit interface)

Preprocess new_data

Hand off to the bridge

How does hardhat help?

Preprocessing

The blueprint returns!

Some restrictions on new_data:

- Formula / recipe preprocessing must be reapplied

- Columns must have the same name and type

Preprocessing

The blueprint returns!

Some restrictions on new_data:

- Formula / recipe preprocessing must be reapplied

- Columns must have the same name and type

Preprocessing

The blueprint returns!

Some restrictions on new_data:

- Formula / recipe preprocessing must be reapplied
- Columns must have the same name and type
  - Factors must have the same levels
  - Column classes must be the same

forge()

```
train <- iris[1:100,]
test <- iris[101:150,]

processed <- mold(Sepal.Length ~ Species + log(Sepal.Width), train)

forged <- forge(test, processed$blueprint)
forged$predictors
#> # A tibble: 50 x 4
#>    Speciessetosa Speciesversicolor Speciesvirginica `log(Sepal.Width)`
#>            <dbl>             <dbl>            <dbl>              <dbl>
#>  1             0                 0                1              1.19
#>  2             0                 0                1              0.993
#>  3             0                 0                1              1.10
#>  4             0                 0                1              1.06
#>  5             0                 0                1              1.10
#>  6             0                 0                1              1.10
#>  7             0                 0                1              0.916
#>  8             0                 0                1              1.06
#>  9             0                 0                1              0.916
#> 10             0                 0                1              1.28
#> # … with 40 more rows
```

Validation - Novel / Missing levels

```
test_lvls ← test
test_lvls ← mutate(test_lvls, Species = as.character(Species))
test_lvls$Species[1] ← "extra"
test_lvls ← mutate(test_lvls, Species = as.factor(Species))

levels(test_lvls$Species)
#> [1] "extra"     "virginica"

forged ← forge(test_lvls, processed$blueprint)
#> Warning: Novel levels found in column 'Species': 'extra'. The levels have
#> been removed, and values have been coerced to 'NA'.

forged$predictors
#> # A tibble: 50 x 4
#>    Speciessetosa Speciesversicolor Speciesvirginica `log(Sepal.Width)`
#>            <dbl>             <dbl>            <dbl>              <dbl>
#>  1            NA                NA               NA              1.19
#>  2             0                 0                1              0.993
#>  3             0                 0                1              1.10
#>  4             0                 0                1              1.06
#>  5             0                 0                1              1.10
#>  6             0                 0                1              1.10
#>  7             0                 0                1              0.916
#>  8             0                 0                1              1.06
#>  9             0                 0                1              0.916
#> 10             0                 0                1              1.28
#> # … with 40 more rows
```

Existing tooling

```
# fitting
frame ← model.frame(Sepal.Length ~ Species + log(Sepal.Width), train)
terms ← delete.response(terms(frame))
orig_lvls ← .getXlevels(terms, frame)
model_obj ← list(stuff = 1, terms = terms, orig_lvls = orig_lvls)

# prediction
test_frame ← model.frame(model_obj$terms, test_lvls)
head(model.matrix(model_obj$terms, test_frame))
#>   (Intercept) Speciesvirginica log(Sepal.Width)
#> 1           1                0        1.1939225
#> 2           1                1        0.9932518
#> 3           1                1        1.0986123
#> 4           1                1        1.0647107
#> 5           1                1        1.0986123
#> 6           1                1        1.0986123

# using xlev
model.frame(model_obj$terms, test_lvls, xlev = model_obj$orig_lvls)
#> Error in model.frame.default(model_obj$terms, test_lvls, xlev = model_obj$orig_lvls):
factor Species has new levels extra
```

Validation - Wrong class

```r
train <- data.frame(date = Sys.Date() + 1:100, y = 1:100)
test <- data.frame(date = 1:5)

processed <- mold(y ~ date, train)
forge(test, processed$blueprint)
#> Can't cast `x$date` <integer> to `to$date` <date>.

terms <- delete.response(terms(model.frame(y ~ date, train)))
model.matrix(terms, model.frame(terms, test))
#>   (Intercept) date
#> 1           1    1
#> 2           1    2
#> 3           1    3
#> 4           1    4
#> 5           1    5
#> attr(,"assign")
#> [1] 0 1

attr(terms, "dataClasses")
#>         y      date
#> "numeric"   "other"
```

predict()

```
predict.logistic_regression <- function(object, new_data,
                                        type = "prob", ... ) {

  forged <- hardhat::forge(new_data, object$blueprint)

  # bridge function
}
```

Demo

Prediction return value

Prediction return value

What do people do with predictions?

Prediction return value

What do people do with predictions?

Attach them to a new_data data frame

Prediction return value

What do people do with predictions?

Attach them to a new_data data frame

How can the result be predictable (type-stable)?

Prediction return value

What do people do with predictions?

Attach them to a new_data data frame

How can the result be predictable (type-stable)?

We need to always return the same data structure

Prediction return value

What do people do with predictions?

Attach them to a new_data data frame

How can the result be predictable (type-stable)?

We need to always return the same data structure

What data structure covers the most use cases?

Prediction return value

What do people do with predictions?

Attach them to a new_data data frame

How can the result be predictable (type-stable)?

We need to always return the same data structure

What data structure covers the most use cases?

A data frame!

Model Prediction - Low level implementation(s)

Structure

```
predict(my_model, new_data, type = c("class", "prob", …))

predict_my_model_class()
predict_my_model_prob()
```

Recommendations

One implementation per type

Return a data frame

🔥 # rows of output = # rows of new_data

Conventions: Column names!

Model Prediction - Low level implementation(s)

Structure

```
predict(my_model, new_data, type = c("class", "prob", …))

predict_my_model_class()
predict_my_model_prob()
```

type	convention
"numeric"	.pred
"class"	.pred_class
"prob"	.pred_{level}

Recommendations

One implementation per type

Return a data frame

🔥 # rows of output = # rows of new_data

Conventions: Column names!

spruce_*()

```
# From the blueprint
# levels(model$blueprint$ptypes$outcomes[[1]])
pred_levels ← c("a", "b")

# You compute this
pred_matrix ← matrix(1:6, ncol = 2)

spruce_prob(pred_levels, pred_matrix)
#> # A tibble: 3 x 2
#>   .pred_a .pred_b
#>     <int>   <int>
#> 1       1       4
#> 2       2       5
#> 3       3       6
```

Demo

Model Prediction - Bridging the gap

Notes

High level interface <--> Low level implementation

Recommendations

Assumes preprocessed new_data

Converts new_data to low level type (matrix)

Call implementation function

Validate output size

Demo

The design of a modeling 📦

Model Fitting

✓ High level user interface

✓ Low level fit implementation

✓ Bridge

✓ Model constructor

Model Prediction

✓ High level predict() method

✓ Low level type specific predict implementations

✓ Bridge