# Applied tidymodeling

Josiah Parry

RStudio, Inc.

updated: 2020-10-22

# Applied tidymodeling

# Modeling in the Tidyverse

What we will cover:

- A brief introduction to tidymodeling
- Productionizing models with plumber
- Deploying and hosting with connect

The end products:

- https://colorado.rstudio.com/rsc/genre-pred/
- https://colorado.rstudio.com/rsc/hiphop-or-country/

# What is tidymodels?

```
library(tidymodels)
```

```
## Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_fr
## when loading 'dplyr'
```

```
## ── Attaching packages ──────────────────────────────────────────────────────
```

```
## ✓ broom     0.5.5      ✓ recipes   0.1.10
## ✓ dials     0.0.6      ✓ rsample   0.0.6
## ✓ dplyr     1.0.0      ✓ tibble    3.0.1
## ✓ ggplot2   3.3.0      ✓ tune      0.1.0
## ✓ infer     0.5.1      ✓ workflows 0.1.1
## ✓ parsnip   0.1.0      ✓ yardstick 0.0.6
## ✓ purrr     0.3.4
```

```
## ── Conflicts ─────────────────────────────────────────────────────── tidym
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x ggplot2::margin() masks dials::margin()
## x recipes::step()   masks stats::step()
```

# tidyverse paradigmn

- `%>%` the pipe

```
mean(1:10)
```

```
## [1] 5.5
```

```
1:10 %>%
  mean()
```

```
## [1] 5.5
```

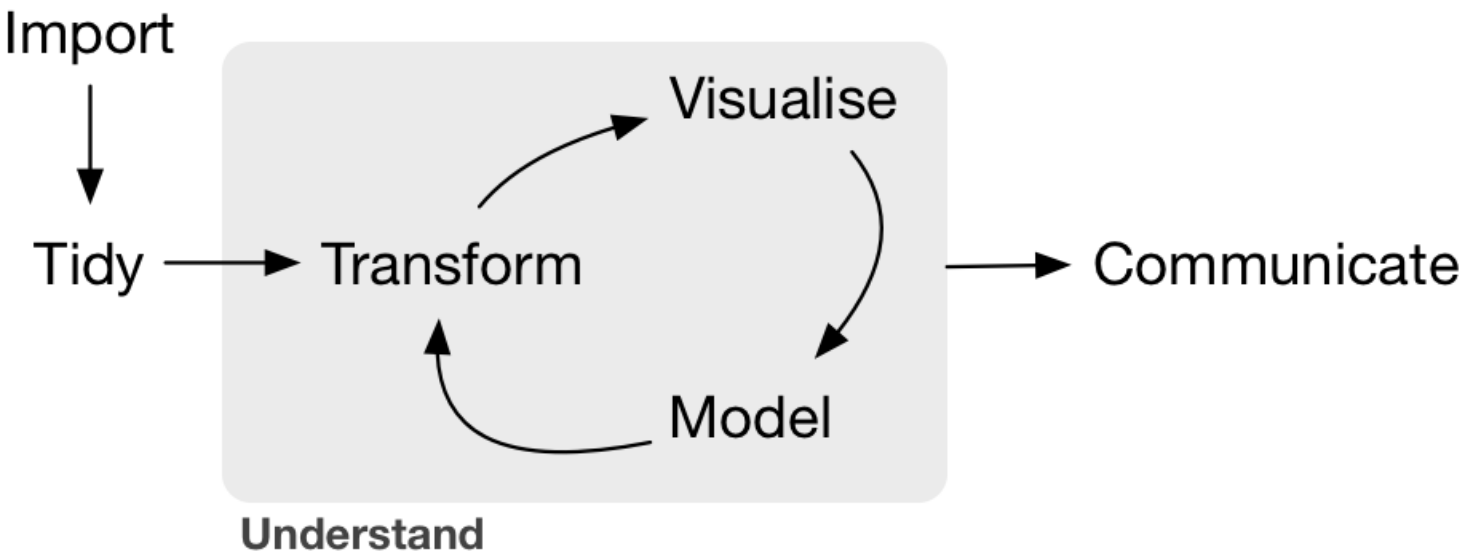
# The Pipe %>%

```
data %>%
  do_something(.) %>%
  do_another_thing(.) %>%
  do_last_thing(.)
```

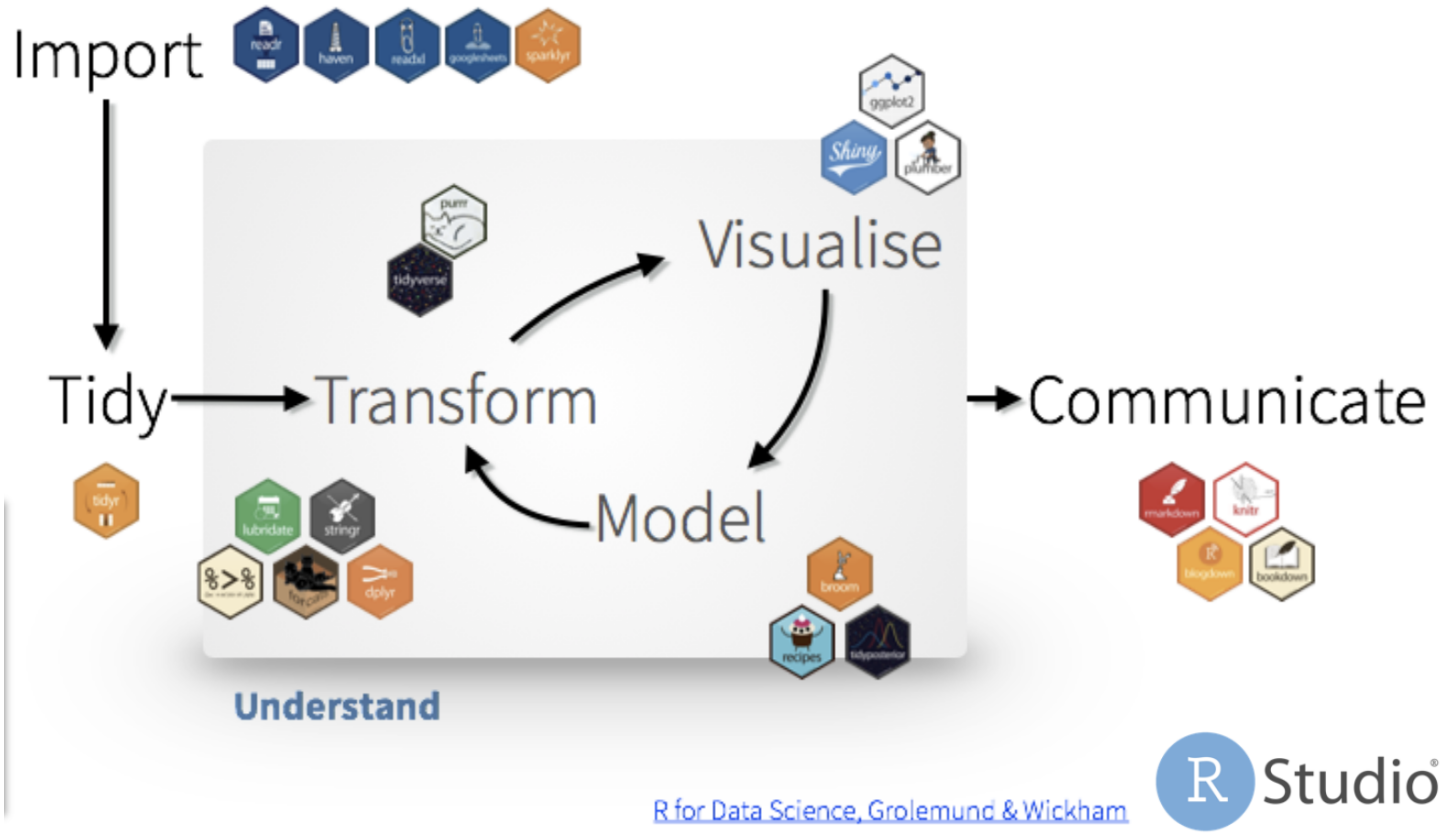
`do_something(data)` is equivalent to:

- `data %>% do_something(data = .)`
- `data %>% do_something(.)`
- `data %>% do_something()`

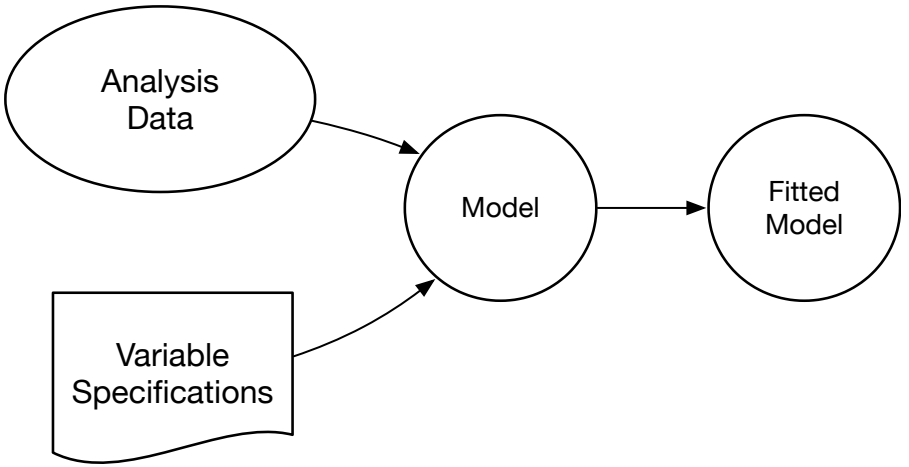
# Analysis Workflow



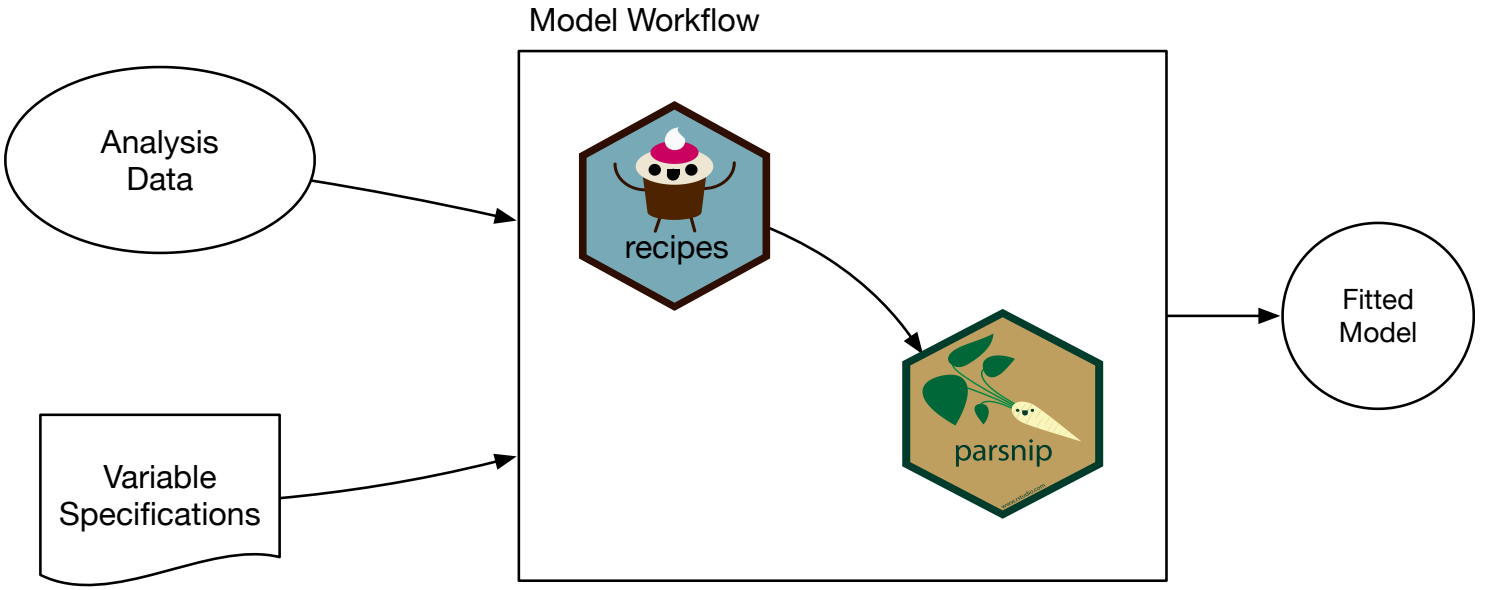
# Tidyverse Workflow



# Modeling Workflow



# Tidymodels workflow



# Motivation: Predicting Class Probabilities

| Function | Package | Code |
|---|---|---|
| lda | MASS | predict(obj) |
| glm | stats | predict(obj, type = "response") |
| rpart | rpart | predict(obj, type = "prob") |
| logitboost | LogitBoost | predict(obj, type = "raw", nIter) |

# parsnip

# parsnip

# parsnip

- General interface for modeling
- specifications:
  - model
  - engine
  - fit
- models

# Example

```
# model
iris_fit <- decision_tree(mode =  "classification") %>%
  # engine
  set_engine("rpart") %>%
  # fit
  fit(Species ~ ., data = iris)
```

# Changing engines

```
# model
iris_fit <- decision_tree(mode =  "classification") %>%
  # engine
  set_engine("C5.0") %>%
  # fit
  fit(Species ~ ., data = iris)

iris_fit
```

# recipes

# recipes

- preprocessing interface
- dplyr-like syntax
- tidyselect-like syntax

# Defining our `recipe()`

- Our recipe is the plan of action
- We can add `step_*()`s to our recipe

```
iris_rec <- recipe(Species ~ ., data = iris)

iris_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome           1
##  predictor           4
```

# preprocessing steps

- pre-processing steps are specified with the `step_*()` functions
- Some of which are:
  - `step_center()`
  - `step_scale()`
  - `step_log()`
- Check reference documentation

# preprocessing steps

- `dplyr`-like syntax:
  - `all_predictors()`
  - `all_outcomes()`
  - `has_type()`
  - `all_numeric()`
  - `all_nominal()`

# Example:

```
iris_steps <- iris_rec %>%
  step_center(all_numeric()) %>%
  step_scale(all_predictors())

iris_steps
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome           1
##  predictor           4
##
## Operations:
##
## Centering for all_numeric
## Scaling for all_predictors
```

# Prepping our `recipe`

- We `prep()` our recipe when we are done specifying the preprocessing steps
- This prepped recipe can be used to preprocess new data

# Prepping our `recipe`

```
prepped <- prep(iris_steps)

prepped
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome           1
##  predictor           4
##
## Training data contained 150 data points and no missing data.
##
## Operations:
##
## Centering for Sepal.Length, Sepal.Width, ... [trained]
## Scaling for Sepal.Length, Sepal.Width, ... [trained]
```

# Preprocessing new data

- We `bake()` our recipe and our ingredients (new data)
- syntax: `bake(prepped_recipe, new_data)`

```
bake(prepped, head(iris))
```

```
## # A tibble: 6 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1       -0.898      1.02          -1.34       -1.31 setosa
## 2       -1.14      -0.132         -1.34       -1.31 setosa
## 3       -1.38       0.327         -1.39       -1.31 setosa
## 4       -1.50       0.0979        -1.28       -1.31 setosa
## 5       -1.02       1.25          -1.34       -1.31 setosa
## 6       -0.535      1.93          -1.17       -1.05 setosa
```

# Partitioning - `rsample`

# rsample

- We want to follow the train and test split
- three key functions:
  - `initial_split(data, prop, strata)`
  - strata - used for stratified sampling
  - `training()`
  - `testing()`

# rsample

```
init_split <- initial_time_split(iris, prop = 2/3, strata = Sepcies)

init_split
```

```
## <Training/Validation/Total>
## <100/50/150>
```

# rsample

```
training(init_split) %>%
  slice(1:10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
```

# rsample

```
testing(init_split) %>%
  slice(1:10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1           6.3         3.3          6.0         2.5 virginica
## 2           5.8         2.7          5.1         1.9 virginica
## 3           7.1         3.0          5.9         2.1 virginica
## 4           6.3         2.9          5.6         1.8 virginica
## 5           6.5         3.0          5.8         2.2 virginica
## 6           7.6         3.0          6.6         2.1 virginica
## 7           4.9         2.5          4.5         1.7 virginica
## 8           7.3         2.9          6.3         1.8 virginica
## 9           6.7         2.5          5.8         1.8 virginica
## 10          7.2         3.6          6.1         2.5 virginica
```

# Model Evalutation: `yardstick`

# yardstick

- A package for evaluating models
- Predictions are returned in a `tibble`
- General interface permits easy comparisons

# prediction

- `type = "class"`

```
predict(iris_fit, iris[1:5,], type = "class")
```

```
## # A tibble: 5 x 1
##   .pred_class
##   <fct>
## 1 setosa
## 2 setosa
## 3 setosa
## 4 setosa
## 5 setosa
```

# prediction

- `type = "prob"`

```
predict(iris_fit, iris[1:5,], type = "prob")
```

```
## # A tibble: 5 x 3
##   .pred_setosa .pred_versicolor .pred_virginica
##          <dbl>            <dbl>           <dbl>
## 1            1                0               0
## 2            1                0               0
## 3            1                0               0
## 4            1                0               0
## 5            1                0               0
```

# other prediction types:

- `conf_int`
- `pred_int`
- `quantile`
- `numeric`
- `raw`

# model `metrics()`

- `metrics()`: provides common performance estimates
- `metric_set()`: used to define custom model metrics

# Metrics example

```
preds <- predict(iris_fit, iris[1:5,]) %>%
  bind_cols(iris[1:5,])

metrics(preds, Species, .pred_class)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass         1
## 2 kap      multiclass       NaN
```

# Our own metric set

```
pref_metrics <- metric_set(accuracy, spec, sens, f_meas)

pref_metrics(two_class_example, truth = truth, estimate = predicted)
```

```
## # A tibble: 4 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.838
## 2 spec     binary         0.793
## 3 sens     binary         0.880
## 4 f_meas   binary         0.849
```

# Putting it together

Code walk through

# very brief intro to Plumber

# Plumber

- Create REST API
- simple code "decorations" (roxygen-like)

# Structure `plumber.R`

- supports:
  - `@get`
  - `@post`
  - `@put`
  - `@delete`
  - `@head`

```
#* @apiTitle engaging title


#* description of endpoint's utility
#* @param param_name useful param description
#* @get /param_endpoint
```

# Endpoint Example

```
#* Retrieve lyrics for a single song
#* @param artist name of the artist
#* @param song the name of the song / track
#* @get /track
function(artist, song) {
  genius::genius_lyrics(artist, song)
}
```

# Launch the API

```
pr <- plumb("plumber.R")
pr$run()
```