An introduction to tidymodels

Davis Vaughan

Goals for today

~~Horrify~~ Teach you

- Point out weaknesses in the current state of affairs
- Provide motivation for tidymodels

Encourage you

- Walk through a "typical" tidymodels analysis

Challenge you

- To try an analysis with tidymodels
- To create consistent R packages using our tools

tidymodels.org

Tidy Modeling with R (tmwr.org)

Modeling in R

- R has always had a rich set of modeling tools that it inherited from S. For example, the formula interface has made it simple to specify potentially complex model structures.
- *R has cutting edge models*. Many researchers in various domains use R as their primary computing environment and their work often results in R packages.
- *It is easy to port or link to other applications*. R doesn't try to be everything to everyone.

However, there is a huge *consistency problem*. For example:

- There are two primary methods for specifying what terms are in a model. Not all models have both.
- 99% of model functions automatically generate dummy variables.
- Many package developers come from other languages (i.e. C) and omit common R idioms.

Between-Package Inconsistency

Syntax for computing predicted class probabilities:

Function	Package	Code
lda	MASS	predict(obj)
glm	stats	predict(obj, type = "response")
gbm	gbm	predict(obj, type = "response", n.trees)
mda	mda	predict(obj, type = "posterior")
rpart	rpart	predict(obj, type = "prob")
Weka	RWeka	predict(obj, type = "probability")
logitboost	LogitBoost	predict(obj, type = "raw", nIter)
pamr.train	pamr	pamr.predict(obj, type = "posterior")

Even tougher are the *within-package* inconsistencies. glmnet and survival are two cases where we have had significant trouble handling within-package issues.

What We Need

Unless you are doing a simple one-off data analysis, the lack of consistency between, and sometimes within, R packages can be very frustrating.

If we could agree on a set of common conventions for interfaces, return values, and other components, everyone's life would be easier.

Once we agree on conventions, **two challenges** are:

- As of Oct 2021, there are over 18K R packages on CRAN. How do we "harmonize" these without breaking everything?
- How can we guide new R users (or people unfamiliar with R) in making good choices as they develop new modeling packages?

Our job is to reduce the pain points of modeling with R

"Modeling" includes everything from classical statistical methods to machine learning

Tidyverse

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

The principles of the tidyverse:

1. Reuse existing data structures.
2. Compose simple functions with the pipe.
3. Embrace functional programming.
4. Design for humans.

Tidymodels

`tidymodels` is a collection of modeling packages that live in the tidyverse that are guided by the same design principles.

Our goals for tidymodels are:

1. Encourage empirical validation and good methodology.

2. Smooth out diverse interfaces.

3. Build highly reusable infrastructure.

4. Enable a wider variety and combination of methodologies.

The `tidymodels` packages address the *retrospective* issues. We have also developed a set of principles and templates to solve the *prospective* issue by making it easy to create new modeling package that follow best practices.


applicable
Baguette
broom
butcher
dials
discrim
ember
finetune
hardhat
parsnip
recipes
rsample
RULES
spatialsample
stacks
TEXT RECIPES
tidymodels
tidyposterior
TUNE
%>%
WORKFLOWS
yardstick

Selected Modeling Packages

- `recipes` is a general data preprocessor with a modern interface. It can create model matrices that incorporate feature engineering, imputation, and other tools.
- `rsample` has infrastructure for *resampling* data so that models can be assessed and empirically validated.
- `parsnip` gives us a unified modeling interface.
- `tune` has functions for grid search and sequential optimization of model parameters.

Loading the Meta-Package


```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'tune':
##   method                from
##   required_pkgs.model_spec parsnip
```

```
## ── Attaching packages ──────────────────────────────────── tidymodels 0.1.4 ──
```

```
## ✓ broom        0.7.9          ✓ recipes      0.1.17
## ✓ dials        0.0.10.9000    ✓ rsample      0.1.0
## ✓ dplyr        1.0.7          ✓ tibble       3.1.6.9000
## ✓ ggplot2      3.3.5          ✓ tidyr        1.1.4
## ✓ infer        1.0.0          ✓ tune         0.1.6.9000
## ✓ modeldata    0.1.1          ✓ workflows    0.2.3.9000
## ✓ parsnip      0.1.7.900      ✓ workflowsets 0.1.0.9000
## ✓ purrr        0.3.4          ✓ yardstick    0.0.8
```

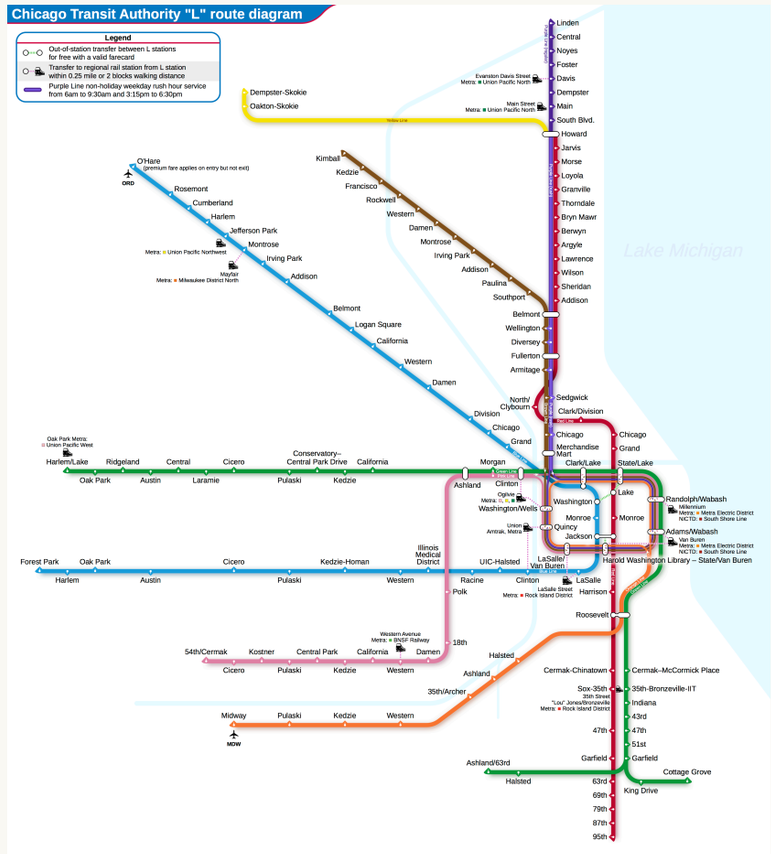
```
## ── Conflicts ─────────────────────────────────────── tidymodels_conflicts() ──
## x dplyr::collapse()  masks glue::collapse()
## x purrr::discard()   masks scales::discard()
## x tidyr::extract()   masks magrittr::extract()
## x purrr::set_names() masks magrittr::set_names()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

Our problem

Let's start by predicting the *ridership* of the Chicago "L" trains.

We have data over 5,698 days between 2001 and 2016 in `data(Chicago, package = "modeldata")`.

What are our predictors? Date, weather data, home game schedules, 14-day lags at other stations.



Our data

```
select(Chicago, ridership, date, Bulls_Home, wind, temp)
```

```
## # A tibble: 5,698 × 5
##    ridership date       Bulls_Home  wind  temp
##        <dbl> <date>          <dbl> <dbl> <dbl>
##  1     15.7  2001-01-22          0   5.2  19.4
##  2     15.8  2001-01-23          1   8.1  30.4
##  3     15.9  2001-01-24          0  10.4  25
##  4     15.9  2001-01-25          0   9.8  22.4
##  5     15.4  2001-01-26          0  12.7  27
##  6      2.42 2001-01-27          1  12.7  24.8
##  7      1.47 2001-01-28          0   8.1  18
##  8     15.5  2001-01-29          0   8.1  32
##  9     15.9  2001-01-30          0   9.2  37.4
## 10     15.9  2001-01-31          0  11.5  34
## # … with 5,688 more rows
```

Our steps

- Identify existing and compute new features to use in the model
- Identify which *type* of model to use, and what *engine* to run it with
- Bundle the above two ideas into a *workflow*
- Fit the workflow once
- Fit the workflow multiple times using *resamples*
- Tune the workflow

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago)
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year"))
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date)
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id")
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors())
```

Other selectors are:

- `all_predictors()`, `all_numeric_predictors()`, and `all_outcomes()`
- `has_type()` and `has_role()`
- Standard `dplyr` selectors like `starts_with()` and so on.

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

#?  step_mutate(temp = (32 * temp - 32) * 5 / 9 )
```

What are our *features*?


```
chicago_rec <- recipe(ridership ~ ., data = Chicago) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

```
chicago_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##         id          1
##    outcome          1
##  predictor         48
##
## Operations:
##
## Date features from date
## Holiday features from date
## Dummy variables from all_nominal_predictors()
## Centering and scaling for all_numeric_predictors()
```

Linear regression specification

"Let's fit a model with a numeric outcome, an intercept, and slopes for each predictor."

```
linear_mod <- linear_reg()
linear_mod
```

```
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

- Other model types include `nearest_neighbors()`, `decision_tree()`, `rand_forest()`, `arima_reg()`, and so on.
- An additional `set_engine()` function lets you adjust the *engine* (backend) to fit with.

Let's fit it with...

```
linear_mod <- linear_reg() %>%
  set_engine("lm")
```

Let's fit it with...

```
linear_mod <- linear_reg() %>%
  set_engine("keras")
```

Let's fit it with...

```
linear_mod <- linear_reg() %>%
  set_engine("spark")
```

Let's fit it with...

```
linear_mod <- linear_reg() %>%
  set_engine("stan")
```

Let's fit it with...

```
linear_mod <- linear_reg() %>%
  set_engine("glmnet")
```

Let's fit it with...


```
linear_mod <- linear_reg(penalty = 0.1, mixture = 0.5) %>%
  set_engine("glmnet")
```

A modeling *workflow*

We can *optionally* bundle the recipe and model together into a *workflow*:

```
glmnet_wflow <- workflow() %>%
  add_recipe(chicago_rec) %>% # or add_formula() or add_variables()
  add_model(linear_mod)
```

Fitting and prediction are very easy:

```
chicago_train <- slice(Chicago, seq(1L, n() - 4))
chicago_test <- slice(Chicago, seq(n() - 3, n()))

glmnet_fit <- fit(glmnet_wflow, data = chicago_train)
predict(glmnet_fit, chicago_test)
```

```
## # A tibble: 4 × 1
##   .pred
##   <dbl>
## 1 21.0
## 2 19.7
## 3  7.88
## 4  7.57
```

A modeling *workflow*


```
glmnet_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 4 Recipe Steps
##
## • step_date()
## • step_holiday()
## • step_dummy()
## • step_normalize()
##
## -- Model -----------------------------------------------------------------------
##
## Call:  glmnet::glmnet(x = maybe_matrix(x), y = y, family = "gaussian",      alpha = ~0.5)
##
##    Df %Dev Lambda
## 1   0  0.0  11.60
## 2   6  9.6  10.60
## 3  10 20.2   9.63
## 4  11 29.4   8.77
## 5  15 37.1   7.99
## 6  16 43.7   7.28
## 7  17 49.2   6.64
```

Model tuning

We probably don't have a good idea for what `penalty` and `mixture` should be

We can *mark them for tuning*:

```
linear_mod <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

glmnet_wflow <- workflow() %>%
  add_recipe(chicago_rec) %>%
  add_model(linear_mod)
```

Recipe arguments can also be simultaneously tuned, i.e.:

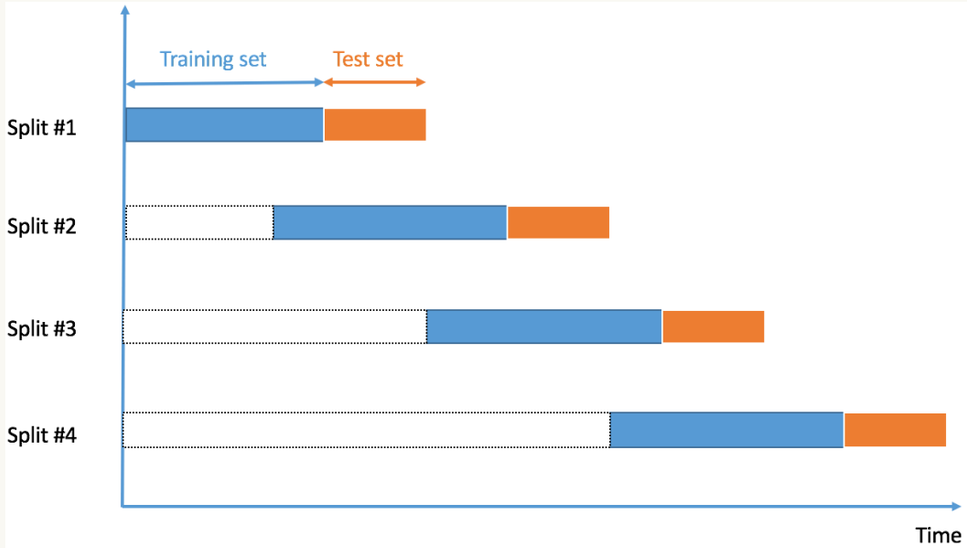
```
recipe() %>%
  step_pca(num_comp = tune())
```

Resampling and grid search

We'll use time series resampling and grid search to optimize the model:

```
chicago_rs <- sliding_period(
  Chicago,
  date,
  period = "month",
  lookback = 14 * 12,
  assess_stop = 1
)

print(chicago_rs, n = 6)
```

```
## # Sliding period resampling
## # A tibble: 19 × 2
##   splits             id
##   <list>             <chr>
## 1 <split [5123/28]> Slice01
## 2 <split [5141/31]> Slice02
## 3 <split [5144/30]> Slice03
## 4 <split [5143/31]> Slice04
## 5 <split [5144/30]> Slice05
## 6 <split [5143/31]> Slice06
## # … with 13 more rows
```



From: https://thierrymoudiki.github.io//blog/2020/03/27/2

Resampling and grid search

What's in a split?

```
split <- chicago_rs$splits[[1]]
split
```

```
## <Analysis/Assess/Total>
## <5123/28/5698>
```

```
print(analysis(split), n = 3)
```

```
## # A tibble: 5,123 × 50
##   ridership Austin Quincy_Wells Belmont Archer_3
##       <dbl>  <dbl>        <dbl>   <dbl>       <d
## 1      15.7   1.46         8.37    4.60        2
## 2      15.8   1.50         8.35    4.72        2
## 3      15.9   1.52         8.36    4.68        2
## # … with 5,120 more rows, and 42 more variables:
## #   Merchandise_Mart <dbl>, Irving_Park <dbl>, W
## #   Harlem <dbl>, Monroe <dbl>, Polk <dbl>, Ashl
## #   Addison <dbl>, Jefferson_Park <dbl>, Montros
## #   temp_min <dbl>, temp <dbl>, temp_max <dbl>,
## #   humidity <dbl>, pressure <dbl>, pressure_cha
## #   wind_max <dbl>, gust <dbl>, gust_max <dbl>,
```

```
print(assessment(split), n = 3)
```

```
## # A tibble: 28 × 50
##   ridership Austin Quincy_Wells Belmont Archer_3
##       <dbl>  <dbl>        <dbl>   <dbl>       <d
## 1      3.47  0.706         1.17    2.42        1
## 2     14.0   1.22          4.79    4.05        1
## 3     20.5   2.12          8.22    5.52        3
## # … with 25 more rows, and 42 more variables: Cl
## #   Merchandise_Mart <dbl>, Irving_Park <dbl>, W
## #   Harlem <dbl>, Monroe <dbl>, Polk <dbl>, Ashl
## #   Addison <dbl>, Jefferson_Park <dbl>, Montros
## #   temp_min <dbl>, temp <dbl>, temp_max <dbl>,
## #   humidity <dbl>, pressure <dbl>, pressure_cha
## #   wind_max <dbl>, gust <dbl>, gust_max <dbl>,
```

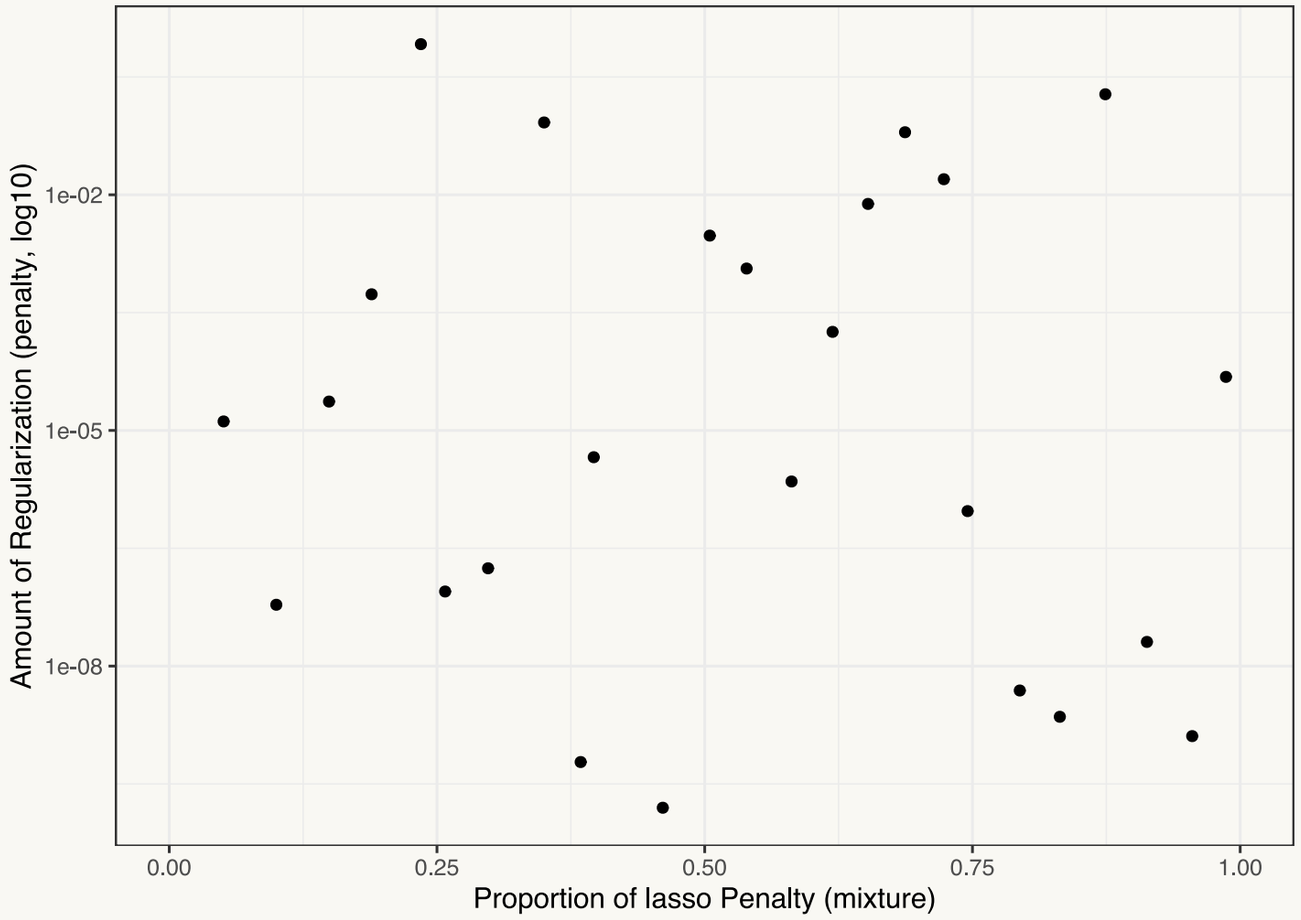
Resampling and grid search

We'll use time series resampling and grid search to optimize the model:

```
library(doMC)
# Or `doFuture::registerDoFuture()`
registerDoMC(cores = parallel::detectCores())

set.seed(1)

glmnet_tune <- glmnet_wflow %>%
  tune_grid(chicago_rs, grid = 25)

show_best(glmnet_tune, metric = "rmse")
```

```
## # A tibble: 5 × 8
##    penalty mixture .metric .estimator  mean     n std_err .config
##      <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 3.02e- 3   0.505 rmse    standard    2.05    19   0.198 Preprocessor1_Model12
## 2 1.15e- 3   0.539 rmse    standard    2.05    19   0.196 Preprocessor1_Model13
## 3 1.57e-10   0.461 rmse    standard    2.05    19   0.196 Preprocessor1_Model11
## 4 4.89e- 9   0.794 rmse    standard    2.05    19   0.196 Preprocessor1_Model20
## 5 4.81e- 5   0.987 rmse    standard    2.05    19   0.196 Preprocessor1_Model25
```

Default space-filling design



Next steps

There are functions to:

- Plot the results
- Substitute the best parameters for the `tune()` placeholders with `finalize_model()`
- Fit the final model with `last_fit()`
- Measure the test set performance

Retrospective vs Prospective

These APIs focus on harmonizing *existing* packages (i.e. the retrospective approach).

Ideally, package authors would create packages adhering to a set of principles which match our expectations as R users, and make it easy to incorporate their packages into the tidymodels ecosystem.

Principles of Modeling Packages

We have a set of *guidelines* for making good modeling packages. For example:

- Support multiple expected interfaces (e.g. formula, x/y, recipe, etc).
- The *user-facing interface* should use the most intuitive data structures for the data (as opposed to the computations). For example, allowing factor outcomes versus requiring 0/1 indicators and data frames versus matrices.
- Separate the interface that the **modeler** uses from the code to do the computations.
- The `predict()` method should produce standardized, predictable results, generated by a standardized `type` (i.e. `type = "prob"` for class probabilities 😄).

Rather than try to make method designers into software developers, have tools to **help them create high quality modeling packages**.

Making better packages with hardhat

Ever used `usethis::create_package()`?

On the modeling side we have `hardhat::create_modeling_package()`.

This generates opinionated scaffolding for new modeling packages.

```
library(hardhat)

create_modeling_package("~/tmp/lantern", model = "lantern_mlp")
```

The result is a "working" user interface that you can plug your implementation code into.

There is a video demo that shows how to create a package in 9 steps.

What's coming up

ML OPS

Model Operations tools for deploying, monitoring, and updating models.

CENSORED DATA MODELS (A.K.A SURVIVAL ANALYSIS)

These models are used to predict time-to-event data (in medicine, sports, HR, etc).

CASE WEIGHTS

Tedious, but often requested. This will impact many of the tidymodels packages.

Thanks

Thanks for the invitation to speak today!

The tidymodels team: Max Kuhn, Julia Silge, and Hannah Frick.

Special thanks to other tidymodels contributors: Edgar Ruiz, Emil Hvitfeldt, Alison Hill, Desirée De Leon, and the tidyverse team.

These slides were made with the `xaringan` package and styled by Alison Hill.

Sources are located at `github.com/DavisVaughan/an-introduction-to-tidymodels`.