This week I started my new job as a software engineer at RStudio, working with Max Kuhn and other folks on [tidymodels](https://www.tidyverse.org/blog/2018/08/tidymodels-0-0-1/). I am really excited about tidymodels because my own experience as a practicing data scientist has shown me some of the areas for growth that still exist in open source software when it comes to modeling and machine learning. Almost nothing has had the kind of dramatic impact on my productivity that the tidyverse and other RStudio investments have had; I am enthusiastic about contributing to that kind of user-focused transformation for modeling and machine learning.

The tidymodels ecosystem is still maturing, but with the release of tune is becoming an option for modeling workflows in the real world. I am still getting my bearings with tidymodels and where current development is happening (and headed next!) but I want to start showing how to use tidymodels in some easy-to-digest ways.

Here is the code, for those who prefer reading instead of or in addition to video.

**Explore data**

Our goal here is to build some very simple models. First, we’ll read in the two files and join them together.

library(tidyverse)

attendance <- read\_csv("<https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-02-04/attendance.csv>")

standings <- read\_csv("<https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-02-04/standings.csv>")

attendance\_joined <- attendance %>%

left\_join(standings,

by = c("year", "team\_name", "team")

)

attendance\_joined

## # A tibble: 10,846 x 20

## team team\_name year total home away week weekly\_attendan… wins loss

##

## 1 Ariz… Cardinals 2000 893926 387475 506451 1 77434 3 13

## 2 Ariz… Cardinals 2000 893926 387475 506451 2 66009 3 13

## 3 Ariz… Cardinals 2000 893926 387475 506451 3 NA 3 13

## 4 Ariz… Cardinals 2000 893926 387475 506451 4 71801 3 13

## 5 Ariz… Cardinals 2000 893926 387475 506451 5 66985 3 13

## 6 Ariz… Cardinals 2000 893926 387475 506451 6 44296 3 13

## 7 Ariz… Cardinals 2000 893926 387475 506451 7 38293 3 13

## 8 Ariz… Cardinals 2000 893926 387475 506451 8 62981 3 13

## 9 Ariz… Cardinals 2000 893926 387475 506451 9 35286 3 13

## 10 Ariz… Cardinals 2000 893926 387475 506451 10 52244 3 13

## # … with 10,836 more rows, and 10 more variables: points\_for ,

## # points\_against , points\_differential , margin\_of\_victory ,

## # strength\_of\_schedule , simple\_rating , offensive\_ranking ,

## # defensive\_ranking , playoffs , sb\_winner

but noticed that we have information on weekly attendance at NFL games, along with characteristics of team records per season such as SRS (Simple Rating System), how many points were scored for/against teams, whether a team made the playoffs, and more. Let’s build a model to predict **weekly attendance**.

How does weekly attendance vary for different teams, and for the seasons they did/did not make the playoffs?

attendance\_joined %>%

filter(![is.na](http://is.na)(weekly\_attendance)) %>%

ggplot(aes(fct\_reorder(team\_name, weekly\_attendance),

weekly\_attendance,

fill = playoffs

)) +

geom\_boxplot(outlier.alpha = 0.5) +

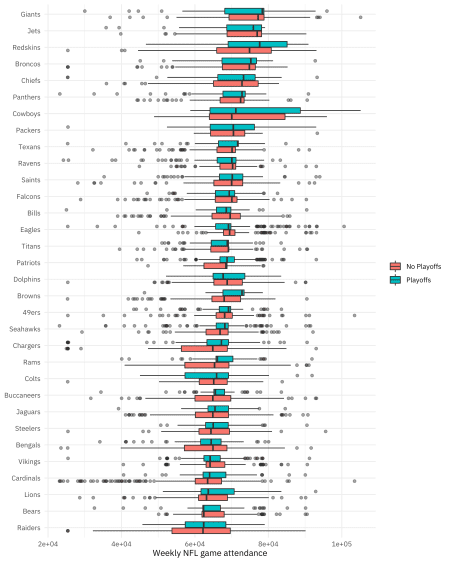
coord\_flip() +

labs(

fill = NULL, x = NULL,

y = "Weekly NFL game attendance"

)



Notice that for the 32 teams in the NFL, we have years they all did and did not make the playoffs, which will be nice for modeling.

How much does margin\_of\_victory, a measure of points scored relative to points allowed, measure the same thing as getting to the playoffs?

attendance\_joined %>%

distinct(team\_name, year, margin\_of\_victory, playoffs) %>%

ggplot(aes(margin\_of\_victory, fill = playoffs)) +

geom\_histogram(position = "identity", alpha = 0.7) +

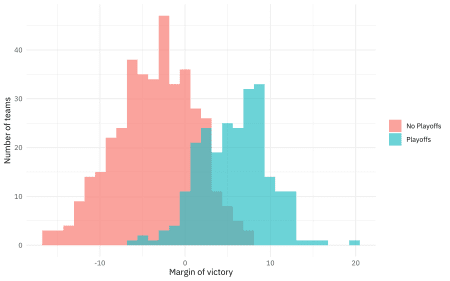
labs(

x = "Margin of victory",

y = "Number of teams",

fill = NULL

)



Are there changes with the week of the season?

attendance\_joined %>%

mutate(week = factor(week)) %>%

ggplot(aes(week, weekly\_attendance, fill = week)) +

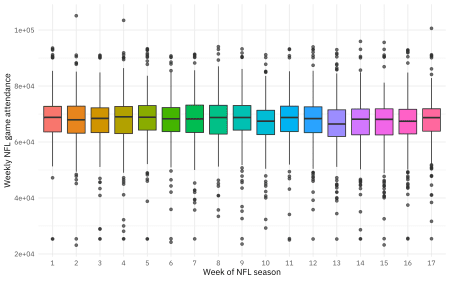
geom\_boxplot(show.legend = FALSE, outlier.alpha = 0.5) +

labs(

x = "Week of NFL season",

y = "Weekly NFL game attendance"

)



Maybe a bit.

This is some initial exploratory data analysis for this dataset, always an important part of a modeling task. To see more examples of EDA for this dataset, you can see the [amazing work that folks share on Twitter](https://twitter.com/hashtag/tidytuesday). The next step for us here is to create a dataset for modeling.

* Let’s remove the weeks that each team did not play (where the weekly attendance is NA).
* Let’s only keep columns for modeling that we want to use for modeling. For example, we will keep margin\_of\_victory and strength\_of\_schedule, but not simple\_rating which is the sum of those first two quantities.

attendance\_df <- attendance\_joined %>%

filter(![is.na](http://is.na)(weekly\_attendance)) %>%

select(

weekly\_attendance, team\_name, year, week,

margin\_of\_victory, strength\_of\_schedule, playoffs

)

attendance\_df

## # A tibble: 10,208 x 7

## weekly\_attendan… team\_name year week margin\_of\_victo… strength\_of\_sch…

##

## 1 77434 Cardinals 2000 1 -14.6 -0.7

## 2 66009 Cardinals 2000 2 -14.6 -0.7

## 3 71801 Cardinals 2000 4 -14.6 -0.7

## 4 66985 Cardinals 2000 5 -14.6 -0.7

## 5 44296 Cardinals 2000 6 -14.6 -0.7

## 6 38293 Cardinals 2000 7 -14.6 -0.7

## 7 62981 Cardinals 2000 8 -14.6 -0.7

## 8 35286 Cardinals 2000 9 -14.6 -0.7

## 9 52244 Cardinals 2000 10 -14.6 -0.7

## 10 64223 Cardinals 2000 11 -14.6 -0.7

## # … with 10,198 more rows, and 1 more variable: playoffs

**Build simple models**

The first step here is to split our data into training and testing tests. We can use initial\_split() to create these datasets, divided so that they each have about the same number of examples of teams that went on to the playoffs.

library(tidymodels)

set.seed(1234)

attendance\_split <- attendance\_df %>%

initial\_split(strata = playoffs)

nfl\_train <- training(attendance\_split)

nfl\_test <- testing(attendance\_split)

Now we can **specify** and then **fit** our models. One of the significant problems that tidymodels solves is how so many modeling packages and functions in R have different inputs, calling sequences, and outputs. The code below might look like overkill to fit linear regression using OLS, but we can use the same framework to fit a regression model using Stan, using regularization, etc. The functions in tidymodels are designed to be composable and consistent.

lm\_spec <- linear\_reg() %>%

set\_engine(engine = "lm")

lm\_spec

## Linear Regression Model Specification (regression)

##

## Computational engine: lm

lm\_fit <- lm\_spec %>%

fit(weekly\_attendance ~ .,

data = nfl\_train

)

lm\_fit

## parsnip model object

##

## Fit time: 16ms

##

## Call:

## stats::lm(formula = formula, data = data)

##

## Coefficients:

## (Intercept) team\_nameBears team\_nameBengals

## -81107.86 -2879.80 -4875.47

## team\_nameBills team\_nameBroncos team\_nameBrowns

## -361.08 2805.24 -324.11

## team\_nameBuccaneers team\_nameCardinals team\_nameChargers

## -3063.65 -6139.80 -6489.31

## team\_nameChiefs team\_nameColts team\_nameCowboys

## 1974.33 -3392.79 6068.70

## team\_nameDolphins team\_nameEagles team\_nameFalcons

## 139.68 1259.16 -204.17

## team\_nameGiants team\_nameJaguars team\_nameJets

## 5447.10 -3095.46 4044.23

## team\_nameLions team\_namePackers team\_namePanthers

## -3480.69 1114.11 1227.32

## team\_namePatriots team\_nameRaiders team\_nameRams

## -214.20 -6324.74 -2884.85

## team\_nameRavens team\_nameRedskins team\_nameSaints

## -398.90 6447.07 380.98

## team\_nameSeahawks team\_nameSteelers team\_nameTexans

## -1405.89 -3567.81 264.07

## team\_nameTitans team\_nameVikings year

## -1118.23 -3183.08 74.73

## week margin\_of\_victory strength\_of\_schedule

## -72.83 137.58 230.74

## playoffsPlayoffs

## -427.94

So that’s one model! Let’s fit another one.

rf\_spec <- rand\_forest(mode = "regression") %>%

set\_engine("ranger")

rf\_spec

## Random Forest Model Specification (regression)

##

## Computational engine: ranger

rf\_fit <- rf\_spec %>%

fit(weekly\_attendance ~ .,

data = nfl\_train

)

rf\_fit

## parsnip model object

##

## Fit time: 2.2s

## Ranger result

##

## Call:

## ranger::ranger(formula = formula, data = data, num.threads = 1, verbose = FALSE, seed = [sample.int](http://sample.int)(10^5, 1))

##

## Type: Regression

## Number of trees: 500

## Sample size: 7656

## Number of independent variables: 6

## Mtry: 2

## Target node size: 5

## Variable importance mode: none

## Splitrule: variance

## OOB prediction error (MSE): 74737868

## R squared (OOB): 0.08217606

Notice that we have fit both of these models using nfl\_train, the training data. We haven’t touched the testing data during training.

**Evaluate models**

When it’s time to evaluate our models (to estimate how well our models will perform on new data), then we will look at nfl\_test. We can predict() what the weekly attendance will be for both the training data and the testing data using both the OLS and random forest models. One of the goals of tidymodels is to be able to use code like the following in predictable, consistent ways for many kinds of models, and to use existing well-suited tidyverse tools for these kinds of tasks.

results\_train <- lm\_fit %>%

predict(new\_data = nfl\_train) %>%

mutate(

truth = nfl\_train$weekly\_attendance,

model = "lm"

) %>%

bind\_rows(rf\_fit %>%

predict(new\_data = nfl\_train) %>%

mutate(

truth = nfl\_train$weekly\_attendance,

model = "rf"

))

results\_test <- lm\_fit %>%

predict(new\_data = nfl\_test) %>%

mutate(

truth = nfl\_test$weekly\_attendance,

model = "lm"

) %>%

bind\_rows(rf\_fit %>%

predict(new\_data = nfl\_test) %>%

mutate(

truth = nfl\_test$weekly\_attendance,

model = "rf"

))

For this regression model, let’s look at the rmse for what we’ve done so far.

results\_train %>%

group\_by(model) %>%

rmse(truth = truth, estimate = .pred)

## # A tibble: 2 x 4

## model .metric .estimator .estimate

##

## 1 lm rmse standard 8307.

## 2 rf rmse standard 6102.

results\_test %>%

group\_by(model) %>%

rmse(truth = truth, estimate = .pred)

## # A tibble: 2 x 4

## model .metric .estimator .estimate

##

## 1 lm rmse standard 8351.

## 2 rf rmse standard 8582.

If we look at the training data, the random forest model performed much better than the linear model; the rmse is much lower. However, the same cannot be said for the testing data! The metric for training and testing for the linear model is about the same, meaning that we have not overfit. For the random forest model, the rmse is *higher* for the testing data than for the training data, by quite a lot. Our training data is not giving us a good idea of how our model is going to perform, and this powerful ML algorithm has overfit to this dataset.

Let’s visualize our sad situation.

results\_test %>%

mutate(train = "testing") %>%

bind\_rows(results\_train %>%

mutate(train = "training")) %>%

ggplot(aes(truth, .pred, color = model)) +

geom\_abline(lty = 2, color = "gray80", size = 1.5) +

geom\_point(alpha = 0.5) +

facet\_wrap(~train) +

labs(

x = "Truth",

y = "Predicted attendance",

color = "Type of model"

)

**Let’s try this again!**

We made not such a great decision in the previous section; we expected the random forest model evaluated one time on the whole training set to help us understand something about how it would perform on new data. This would be a reasonable expectation for the linear model, but not for the random forest. Fortunately, we have some options. We can **resample** the training set to produce an estimate of how the model will perform. Let’s divide our training set nfl\_train into folds (say, 10) and fit 10 versions of our model (each one trained on nine folds and evaluated on one heldout fold). Then let’s measure how well our model(s) performs. The function vfold\_cv() creates folds for cross-validation, the function fit\_resamples() fits models to resamples such as these (to measure performance), and then we can collect\_metrics() from the result.

set.seed(1234)

nfl\_folds <- vfold\_cv(nfl\_train, strata = playoffs)

rf\_res <- fit\_resamples(

weekly\_attendance ~ .,

rf\_spec,

nfl\_folds,

control = control\_resamples(save\_pred = TRUE)

)

rf\_res %>%

collect\_metrics()

## # A tibble: 2 x 5

## .metric .estimator mean n std\_err

##

## 1 rmse standard 8255. 10 115.

## 2 rsq standard 0.164 10 0.00988

Remember that this is still the **training** dataset. We would take this step instead of the chunk above with predict(new\_data = nfl\_train), and we would still compare to how the model performs on the testing data. Notice that now we have a realistic estimate from the training data that is close to the testing data! We can even visualize our model results for the resamples.

rf\_res %>%

unnest(.predictions) %>%

ggplot(aes(weekly\_attendance, .pred, color = id)) +

geom\_abline(lty = 2, color = "gray80", size = 1.5) +

geom\_point(alpha = 0.5) +

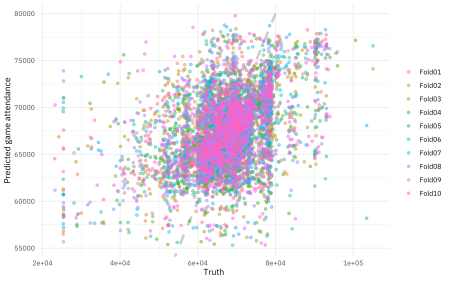
labs(

x = "Truth",

y = "Predicted game attendance",

color = NULL

)



**Summary**

Let me know if you have questions or feedback about this introduction to tidymodels and how to get started. I hope to do some more screencasts and blog posts about getting started with tidymodels, perhaps with more #TidyTuesday data!