Applied Machine Learning - Feature Engineering

Max Kuhn (RStudio)

Loading

```r
library(tidymodels)
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 0.0.2 ──
```

```
## ✔ broom     0.5.2       ✔ purrr     0.3.2
## ✔ dials     0.0.2       ✔ recipes   0.1.6
## ✔ dplyr     0.8.3       ✔ rsample   0.0.5
## ✔ ggplot2   3.2.0       ✔ tibble    2.1.3
## ✔ infer     0.4.0.1     ✔ yardstick 0.0.3
## ✔ parsnip   0.0.3
```

```
## ── Conflicts ──────────────────────────────────────── tidymodels_conflicts() ──
## ✖ purrr::discard() masks scales::discard()
## ✖ dplyr::filter()  masks stats::filter()
## ✖ dplyr::lag()     masks stats::lag()
## ✖ recipes::step()  masks stats::step()
```

Previously

```r
library(AmesHousing)

ames <-
  make_ames() %>%
  dplyr::select(-matches("Qu"))

set.seed(4595)
data_split <- initial_split(ames, strata = "Sale_Price")
ames_train <- training(data_split)
ames_test  <- testing(data_split)

spec_lm <-
  linear_reg() %>%
  set_engine("lm")

perf_metrics <- metric_set(rmse, rsq, ccc)
```

Feature Engineering

Preprocessing and Feature Engineering

This part mostly concerns what we can *do* to our variables to make the models more effective.

This is mostly related to the predictors. Operations that we might use are:

- transformations of individual predictors or groups of variables
- alternate encodings of a variable
- elimination of predictors (unsupervised)

In statistics, this is generally called *preprocessing* the data. As usual, the computer science side of modeling has a much flashier name: *feature engineering*.

Reasons for Modifying the Data

- Some models (K-NN, SVMs, PLS, neural networks) require that the predictor variables have the same units. **Centering** and **scaling** the predictors can be used for this purpose.
- Other models are very sensitive to correlations between the predictors and **filters** or **PCA signal extraction** can improve the model.
- As we'll see in an example, changing the scale of the predictors using a **transformation** can lead to a big improvement.
- In other cases, the data can be **encoded** in a way that maximizes its effect on the model. Representing the date as the day of the week can be very effective for modeling public transportation data.
- Many models cannot cope with missing data so **imputation** strategies might be necessary.
- Development of new *features* that represent something important to the outcome (e.g. compute distances to public transportation, university buildings, public schools, etc.)

Preprocessing Categorical Predictors

Dummy Variables

One common procedure for modeling is to create numeric representations of categorical data. This is usually done via *dummy variables*: a set of binary 0/1 variables for different levels of an R factor.

For example, the Ames housing data contains a predictor called `Alley` with levels: 'Gravel', 'No_Alley_Access', 'Paved'.

Most dummy variable procedures would make *two* numeric variables from this predictor that are 1 when the observation has that level, and 0 otherwise.

Data	**Dummy Variables**	
	No_Alley_Access	**Paved**
Gravel	0	0
No_Alley_Access	1	0
Paved	0	1

Dummy Variables

If there are *C* levels of the factor, only *C*-1 dummy variables are created since the last can be inferred from the others. There are different contrast schemes for creating the new variables.

For ordered factors, *polynomial* contrasts are used. This is, in many cases, a bad idea. I usually convert them to unordered factors.

How do you create them in R?

The formula method does this for you[1]. Otherwise, the traditional method is to use `model.matrix()` to create a matrix. However, there are some caveats to this that can make things difficult.

We'll show another method for making them shortly.

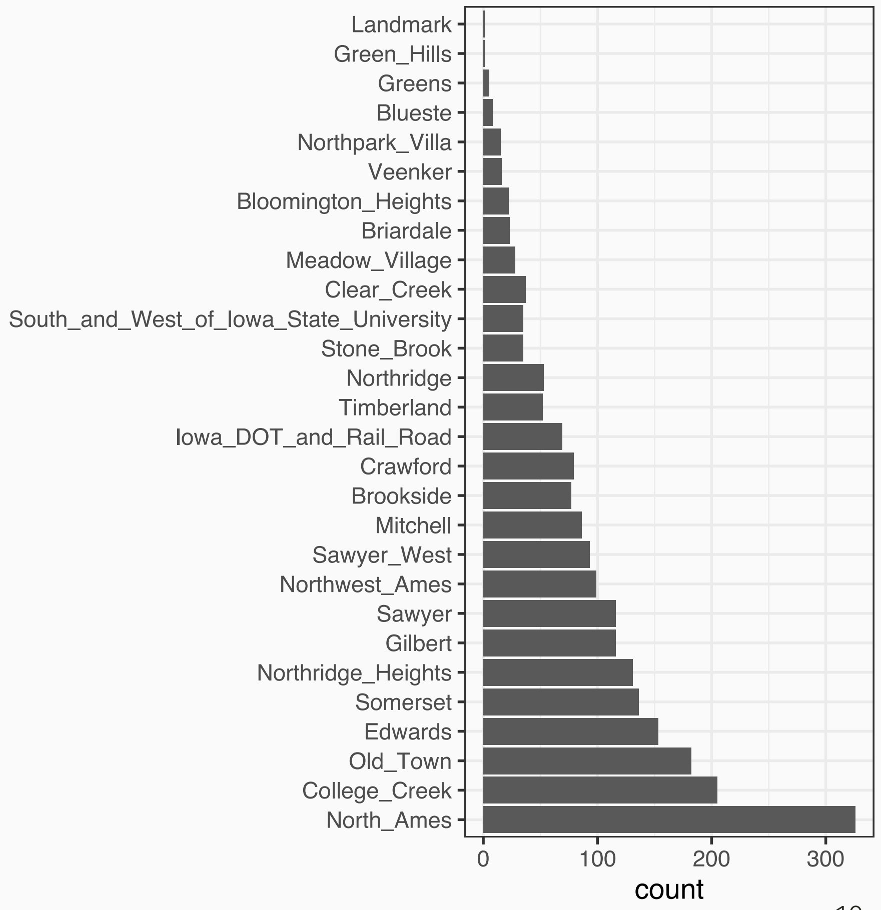
[1] *Almost always* at least. Tree- and rule-based model functions do not. Examples are `randomforest::randomForest`, `ranger::ranger`, `rpart::rpart`, `C50::C5.0`, `Cubist::cubist`, `klaR::NaiveBayes` and others.

Infrequent Levels in Categorical Factors

One issue is: what happens when there are very few values of a level?

Consider the Ames training set and the `Neighborhood` variable.

If these data are resampled, what would happen to Landmark and similar locations when dummy variables are created?



Infrequent Levels in Categorical Factors

A *zero-variance* predictor that has only a single value (zero) would be the result.

Many models (e.g. linear/logistic regression, etc.) would find this numerically problematic and issue a warning and `NA` values for that coefficient. Trees and similar models would not notice.

There are two main approaches to dealing with this:

- Run a filter on the training set predictors prior to running the model and remove the zero-variance predictors.
- Recode the factor so that infrequently occurring predictors (and possibly new values) are pooled into an "other" category.

However, `model.matrix()` and the formula method are incapable of doing either of these.

Recipes

Recipes are an alternative method for creating the data frame of predictors for a model.

They allow for a sequence of *steps* that define how data should be handled.

Recall the previous part where we used the formula `log10(Sale_Price) ~ Longitude + Latitude`? These steps are:

- Assign `Sale_Price` to be the outcome
- Assign `Longitude` and `Latitude` as predictors
- Log transform the outcome

To start using a recipe, these steps can be done using

```r
# recipes loaded by tidymodels
mod_rec <- recipe(Sale_Price ~ Longitude + Latitude, data = ames_train) %>%
  step_log(Sale_Price, base = 10)
```

This creates the recipe for data processing (but does not execute it yet)

Recipes and Categorical Predictors

To deal with the dummy variable issue, we can expand the recipe with more steps:

```r
mod_rec <- recipe(
    Sale_Price ~ Longitude + Latitude + Neighborhood,
    data = ames_train
  ) %>%
  step_log(Sale_Price, base = 10) %>%

  # Lump factor levels that occur in
  # <= 5% of data as "other"
  step_other(Neighborhood, threshold = 0.05) %>%

  # Create dummy variables for _any_ factor variables
  step_dummy(all_nominal())
```

```r
mod_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          3
##
## Operations:
##
## Log transformation on Sale_Price
## Collapsing factor levels for Neighborhood
## Dummy variables from all_nominal
```

Note that we can use standard `dplyr` selectors as well as some new ones based on the data type (`all_nominal()`) or by their role in the analysis (`all_predictors()`).

Using Recipes

```
 recipe   -->      prep     --> bake/juice

(define) --> (calculate) -->   (apply)
```

Preparing the Recipe

Now that we have a preprocessing *specification*, let's run it on the training set to *prepare* the recipe:

```
mod_rec_trained <- prep(mod_rec, training = ames_train, verbose = TRUE)
```

```
## oper 1 step log [training]
## oper 2 step other [training]
## oper 3 step dummy [training]
## The retained training set is ~ 0.19 Mb  in memory.
```

Here, the "training" is to determine which levels to lump together and to enumerate the factor levels of the `Neighborhood` variable.

An unused option, `retain = TRUE`, that keeps the processed version of the training set around so we don't have to recompute it.

Preparing the Recipe

```
mod_rec_trained
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          3
##
## Training data contained 2199 data points and no missing data.
##
## Operations:
##
## Log transformation on Sale_Price [trained]
## Collapsing factor levels for Neighborhood [trained]
## Dummy variables from Neighborhood [trained]
```

Getting the Values

Once the recipe is prepared, it can be applied to any data set using `bake()`:

```
ames_test_dummies <- bake(mod_rec_trained, new_data = ames_test)
names(ames_test_dummies)
```

```
##  [1] "Sale_Price"                  "Longitude"                   "Latitude"
##  [4] "Neighborhood_College_Creek"  "Neighborhood_Old_Town"       "Neighborhood_Edwards"
##  [7] "Neighborhood_Somerset"       "Neighborhood_Northridge_Heights" "Neighborhood_Gilbert"
## [10] "Neighborhood_Sawyer"         "Neighborhood_other"
```

If `retain = TRUE`, the training set does not need to be "rebaked". The `juice()` function can return the processed version of the training data.

Selectors can be used with `bake()` to only extract relevant columns and the default is `everything()`.

`juice()` is used to get the training set results (for almost free)

`bake()` is used for any data set

How Data Are Used

Note that we have:

```
recipe(..., data = data_set)
prep(...,   training = data_set)
bake(...,   new_data = data_set)
```

In the first case, `data` is used by the `recipe` function only to determine the column names and types (e.g. factor, numeric, etc). A small subset can be passed via `head` or `slice`.

For `prep`, the `training` argument should have all of the data used to estimate parameters and other quantities.

For `bake`, `new_data` is the data that the pre-processing should be applied to.

Hands-On: Zero-Variance Filter

Instead of using `step_other()`, take 10 minutes and research how to eliminate any zero-variance predictors using the `recipe` reference site.

Re-run the recipe with this step.

What were the results?

Do you prefer either of these approaches to the other?

10:00

Interaction Effects

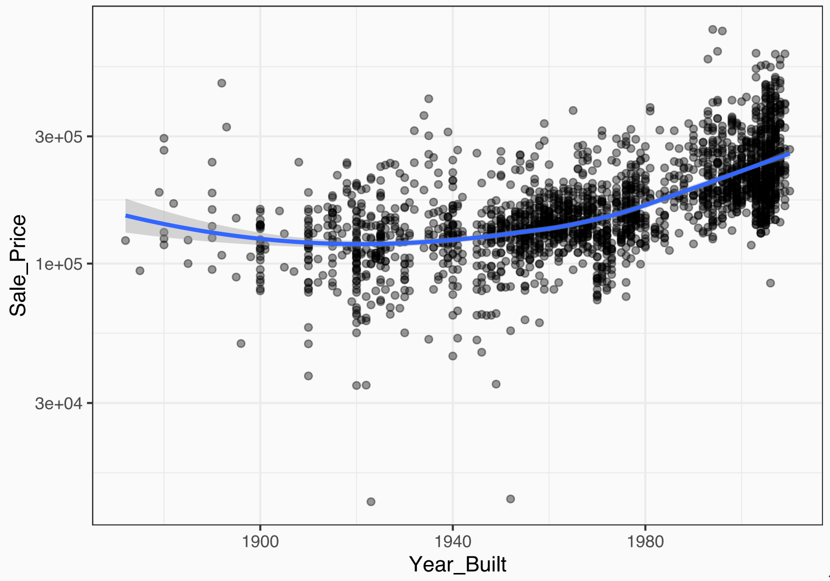
Interactions

An **interaction** between two predictors indicates that the relationship between the predictors and the outcome cannot be describe using only one of the variables.

For example, let's look at the relationship between the price of a house and the year in which it was built. The relationship appears to be slightly nonlinear, possibly quadratic:

```r
price_breaks <- (1:6)*(10^5)

ggplot(
    ames_train,
    aes(x = Year_Built, y = Sale_Price)
  ) +
  geom_point(alpha = 0.4) +
  scale_y_log10() +
  geom_smooth(method = "loess")
```

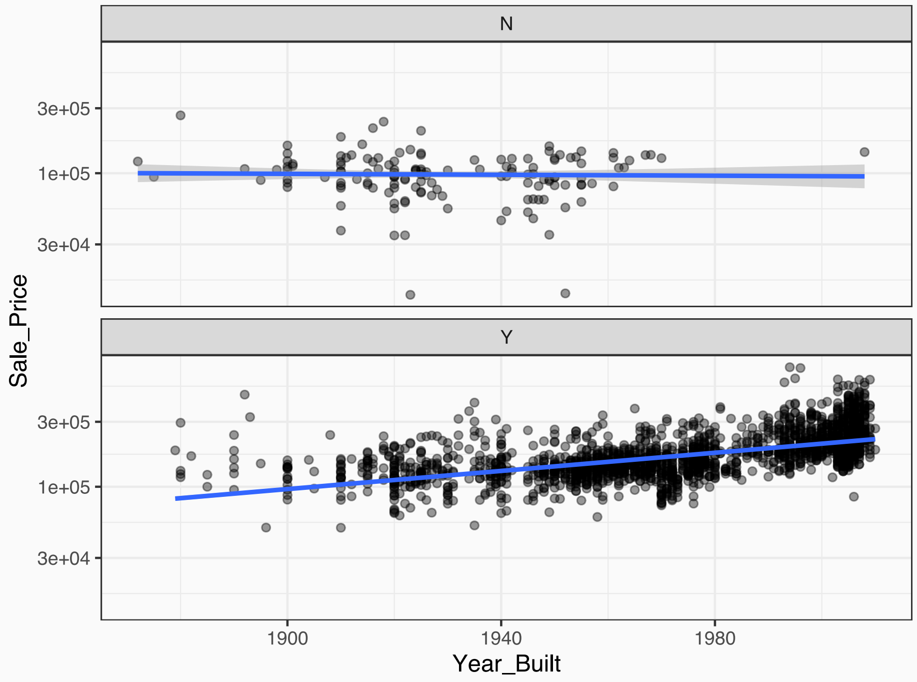


Interactions

However... what if we seperate this trend based on whether the property has air conditioning (93.6% of the training set) or not (6.4%):

```r
library(MASS) # to get robust linear regression model

ggplot(
    ames_train,
    aes(x = Year_Built,
        y = Sale_Price)
  ) +
  geom_point(alpha = 0.4) +
  scale_y_log10() +
  facet_wrap(~ Central_Air, nrow = 2) +
  geom_smooth(method = "rlm")
```



Interactions

It appears as though the relationship between the year built and the sale price is somewhat *different* for the two groups.

- When there is no AC, the trend is perhaps flat or slightly decreasing.
- With AC, there is a linear increasing trend or is perhaps slightly quadratic with some outliers at the low end.

```r
mod1 <- lm(log10(Sale_Price) ~ Year_Built + Central_Air, data = ames_train)
mod2 <- lm(log10(Sale_Price) ~ Year_Built + Central_Air + Year_Built:Central_Air, data = ames_train)
anova(mod1, mod2)
```

```
## Analysis of Variance Table
##
## Model 1: log10(Sale_Price) ~ Year_Built + Central_Air
## Model 2: log10(Sale_Price) ~ Year_Built + Central_Air + Year_Built:Central_Air
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1   2196 42.741
## 2   2195 41.733  1    1.0075 52.993 4.64e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interactions in Recipes

We first create the dummy variables for the qualitative predictor (`Central_Air`) then use a formula to create the interaction using the `:` operator in an additional step:

```
recipe(Sale_Price ~ Year_Built + Central_Air, data = ames_train) %>%
  step_log(Sale_Price) %>%
  step_dummy(Central_Air) %>%
  step_interact(~ starts_with("Central_Air"):Year_Built) %>%
  prep(training = ames_train) %>%
  juice() %>%
  # select a few rows with different values
  slice(153:157)
```

```
## # A tibble: 5 x 4
##   Year_Built Sale_Price Central_Air_Y Central_Air_Y_x_Year_Built
##        <int>      <dbl>         <dbl>                      <dbl>
## 1       1915       11.9             1                       1915
## 2       1912       12.0             1                       1912
## 3       1920       11.7             1                       1920
## 4       1963       11.6             0                          0
## 5       1930       10.9             0                          0
```

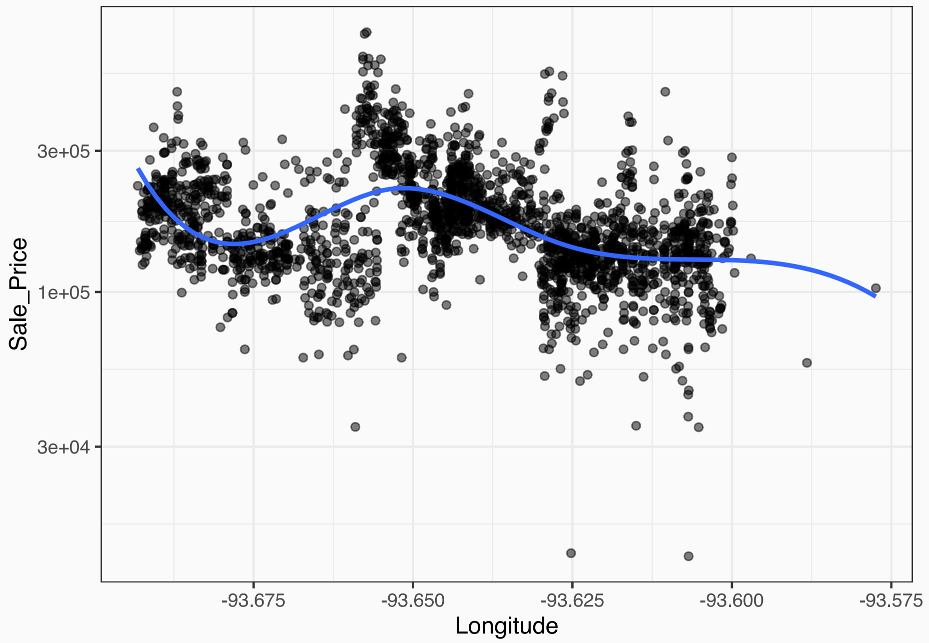
Recipe and Models

Longitude

```r
ggplot(ames_train,
       aes(x = Longitude, y = Sale_Price)) +
  geom_point(alpha = .5) +
  geom_smooth(
    method = "lm",
    formula = y ~ splines::bs(x, 5),
    se = FALSE
  ) +
  scale_y_log10()
```

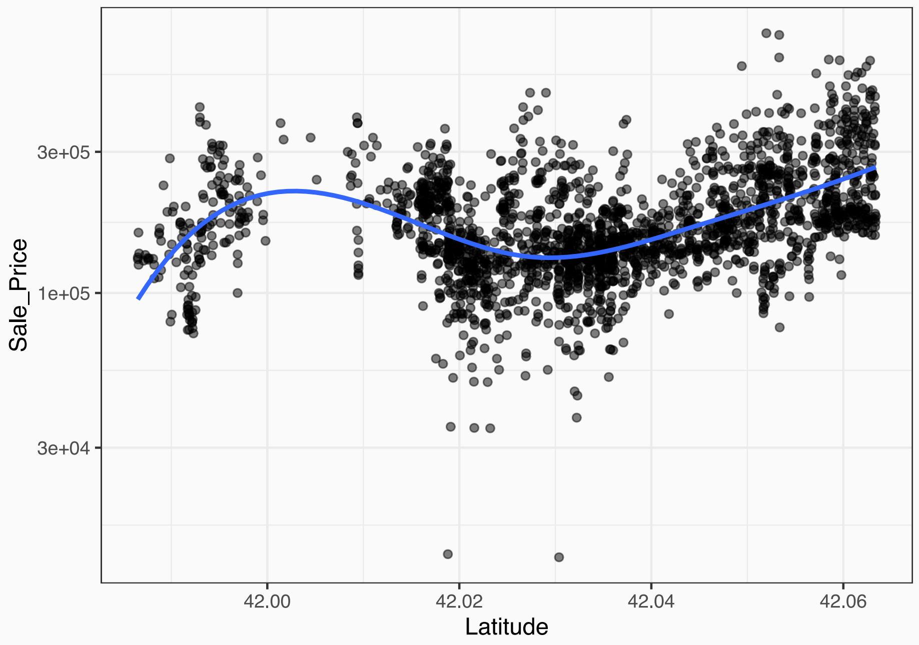
Splines add nonlinear versions of the predictor to a linear model to create smooth and flexible relationships between the predictor and outcome.

This "basis expansion" technique will be seen again in the regression section of the workshop.



Latitude

```r
ggplot(ames_train,
       aes(x = Latitude, y = Sale_Price)) +
  geom_point(alpha = .5) +
  geom_smooth(
    method = "lm",
    formula = y ~ splines::bs(x, 5),
    se = FALSE
  ) +
  scale_y_log10()
```



Linear Models Again

Let's add a few extra predictors and some preprocessing.

- Two numeric predictors are very skewed and could use a transformation (`Lot_Area` and `Gr_Liv_Area`).
- We'll add neighborhood in as well and a few other house features.
- Our plots suggests that the coordinates can be helpful but probably require a nonlinear representation. We can add these using *B-splines* with 5 degrees of freedom. To evaluate this, we will create two versions of the recipe to evaluate this hypothesis.

```
ames_rec <-
  recipe(Sale_Price ~ Bldg_Type + Neighborhood + Year_Built +
           Gr_Liv_Area + Full_Bath + Year_Sold + Lot_Area +
           Central_Air + Longitude + Latitude,
         data = ames_train) %>%
  step_log(Sale_Price, base = 10) %>%
  step_BoxCox(Lot_Area, Gr_Liv_Area) %>%
  step_other(Neighborhood, threshold = 0.05)  %>%
  step_dummy(all_nominal()) %>%
  step_interact(~ starts_with("Central_Air"):Year_Built) %>%
  step_bs(Longitude, Latitude, options = list(df = 5))
```

Combining the Recipe with a Model

It's pretty simple:

```
ames_rec <- prep(ames_rec)

fit_lm <-
  spec_lm %>%
  fit(Sale_Price ~ ., data = juice(ames_rec))   # The recipe puts Sale_Price on the log scale

glance(fit_lm$fit)
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared  sigma statistic p.value    df logLik    AIC    BIC deviance df.residual
##       <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <int>  <dbl>  <dbl>  <dbl>    <dbl>       <int>
## 1     0.800         0.798 0.0804      347.       0    26  2435. -4816. -4662.     14.1        2173
```

```
holdout_data <- bake(ames_rec, ames_test, all_predictors())

# but let's not do this
# predict(fit_lm, new_data = holdout_data)
```

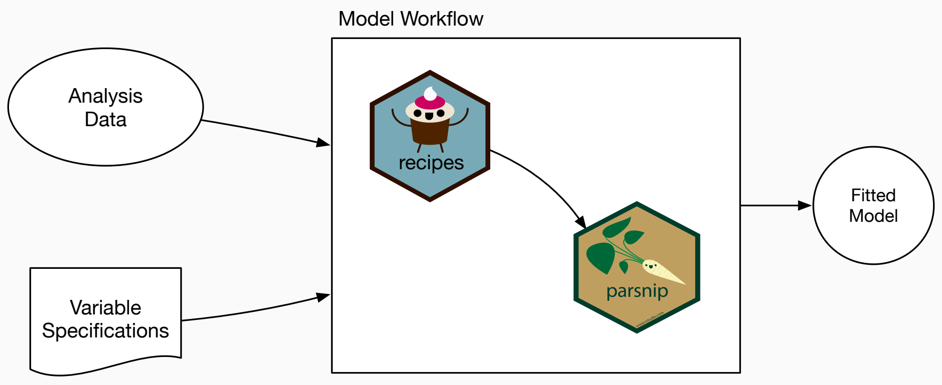
That Modeling Process Again

We are

- estimating transformations on some variables,
- making nonlinear features for the geocoding variables, and
- estimating new encodings for some factor variables.

These should definately be included in our overall modeling process.

This will come into play in the next shortly


Model Workflow
Analysis Data
Variable Specifications
recipes
parsnip
Fitted Model