Feature Engineering

(based on tmwr.org)

Matthew McDonald

What is Feature Engineering

- Reformatting predictor values to make them easier for a model to use effectively.
- Includes transformations and encodings of the data to best represent their important characteristics.

Ames Data

Take the location of a house in Ames as a more involved example.

Ways that this spatial information can be exposed to a model:

- neighborhood (a qualitative measure)
- longitude/latitude
- distance to the nearest school or Iowa State University

Other Examples

- Some algorithms are sensitive to correlated predictor variables.
  - Remedy: feature extraction (PCA) or the removal of some predictors.
- Some algorithms cannot handle missing data.
  - Remedy: Imputation (which can be a sub-model)
- Some algorithms are sensitive to skewed data (outliers)
  - Remedy: Variable Transformation (skewed -> symmetric)
- Some model algorithms use geometric distance metrics
  - Remedy: Numeric predictors must be centered and scaled so that they are all in the same units.
- Some model algorithms are sensitive to class imbalances
  - Remedy: Upsampling/Downsampling the data.

Useful List of Required Feature Engineering

https://www.tmwr.org/pre-proc-table#pre-proc-table

recipes

- A package for data preparation included in tidymodels
- Combine different feature engineering and preprocessing tasks into a single object and then apply these transformations to different data sets.

Preprocessing the Ames Data

A simple regression for estimating sale price:

```
lm(Sale_Price ~ Neighborhood + log10(Gr_Liv_Area) + Year_Built + Bldg_Type, data = ames)
```

Incorporates:

- The neighborhood (qualitative, with 29 neighborhoods in the training set)
- The gross above-grade living area (continuous, named `Gr_Liv_Area`)
- The year built (`Year_Built`)
- The type of building (`Bldg_Type` with values `OneFam` (\(n = 1,936\)), `TwoFmCon` (\(n = 50\)), `Duplex` (\(n = 88\)), `Twnhs` (\(n = 77\)), and `TwnhsE` (\(n = 191\)))
- Note: Sale Price has already been transformed by log10()

Simple recipe for Ames Data

```r
library(tidymodels) # Includes the recipes package
tidymodels_prefer()

simple_ames <-
  recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
         data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_dummy(all_nominal_predictors())
simple_ames
```

```
── Recipe ──────────────────────────────────────────────────────────────────────

── Inputs

Number of variables by role

outcome:   1
predictor: 4

── Operations

• Log transformation on: Gr_Liv_Area

• Dummy variables from: all_nominal_predictors()
```

Advantages of recipes

- These computations can be recycled across models since they are not tightly coupled to the modeling function.
- A recipe enables a broader set of data processing choices than formulas can offer.
- The syntax can be very compact. For example, `all_nominal_predictors()` can be used to capture many variables for specific types of processing while a formula would require each to be explicitly listed.
- All data processing can be captured in a single R object instead of in scripts that are repeated, or even spread across different files.

How Data Are Used by the `recipe()`

Data are passed to recipes at different stages.

- When calling `recipe(..., data)`, the data set is used to determine the data types of each column
- When preparing the data using `fit(workflow, data)`, the training data are used for all estimation operations including a recipe that may be part of the `workflow`
  - Example: determining factor levels to computing PCA components
- When using `predict(workflow, new_data)`, no model or preprocessor parameters like those from recipes are re-estimated using the values in `new_data`.
  - Example: Centering and scaling using `step_normalize()`
    - Means and standard deviations are determined from the training set; new samples at prediction time are standardized using these values from training when `predict()` is invoked.

Using recipes

This object can be attached to a workflow:

```
lm_wflow <- lm_wflow %>%
  add_recipe(simple_ames)

lm_wflow
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: linear_reg()

── Preprocessor ────────────────────────────────────────────────────────────────
2 Recipe Steps

• step_log()
• step_dummy()

── Model ───────────────────────────────────────────────────────────────────────
Linear Regression Model Specification (regression)

Computational engine: lm
```

Fitting and Predicting

```
1  lm_fit <- fit(lm_wflow, ames_train)
```

The `predict()` method applies the same preprocessing that was used on the training set to the new data before passing them along to the model's `predict()` method:

```
1  predict(lm_fit, ames_test %>% slice(1:3))
```

```
# A tibble: 3 × 1
  .pred
  <dbl>
1  5.08
2  5.32
3  5.28
```

Centering and Scaling

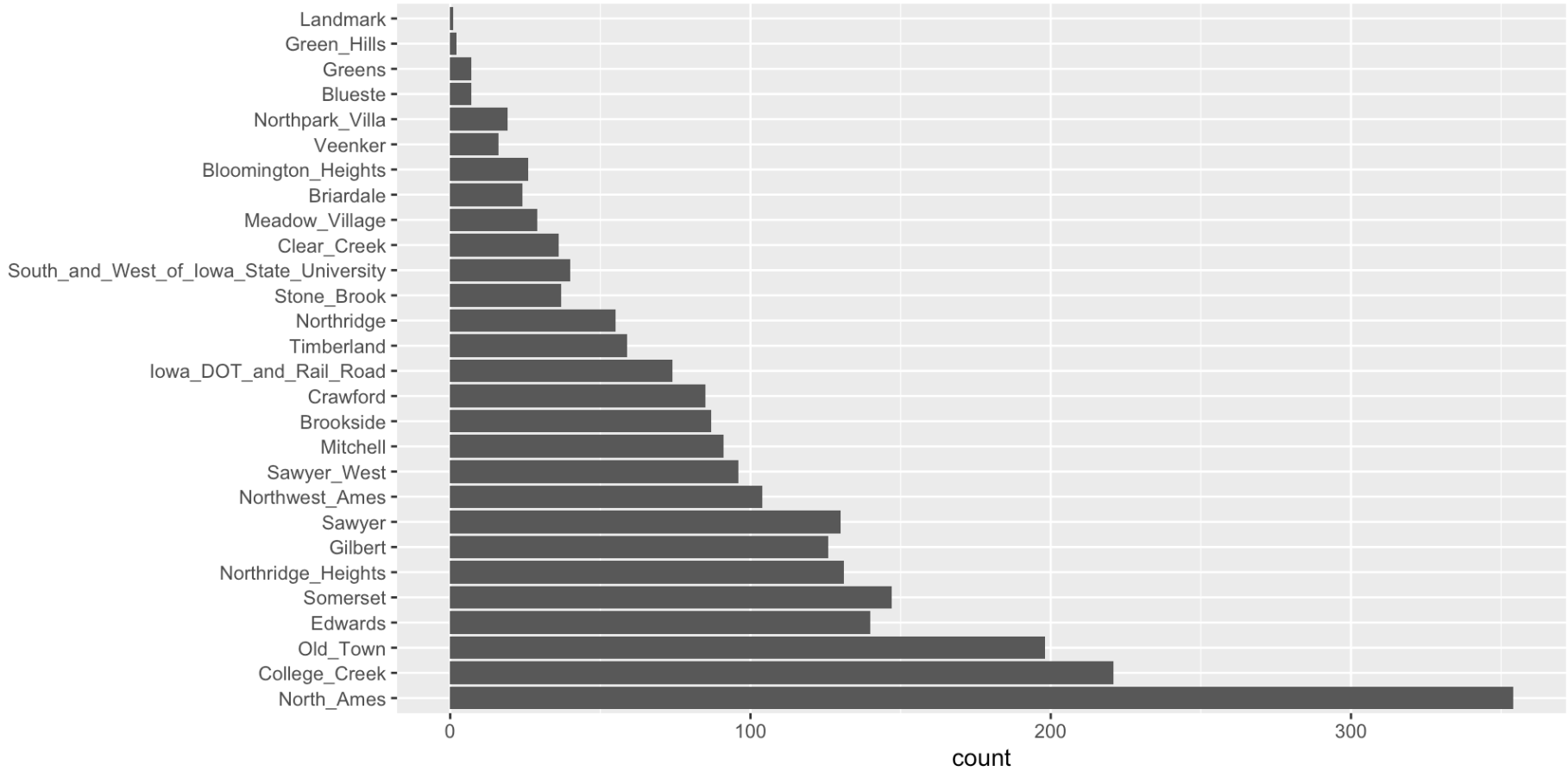
- **Purpose of Centering:** Adjusts each feature to have a mean of zero, removing the mean value from each observation to center the data around the origin.
- **Purpose of Scaling:** Adjusts the scale of the features so that they all have the same standard deviation or variance, typically resulting in each feature having a unit variance.
- **Benefits:** Helps in faster convergence of algorithms, reduces the impact of differing scales among features, and can improve model accuracy and interpretability.
- `step_normalize()` implements centering and scaling within the recipes framework

Dummy Variables

Transforming nominal or qualitative data (factors or characters) so that they can be encoded or represented numerically.

- `step_unknown()` can be used to change missing values to a dedicated factor level.
- `step_novel()` can allot a new level if we anticipate that a new factor level may be encountered in future data
- `step_other()` can be used to analyze the frequencies of the factor levels in the training set and convert infrequently occurring values to a catch-all level of “other,”

Ames Neighborhoods



Frequencies of neighborhoods in the Ames training set

Ames Dummy Variables

For the Ames data, we can amend the recipe to use:

```
simple_ames <-
  recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
         data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  step_dummy(all_nominal_predictors())
```

Dummy Variable Example

Encoding Bldg_Type Variable:

Illustration of binary encodings (i.e., dummy variables) for a qualitative predictor.

Raw Data	TwoFmCon	Duplex	Twnhs	TwnhsE
OneFam	0	0	0	0
TwoFmCon	1	0	0	0
Duplex	0	1	0	0
Twnhs	0	0	1	0
TwnhsE	0	0	0	1

Interaction terms

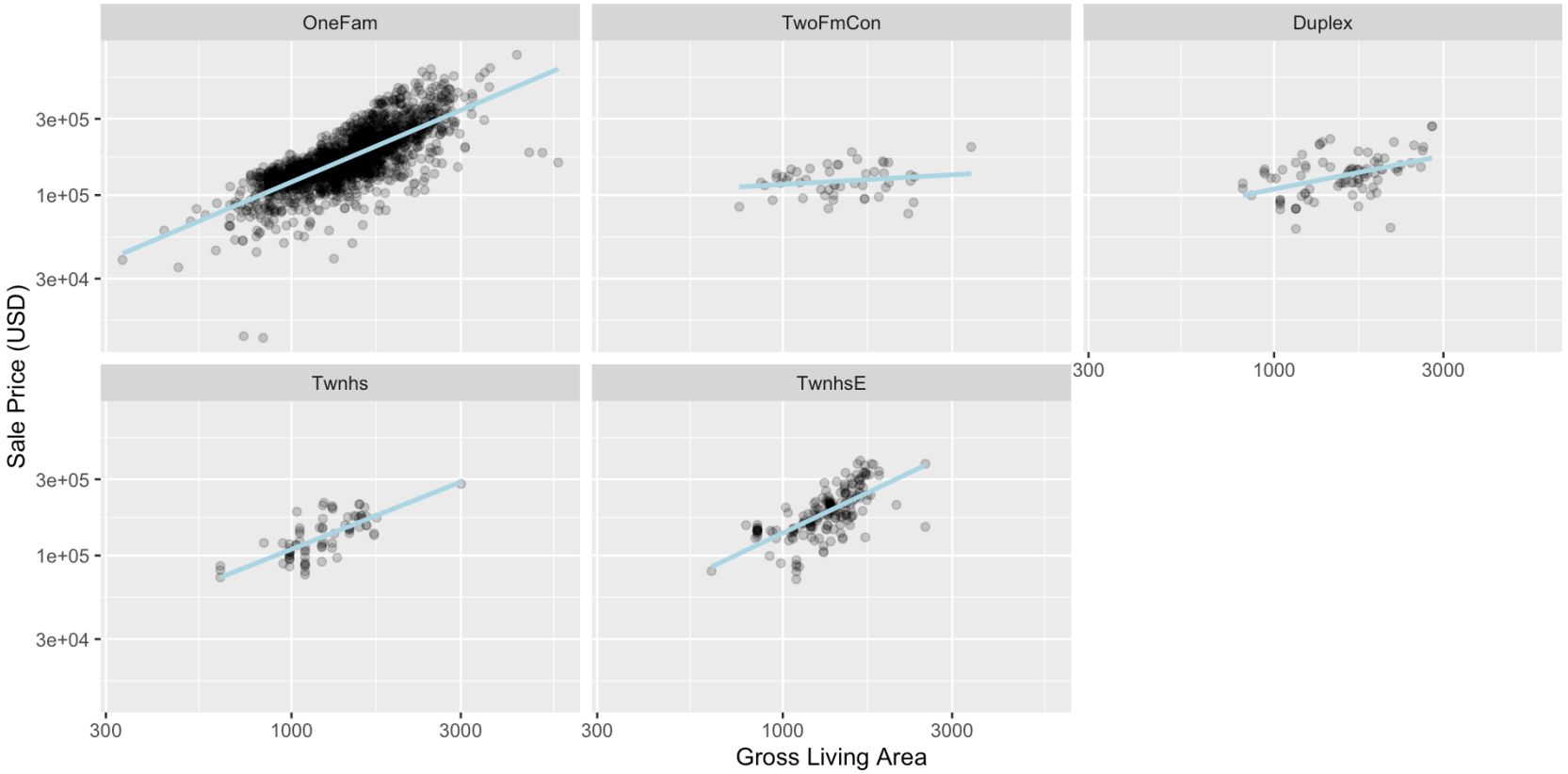
- Interaction effects involve two or more predictors.
- Such an effect occurs when one predictor has an effect on the outcome that is contingent on one or more other predictors.
- Numerically, an interaction term between predictors is encoded as their product.

Interactions in Ames

After exploring the Ames training set, we might find that the regression slopes for the gross living area differ for different building types.

```
ggplot(ames_train, aes(x = Gr_Liv_Area, y = 10^Sale_Price)) +
  geom_point(alpha = .2) +
  facet_wrap(~ Bldg_Type) +
  geom_smooth(method = lm, formula = y ~ x, se = FALSE, color = "lightblue") +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Gross Living Area", y = "Sale Price (USD)")
```

Interactions in Ames



Coding Interactions in R Formulas

```
Sale_Price ~ Neighborhood + log10(Gr_Liv_Area) + Bldg_Type +
  log10(Gr_Liv_Area):Bldg_Type
# or
Sale_Price ~ Neighborhood + log10(Gr_Liv_Area) * Bldg_Type
```

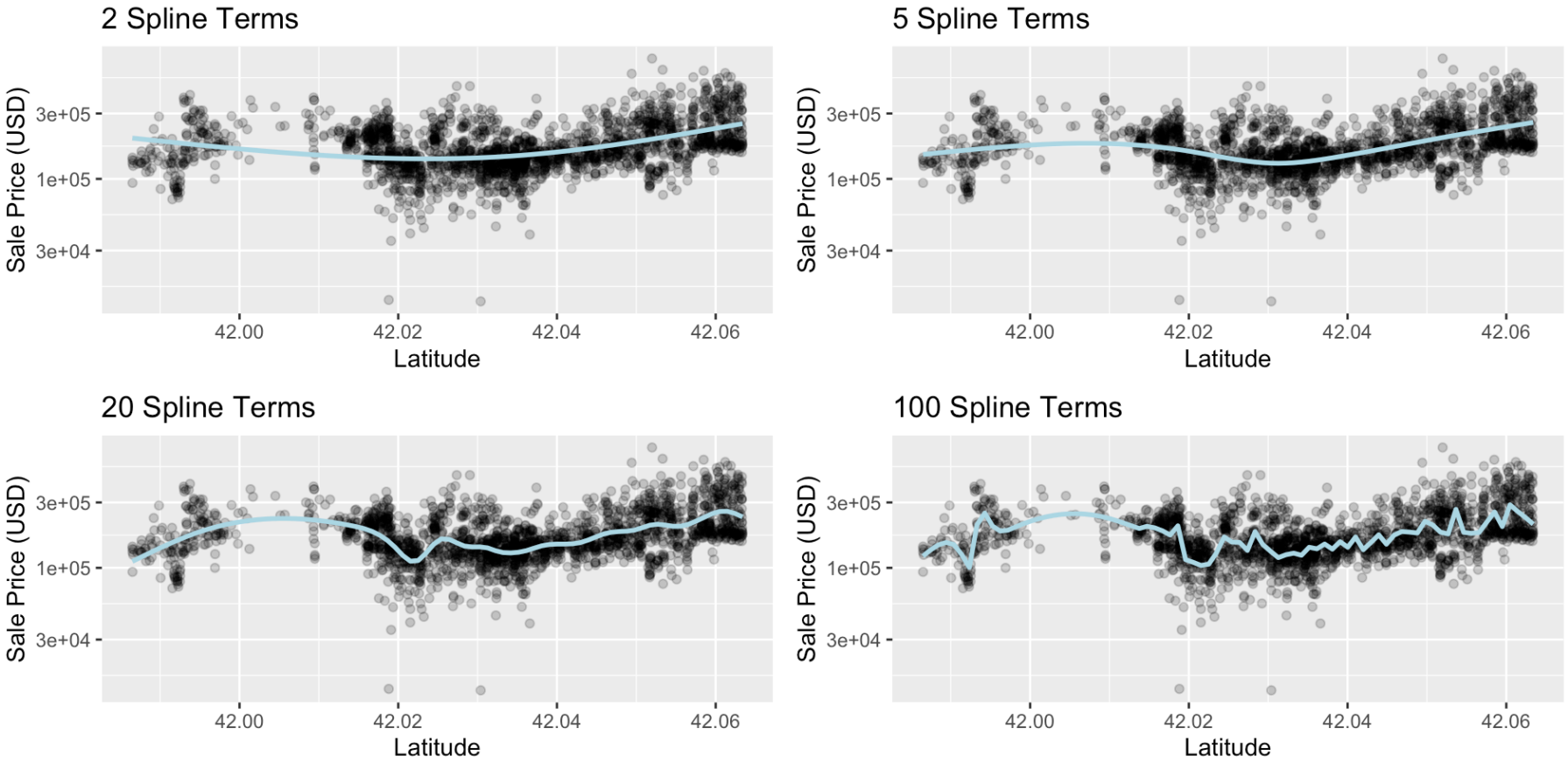
Coding Interactions Using recipes

```
simple_ames <-
  recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
         data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  step_dummy(all_nominal_predictors()) %>%
  # Gr_Liv_Area is on the log scale from a previous step
  step_interact( ~ Gr_Liv_Area:starts_with("Bldg_Type_") )
```

Spline functions

- Splines help model non-linear relationships
- Some types of predictive modeling algorithms can adaptively approximate nonlinearity during training.
- It is not uncommon to try to use a simple model, such as a linear fit, and add in specific nonlinear features for predictors that may need them
- Splines replace the existing numeric predictor with a set of columns that allow a model to emulate a flexible, nonlinear relationship.
- As more spline terms are added to the data, the capacity to nonlinearly represent the relationship increases.
- Unfortunately, it may also increase the likelihood of picking up on data trends that occur by chance (i.e., overfitting).

NonLinearity in the Ames Data



Sale price versus latitude, with trend lines using natural splines with different degrees of freedom

Adding a Spline Function to the Recipe

The `ns()` function in the `splines` package generates feature columns using functions called *natural splines*.

`step_ns` implements a spline fitting step into our model preprocessing.

```
recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type + Latitude,
         data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_interact( ~ Gr_Liv_Area:starts_with("Bldg_Type_") ) %>%
  step_ns(Latitude, deg_free = 20)
```

Feature Extraction

- *feature extraction*: Techniques creating new features from the predictors that capture the information in the broader set as a whole.
- Principal Component Analysis (PCA) tries to extract as much of the original information in the predictor set as possible using a smaller number of features.
  - Each new feature in PCA is a linear combination of the original predictors.
  - Each of the new features are uncorrelated with one another.
  - PCA can be very effective at reducing the correlation between predictors.
  - Note: PCA is only aware of the predictors; the new PCA features might not be associated with the outcome.

Applying PCA to the Ames recipe

- In the Ames data, several predictors measure size of the property, such as the total basement size (`Total_Bsmt_SF`), size of the first floor (`First_Flr_SF`), the gross living area (`Gr_Liv_Area`).
- PCA might be an option to represent these potentially redundant variables as a smaller feature set.

```
# Use a regular expression to capture house size predictors:
step_pca(matches("(SF$)|(Gr_Liv)"), num_comp = 3)
```

Note that all of these columns are measured in square feet. PCA assumes that all of the predictors are on the same scale. That's true in this case, but often this step can be preceded by `step_normalize()`, which will center and scale each column.

Alternative Feature Extraction Methods

- independent component analysis (ICA),
- non-negative matrix factorization (NNMF),
- multidimensional scaling (MDS),
- uniform manifold approximation and projection (UMAP)

Class Imbalances

Techniques for class imbalances change the class proportions in the data being given to the mode

- *Downsampling* the data keeps the minority class and takes a random sample of the majority class so that class frequencies are balanced. (`step_downsample()` in the *themis* package)
- *Upsampling* replicates samples from the minority class to balance the classes. Some techniques do this by synthesizing new samples that resemble the minority class data while other methods simply add the same minority samples repeatedly. (`step_upsample()` in the *themis* package)
- *Hybrid methods* do a combination of both. (available in the *themis* package)

Other Preprocessing Steps

A complete list of preprocessing steps is available at:

https://recipes.tidymodels.org/reference/index.html

Additionally, the framework is flexible enough to create your own recipes if needed (kind of a big lift, though)

Using Recipes to Model Ames

```
library(tidymodels)
data(ames)
ames <- mutate(ames, Sale_Price = log10(Sale_Price))

set.seed(502)
ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)

ames_rec <-
  recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type +
           Latitude + Longitude, data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_interact( ~ Gr_Liv_Area:starts_with("Bldg_Type_") ) %>%
  step_ns(Latitude, Longitude, deg_free = 20)

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

lm_wflow <-
  workflow() %>%
  add_model(lm_model) %>%
  add_recipe(ames_rec)

lm_fit <- fit(lm_wflow, ames_train)
```