Deeper Dive into Tidymodels for Machine Learning in R

John Lewis

2020/04/08 (updated: 2020-08-13)

Applied Tidymodeling Webinar

Modeling in the Tidyverse

What we will cover:

1) Continue working with the Ames data/model but add new predictors

2) With added discussion of the bias-variance tradeoff

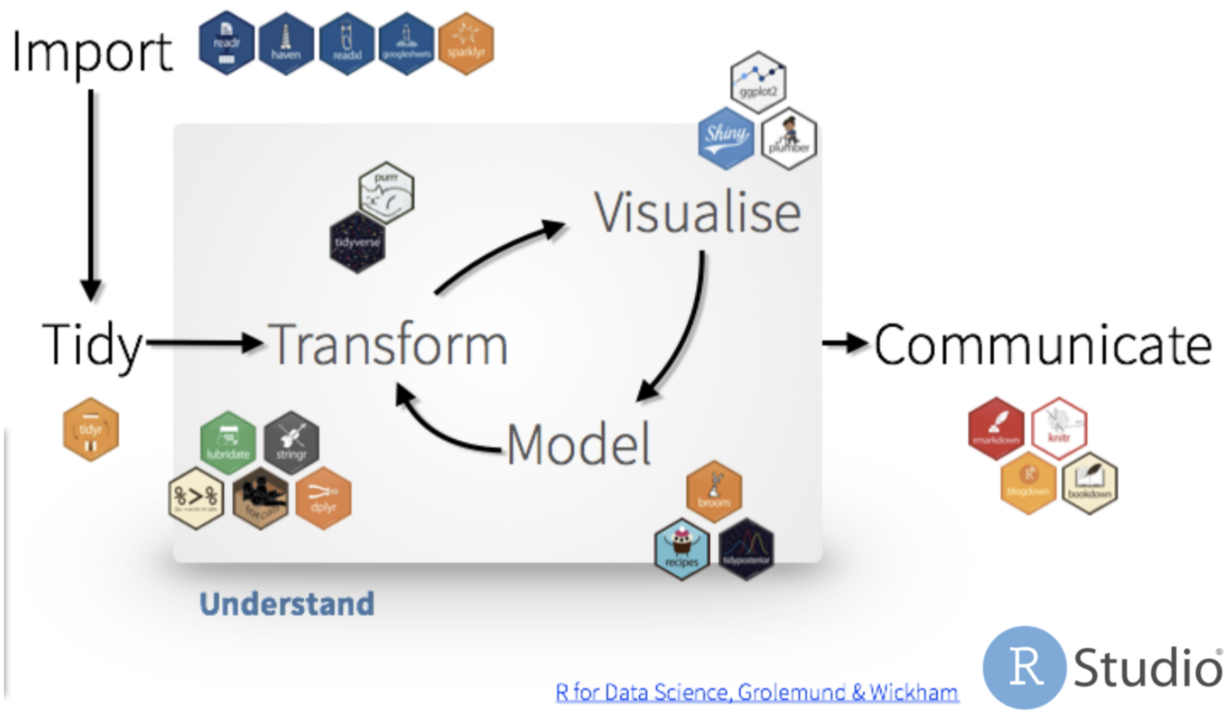
3) Add & compare final results from kNN model, a random forest model and glm (lasso) model

4) Discuss 'overfitting' in the context of fitting these three models

5) Actually work on a ML classification problem in real time

What is tidymodels?

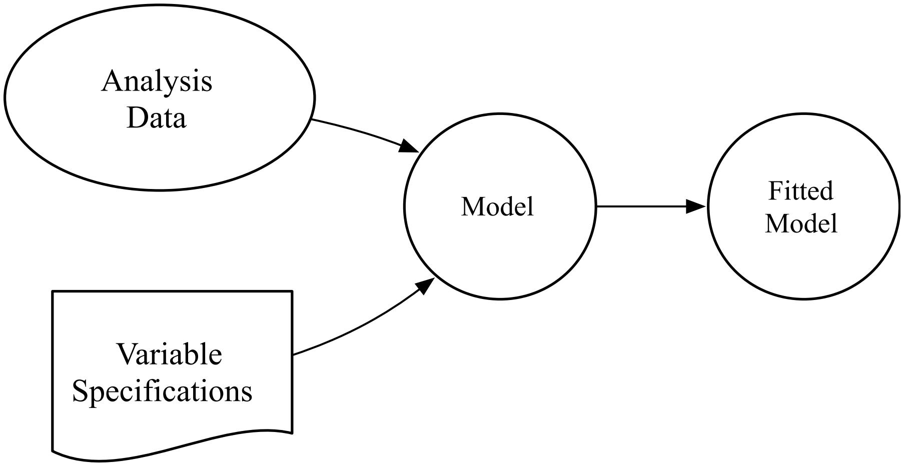
Analysis Workflow



A reminder-please see the material in my webinar presentation. That workflow will be a template for how the following code structure proceeds. However, the examples included here are more extensive.

```
library(tidyverse)
## -- Attaching packages ---------------------------------------------
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts ------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
library(tidymodels)
## -- Attaching packages ---------------------------------------------
## v broom     0.7.0      v recipes   0.1.13
## v dials     0.0.8      v rsample   0.0.7
## v infer     0.5.3      v tune      0.1.1
## v modeldata 0.0.2      v workflows 0.1.2
## v parsnip   0.1.3      v yardstick 0.0.7
## -- Conflicts ------------------------------------------------------
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

Modeling Worflow



Source: Max Kuhn & Davis Vaughan
https://github.com/rstudio-conf-2020/applied-ml/blob/master/Part_1.pdf

Exploring the Data

```r
data(ames, package = "modeldata") #load the data
nrow(ames)
```

```
## [1] 2930
```

```r
ncol(ames)
```

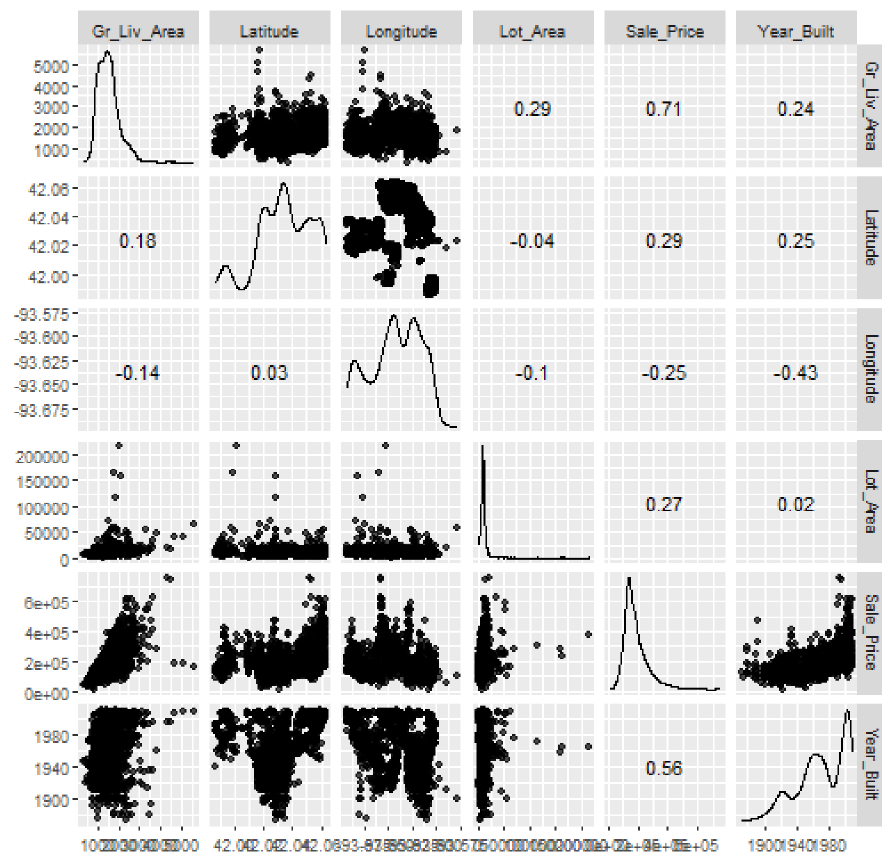
```
## [1] 74
```

```r
DT::datatable(head(ames,15),
  fillContainer = TRUE, options = list(searching = FALSE, pageLength
```

Show 81 entries

	MS_SubClass	MS_Zoning	Lot_Frontage	Lot_Area	Street
1	One_Story_1946_and_Newer_All_Styles	Residential_Low_Density			
2	One_Story_1946_and_Newer_All_Styles	Residential_High_Density			

```r
ames_df <- ames%>%
      select(Longitude,Latitude,Gr_Liv_Area,Lot_Area,Neighborhood, Ye
# use library(GGally) for the accompanying plot
GGally::ggscatmat(ames_df, alpha = 0.7)
```



Now using `rsample`

Partitioning - `rsample`

`rsample`

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

Data Partitioning for Ames

Use rsample

```
set.seed(123)
ames_split <- initial_split(ames_df, prop = .70)#prop defines the amo
ames_train <- training(ames_split)# our training data
ames_train%>%
  slice(1:10)
## # A tibble: 10 x 7
##    Longitude Latitude Gr_Liv_Area Lot_Area Neighborhood Year_Buil
##        <dbl>    <dbl>       <int>    <int> <fct>             <int
##  1     -93.6     42.1        1656    31770 North_Ames         196
##  2     -93.6     42.1        1329    14267 North_Ames         195
##  3     -93.6     42.1        2110    11160 North_Ames         196
##  4     -93.6     42.1        1629    13830 Gilbert            199
##  5     -93.6     42.1        1604     9978 Gilbert            199
##  6     -93.6     42.1        1338     4920 Stone_Brook        200
##  7     -93.6     42.1        1655    10000 Gilbert            199
##  8     -93.6     42.1        1187     7980 Gilbert            199
##  9     -93.6     42.1        1341    10176 Gilbert            199
## 10     -93.6     42.1        1502     6820 Stone_Brook        198
```

This is just an example to show what the test data set looks like-there is no need to do this step

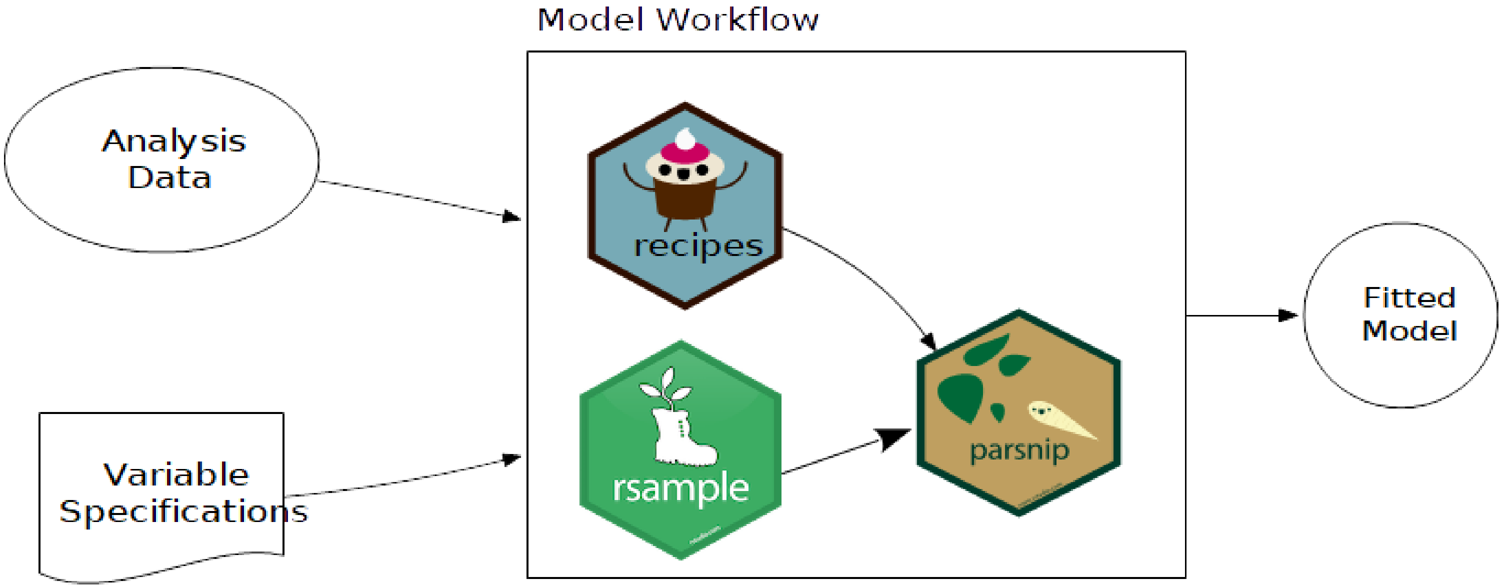
```
ames_test <- testing(ames_split)
ames_test %>%
slice(1:10)
```

```
## # A tibble: 10 x 7
##    Longitude Latitude Gr_Liv_Area Lot_Area Neighborhood    Year_Built Sale
##        <dbl>    <dbl>       <int>    <int> <fct>                <int>
##  1     -93.6     42.1         896    11622 North_Ames            1961
##  2     -93.6     42.1        1280     5005 Stone_Brook           1992
##  3     -93.6     42.1        1616     5389 Stone_Brook           1995
##  4     -93.6     42.1        1804     7500 Gilbert               1999
##  5     -93.6     42.1        1465     8402 Gilbert               1998
##  6     -93.6     42.1        3279    53504 Stone_Brook           2003
##  7     -93.6     42.1        1856    11394 Stone_Brook           2010
##  8     -93.6     42.1        1004    11241 North_Ames            1970
##  9     -93.6     42.1        1092     1680 Briardale             1971
## 10     -93.7     42.1        1940    10159 Northridge_Hei~       2009
```

Cross validation (cv) statement for later use

```
ames_cv <- vfold_cv(ames_train)# creating the object in order to do c
```

Tidymodels workflow



Source: Max Kuhn & Davis Vaughan https://github.com/rstudio-conf-2020/applied-ml/blob/master/Part_3.pdf

Feature Engineering

`recipes`

recipes

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

Defining our `recipe()`

- Our recipe is the plan of action-apply a formula
- We can, also, add `step_*()`s to our recipe
- The following is an example of a formula specification

```
# our formula
ames_ex <- recipe(Sale_Price ~ Gr_Liv_Area + Lot_Area + Longitude +
                  Latitude + Neighborhood + Year_Built,
                  data = ames_train) #specify our formula
```

Notice that in this model we have added 4 new predictor variables compared to the other webinar example.

Some preprocessing steps

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

preprocessing steps are:

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

Just for information sake - these function hardly used anymore since the `workflow` package

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

Preprocessing new data

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

Final feature engineering recipe for Ames

```r
#use recipes
mod_rec <- recipe(Sale_Price ~ Gr_Liv_Area + Lot_Area + Longitude +
                  Latitude + Neighborhood + Year_Built,
                  data = ames_train) %>%
  step_log(Sale_Price, base = 10) %>%
  step_YeoJohnson(Lot_Area, Gr_Liv_Area) %>%
  step_other(Neighborhood, threshold = 0.05) %>%
  step_dummy(all_nominal())%>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric())

summary(mod_rec)
```

```
## # A tibble: 7 x 4
##   variable     type    role      source
##   <chr>        <chr>   <chr>     <chr>
## 1 Gr_Liv_Area  numeric predictor original
## 2 Lot_Area     numeric predictor original
## 3 Longitude    numeric predictor original
## 4 Latitude     numeric predictor original
## 5 Neighborhood nominal predictor original
## 6 Year_Built   numeric predictor original
## 7 Sale_Price   numeric outcome   original
```

What is `parsnip`?

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

Example

```r
#use parsnip
knn_mod <-
  nearest_neighbor(neighbors=tune()) %>%
  set_engine("kknn")%>%
  set_mode("regression")

knn_mod
```

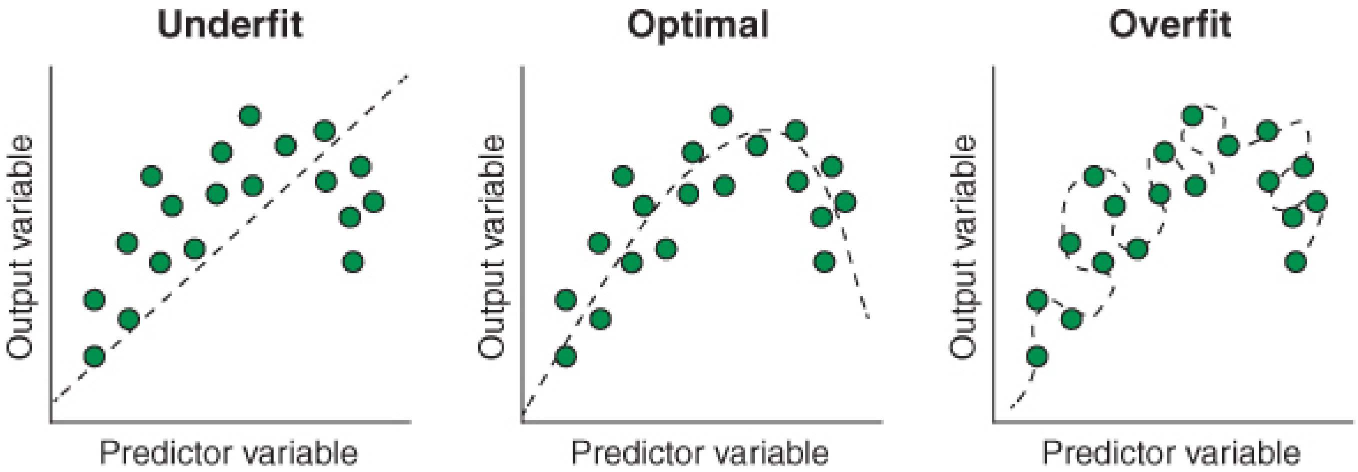
```
## K-Nearest Neighbor Model Specification (regression)
##
## Main Arguments:
##   neighbors = tune()
##
## Computational engine: kknn
```

Use workflow package

Construct a workflow that combines your recipe and your model

```
ml_wflow <-
  workflow() %>%
  add_recipe(mod_rec) %>%
  add_model(knn_mod)
```

A problem occurs when we start fitting the training data where the learning algorithm can overfit or underfit the data



Source:Rhys, H., 2020: Machine Learning with R, the tidyverse and mlr, Manning, Shelter Island

Underfitting and Overfitting

-models that "underfit" have high bias

reasons:

1) model used is too simple

2) features used are not informative enough

-models that "overfit" have high variance

1) model is too complex for the data

2) too many features in relation to the training data

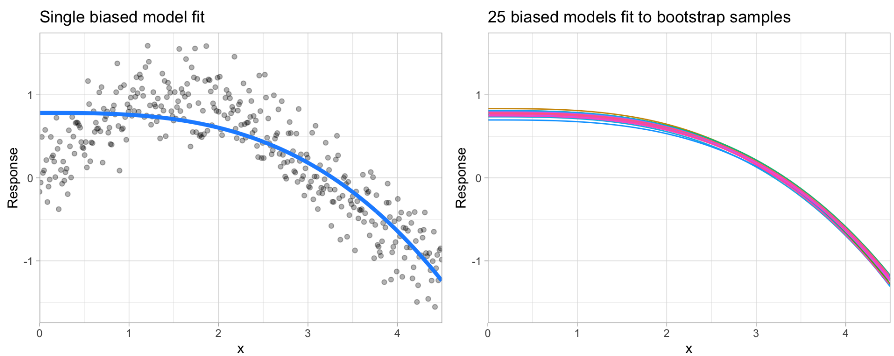
Some definitions

***Bias* is the difference between the average prediction of the model and the correct value you are trying to predict. It provides information on how well the model is describing the underlying structure of the data.**

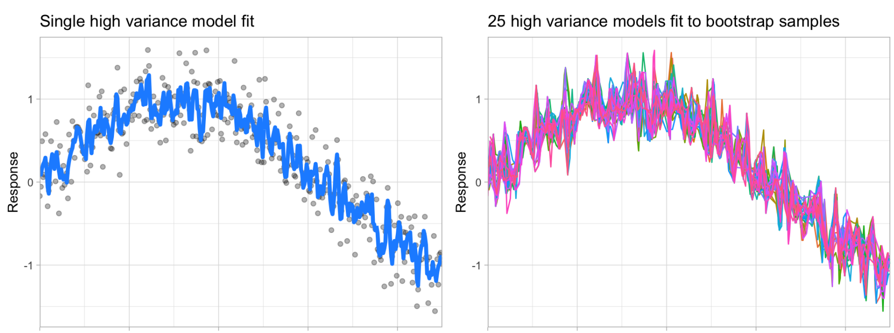
***Variance* is the error due to the variability in the model's predictive ability for a given point. With high variance there is the risk of overfitting. There may be good performance on the training data but the model will not generalize well to the test (unseen) data.**

Source:Boehmke, B. and Greenwell, B., 2020:Hands-On Machine Learning with R, CRC Press, NY.

Underfit



Overfit



Source:Boehmke, B. and Greenwell, B., 2020:Hands-On Machine Learning with R, CRC Press, NY.

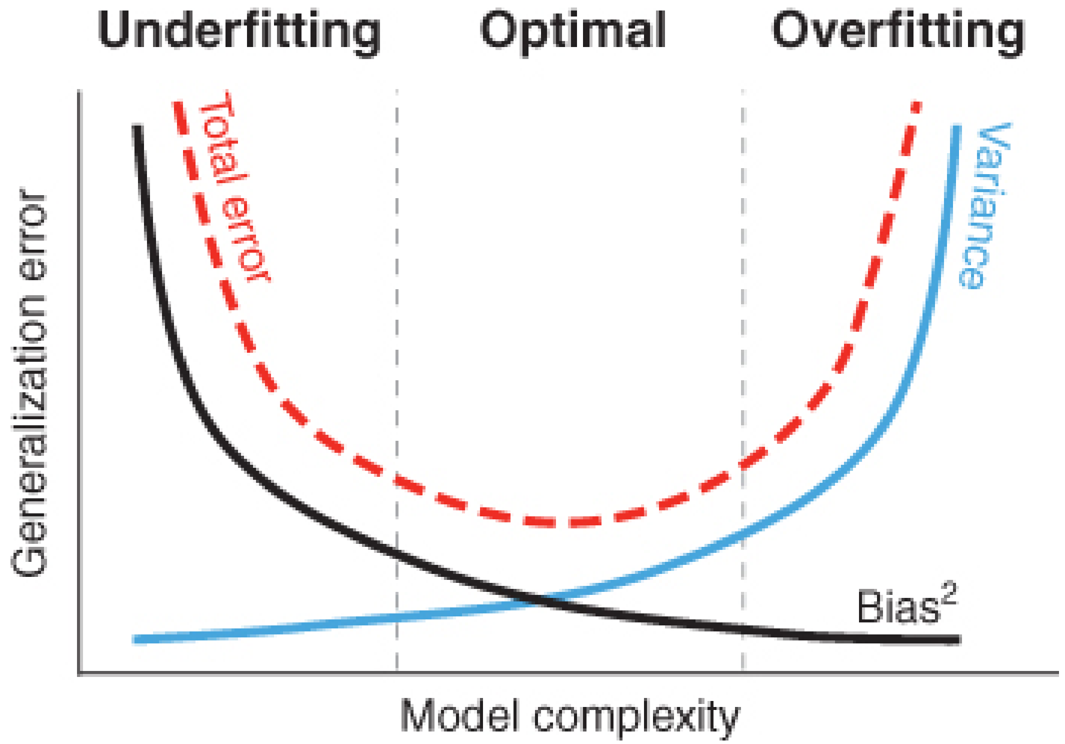
Dealing with Overfitting

Use **regularization** - this encompasses methods that forces the learning algorithm to build less complex models and significantly reduces the variance

This is known as the:

Bias-Variance Tradeoff

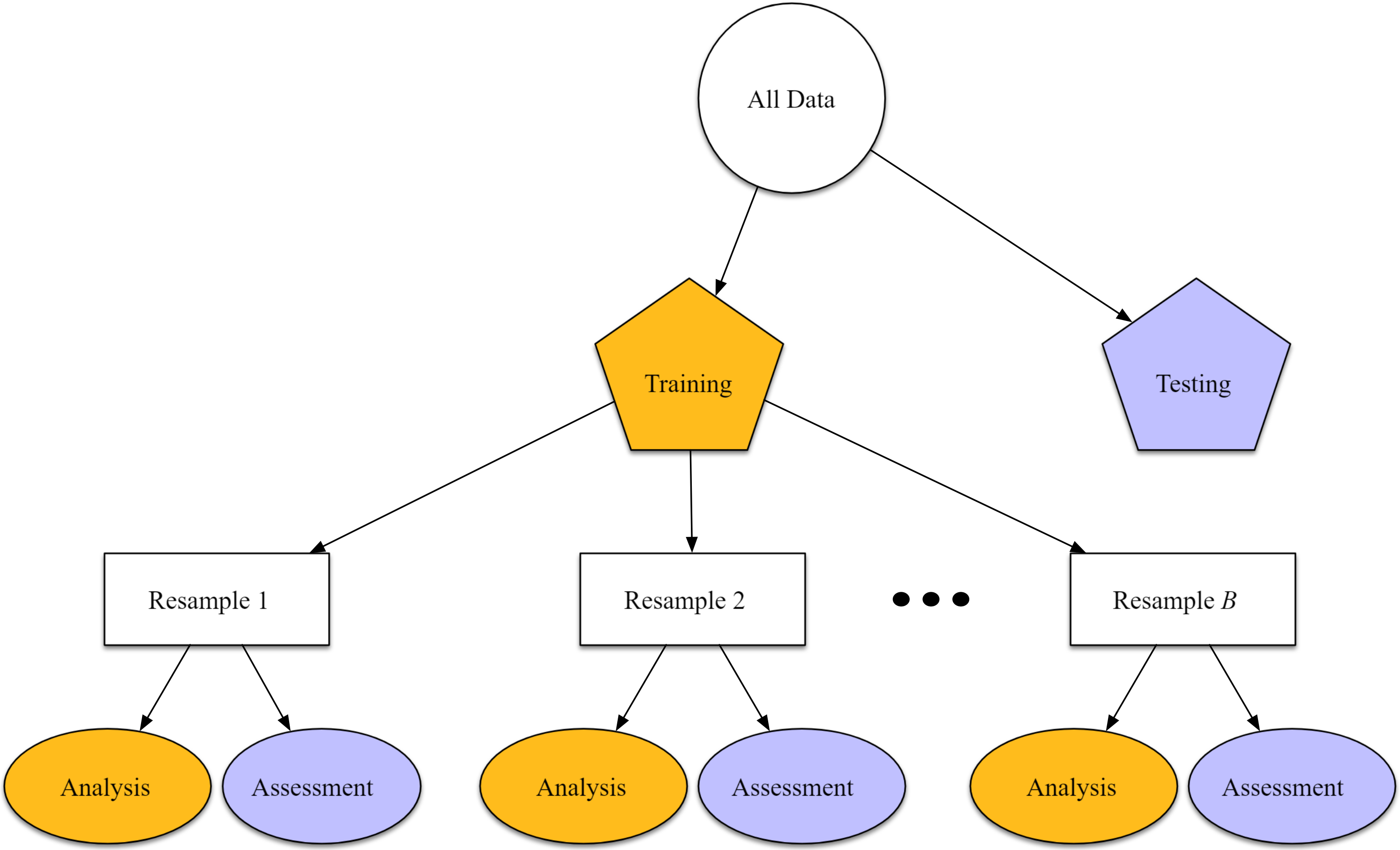
Bias-Variance Tradeoff



Hyperparameter tuning is used to find the "optimum" value for this tradeoff

Source:Rhys, H., 2020: Machine Learning with R, the tidyverse and mlr, Manning, Shelter Island

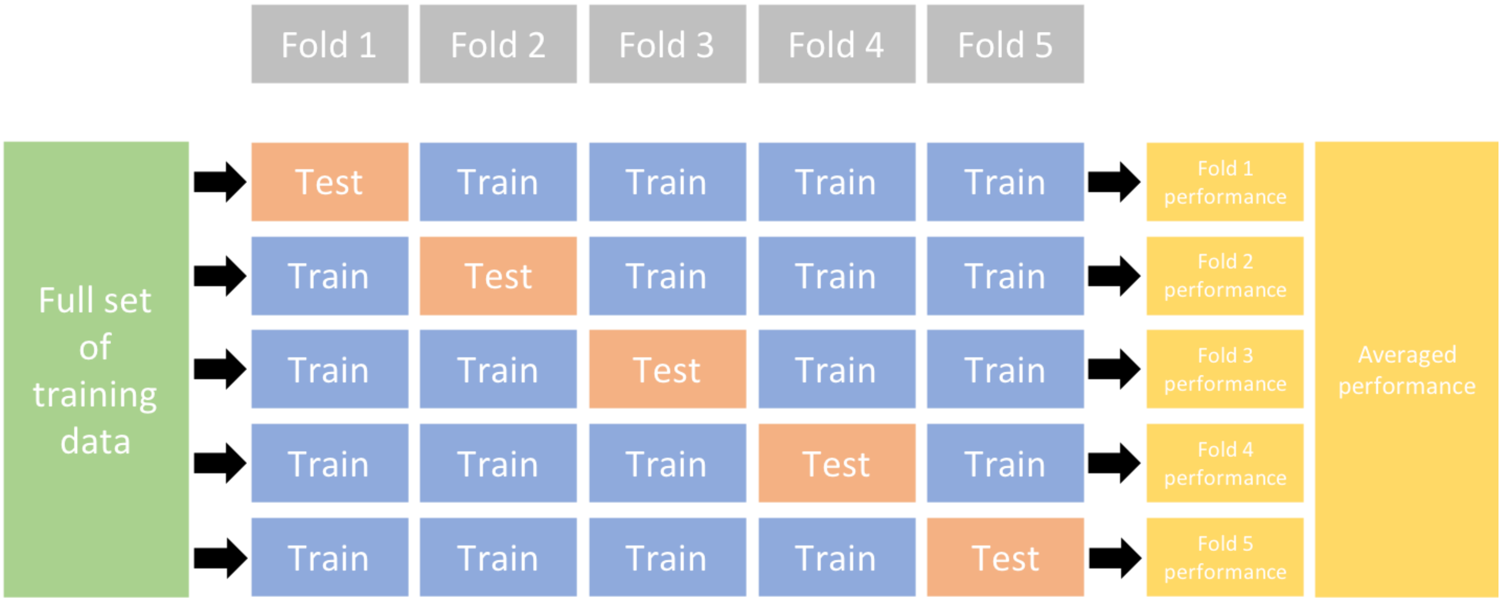
Diagram of resampling scheme-cross validation(cv-folds) for obtaining the hold-out data sets



source:https://rviews.rstudio.com/2020/04/21/the-case-for-tidymodels/

In cross validation the model, trained on the analysis set, is applied to the assessment set to generate predictions, and performance statistics are computed based on those predictions.

In this example, 10-fold CV moves iteratively through the folds and leaves a different 10% out each time for model assessment. At the end of this process, there are 10 sets of performance statistics that were created on 10 data sets that were not used in the modeling process.



Cross Validation & tuning the knn model

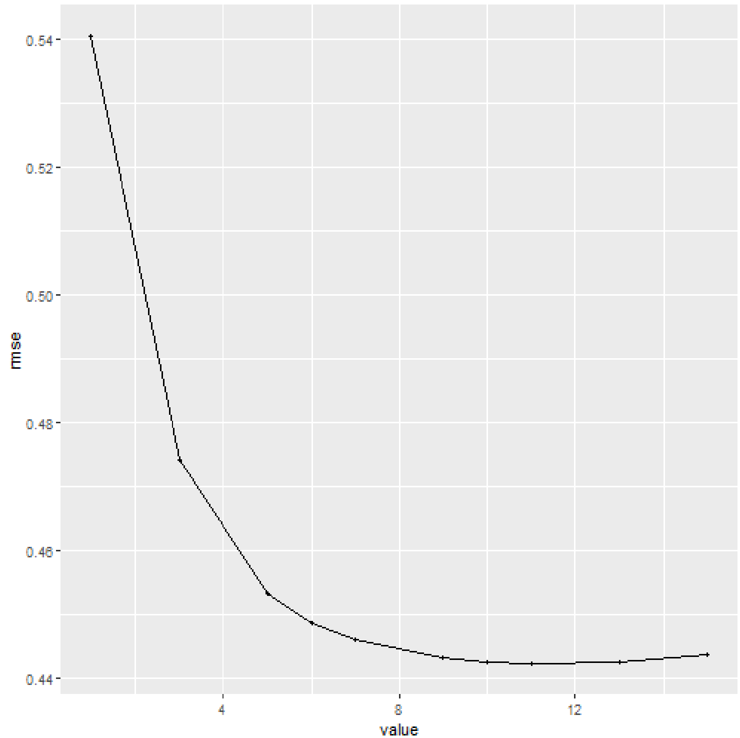
```
set.seed(456)
ml_wflow_tune <-
  ml_wflow %>%
  tune_grid(resamples = ames_cv, # cv object
                 grid = 10, # grid values
                 metrics = metric_set(rmse,rsq))#performance metric o

ml_wflow_tune
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits             id     .metrics          .notes
##    <list>             <chr>  <list>            <list>
##  1 <split [1.8K/206]> Fold01 <tibble [20 x 5]> <tibble [0 x 1]>
##  2 <split [1.8K/205]> Fold02 <tibble [20 x 5]> <tibble [0 x 1]>
##  3 <split [1.8K/205]> Fold03 <tibble [20 x 5]> <tibble [0 x 1]>
##  4 <split [1.8K/205]> Fold04 <tibble [20 x 5]> <tibble [0 x 1]>
##  5 <split [1.8K/205]> Fold05 <tibble [20 x 5]> <tibble [0 x 1]>
##  6 <split [1.8K/205]> Fold06 <tibble [20 x 5]> <tibble [0 x 1]>
##  7 <split [1.8K/205]> Fold07 <tibble [20 x 5]> <tibble [0 x 1]>
##  8 <split [1.8K/205]> Fold08 <tibble [20 x 5]> <tibble [0 x 1]>
##  9 <split [1.8K/205]> Fold09 <tibble [20 x 5]> <tibble [0 x 1]>
```

Resampling allows us to simulate how well our model will perform on new data, and the test set acts as the final, unbiased check for our model's performance.

```r
# Plot tuning performance results over iterations

autoplot(ml_wflow_tune, metric = "rmse") #use quick plot - autoplot :
```



Model Evalutation: `yardstick`

`yardstick`

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

Validation of our model

```
# Collecting the metrics

res_kn  <- ml_wflow_tune %>%
  collect_metrics()
res_kn
## # A tibble: 20 x 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 rmse    standard   0.540    10  0.0263 Model01
##  2         1 rsq     standard   0.724    10  0.0240 Model01
##  3         3 rmse    standard   0.474    10  0.0240 Model02
##  4         3 rsq     standard   0.776    10  0.0196 Model02
##  5         5 rmse    standard   0.453    10  0.0238 Model03
##  6         5 rsq     standard   0.794    10  0.0183 Model03
##  7         6 rmse    standard   0.449    10  0.0236 Model04
##  8         6 rsq     standard   0.797    10  0.0179 Model04
##  9         7 rmse    standard   0.446    10  0.0235 Model05
## 10         7 rsq     standard   0.800    10  0.0176 Model05
## 11         9 rmse    standard   0.443    10  0.0233 Model06
## 12         9 rsq     standard   0.802    10  0.0171 Model06
## 13        10 rmse    standard   0.443    10  0.0232 Model07
## 14        10 rsq     standard   0.803    10  0.0169 Model07
## 15        11 rmse    standard   0.442    10  0.0231 Model08
```

```r
# Select the best parameters

best_params <-
  ml_wflow_tune %>%
  select_best(metric = "rmse")
```

Refit using the entire training dataset

```r
 # use of tune and parsnip again

ames_reg_res <-
  ml_wflow %>%
  finalize_workflow(best_params)

#make sure you use the initial split data
ames_wfl_fit <- ames_reg_res %>%
  last_fit(ames_split)
```

Getting the final evaluation metrics

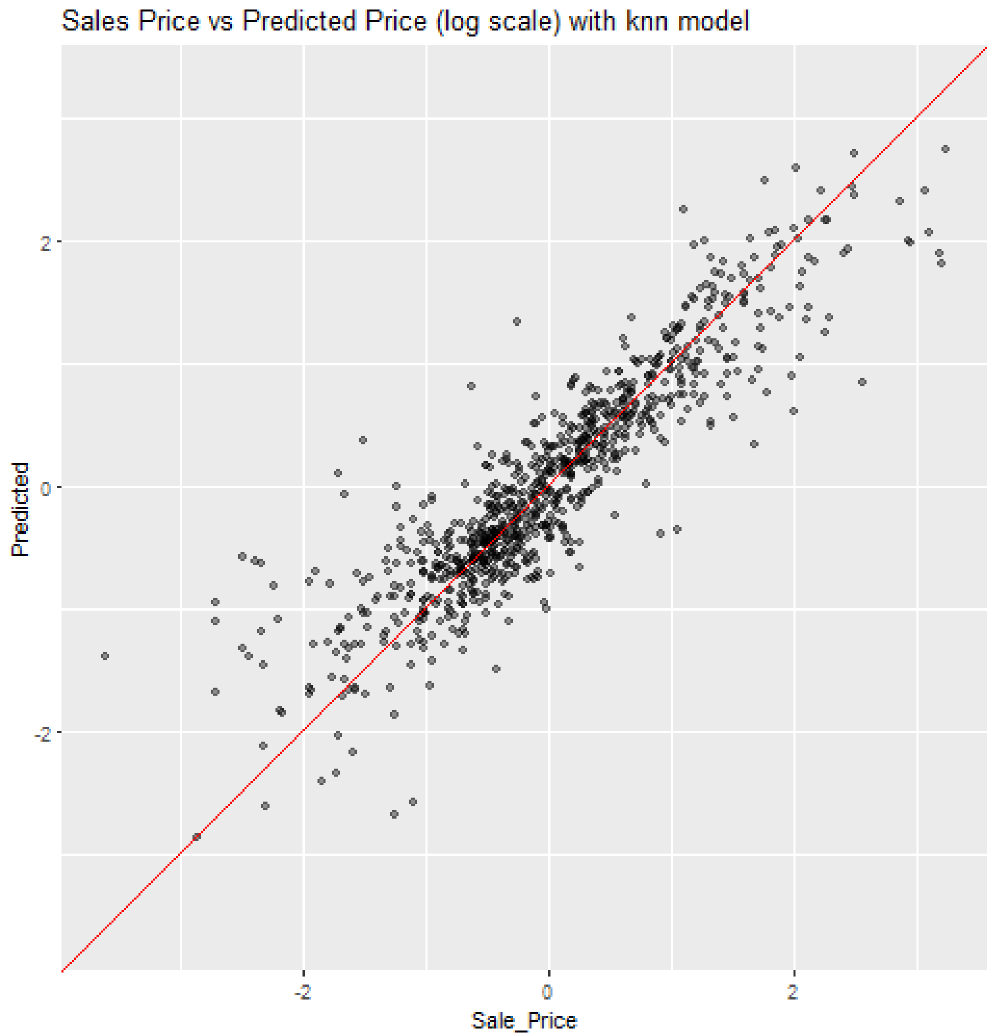
```r
# Since we fitted on the train and test data the
# metrics have been evaluated for the test data

knn_test_performance <- ames_wfl_fit %>% collect_metrics()
knn_test_performance
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.434
## 2 rsq     standard       0.809
```

```r
#extract the test set predictions themselves
test_predictions <- ames_wfl_fit %>% collect_predictions()
test_predictions
```

```
## # A tibble: 879 x 4
##    id                .pred  .row Sale_Price
##    <chr>             <dbl> <int>      <dbl>
##  1 train/test split -0.796     2     -1.13
##  2 train/test split  0.315     8      0.345
##  3 train/test split  0.993     9      0.862
##  4 train/test split  0.332    10      0.313
##  5 train/test split  0.162    13      0.199
##  6 train/test split  2.32     16      2.87
##  7 train/test split  1.87     18      2.11
##  8 train/test split -0.541    24     -0.269
##  9 train/test split -1.29     31     -1.11
## 10 train/test split  1.46     39      2.12
## # ... with 869 more rows
```


Sales Price vs Predicted Price (log scale) with knn model
Predicted
2
0
-2
-2
0
2
Sale_Price

Now let's run a random forest model

Start with parsnips-preprocessing already done

```r
# random forest model-tuning mtry & min_n, trees=500
rf_model <- rand_forest(mtry=tune(), min_n=tune(), trees=500)%>%
  set_mode("regression") %>%
  set_engine("ranger")
```

Create a workflow object since we are changing the model specifications

```
rf_wflow <-
  workflow() %>%
  add_recipe(mod_rec) %>%
  add_model(rf_model)
```

Develope grid for hyperparameter search

This is a 2d searching over a predefined parameter space

```
set.seed(522)
rf_grid <-
  expand_grid(mtry = seq(2, 14, length=7),
                          min_n = c(1,2,3,4,5))
```

For parallel processing

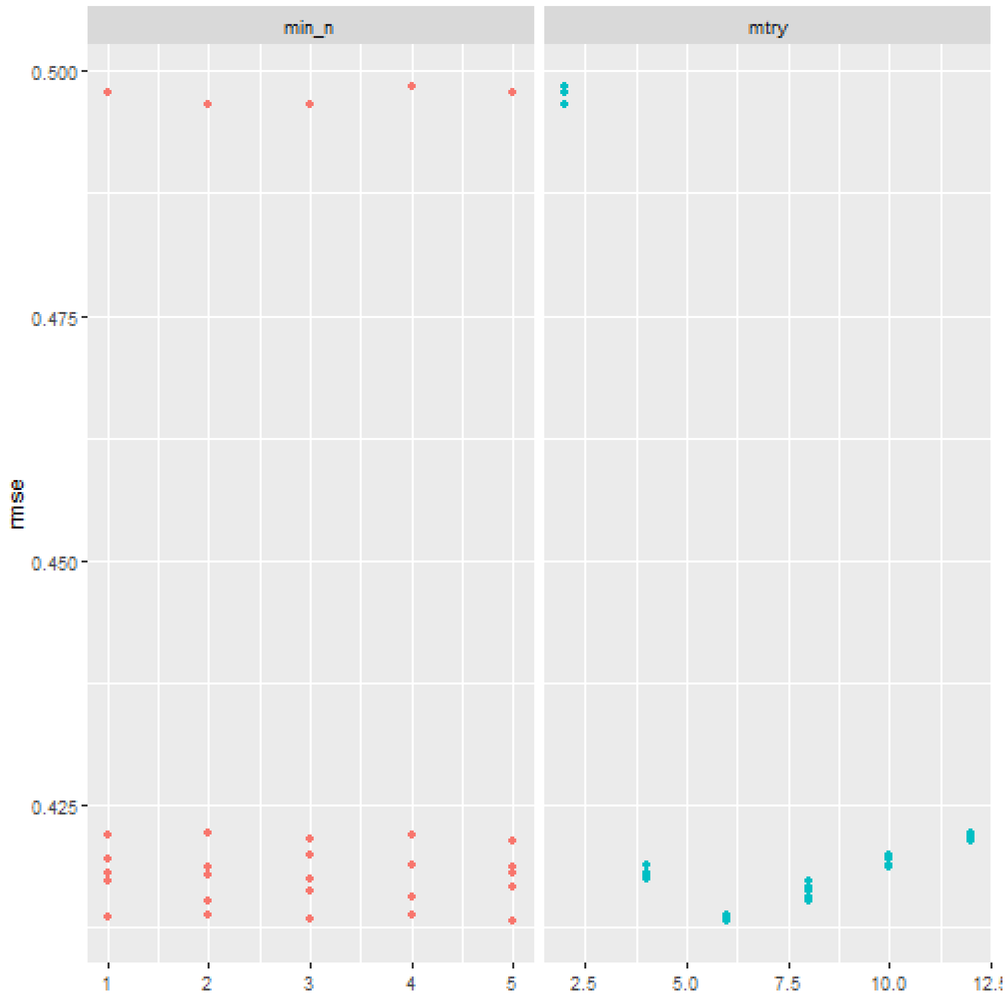
```
doParallel::registerDoParallel()
```

Find best tuned model using rf_grid

```
set.seed(123)

rf_wflow_tune <-
  rf_wflow %>%
  tune_grid(resamples = ames_cv,
                    grid = rf_grid,
                    metrics = metric_set(rmse,rsq))
```

Plotting results of tuning hyperparameters



Select best parameters

```
best_params <-
  rf_wflow_tune %>%
  select_best(metric = "rmse")


## Refit using the entire dataset

rf_ames_reg_res <-
  rf_wflow %>%
  finalize_workflow(best_params)

#make sure you use the initial split data
rf_ames_wfl_fit <- rf_ames_reg_res %>%
  last_fit(ames_split)
```

Since we fitted on the train and then test data the metrics, now, have been evaluated on the test data

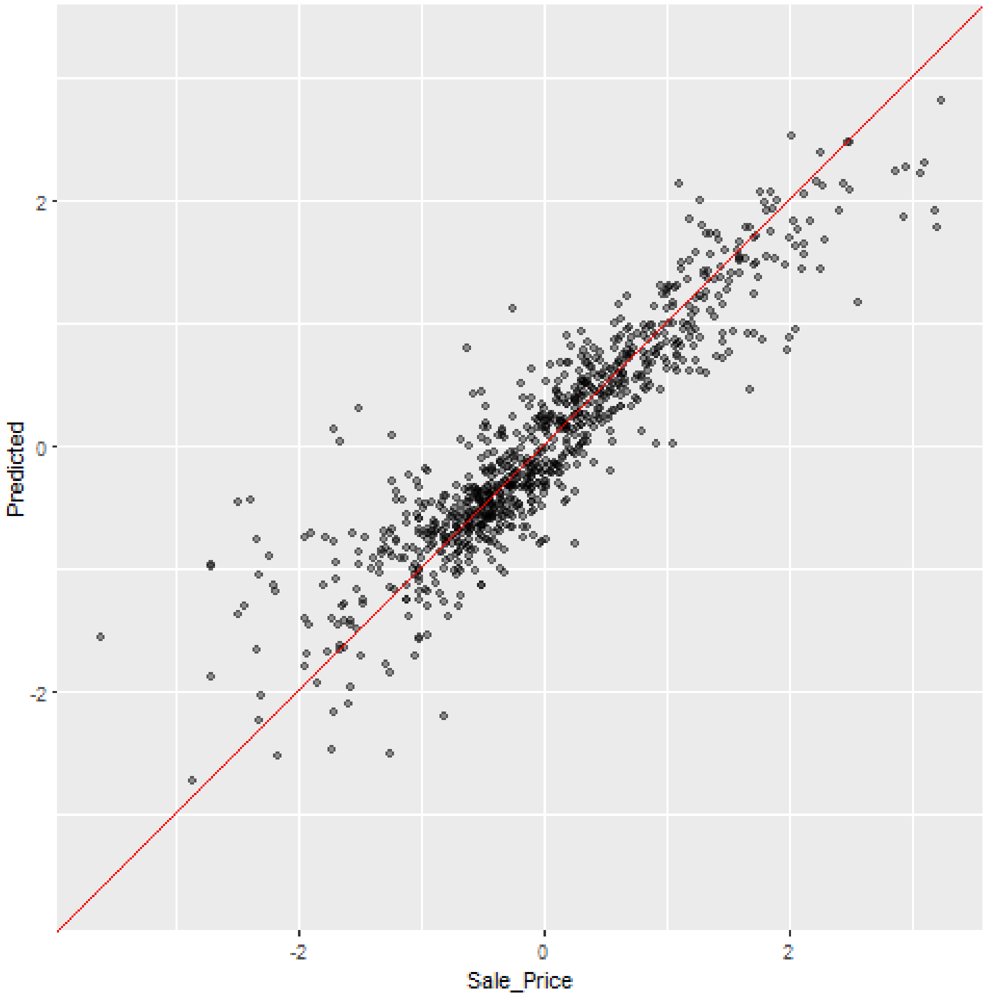
```
rf_test_performance <- rf_ames_wfl_fit %>% collect_metrics()
rf_test_performance
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.416
## 2 rsq     standard       0.825
```

Extract the test set predictions themselves

```
rf_test_predictions <- rf_ames_wfl_fit %>% collect_predictions()
rf_test_predictions
## # A tibble: 879 x 4
##    id                .pred  .row Sale_Price
##    <chr>             <dbl> <int>      <dbl>
##  1 train/test split -0.570     2     -1.13
##  2 train/test split  0.250     8      0.345
##  3 train/test split  0.972     9      0.862
##  4 train/test split  0.399    10      0.313
##  5 train/test split  0.120    13      0.199
##  6 train/test split  2.24     16      2.87
##  7 train/test split  1.65     18      2.11
##  8 train/test split -0.310    24     -0.269
##  9 train/test split -1.26     31     -1.11
## 10 train/test split  1.56     39      2.12
## # ... with 869 more rows
```

Sales Price vs Predicted Price (log scale) with Random Forest



The third model we'll run we be a 'lasso' model using glmnet

Start again with `parsnips`

```r
#model specification-using glmnet
# tuning hyperparameters-penalty. Setting mixture equal to 1
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet") %>%
  set_mode("regression")
```

Workflow

```
lasso_rec <- mod_rec #using same recipe as before
# workflow
lasso_wf <- workflow() %>%
  add_recipe(lasso_rec) %>%
  add_model(lasso_spec)
```

```
lasso_wf
## == Workflow =================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor -------------------------------------------------------------
## 6 Recipe Steps
##
## * step_log()
## * step_YeoJohnson()
## * step_other()
## * step_dummy()
## * step_zv()
## * step_normalize()
##
## -- Model --------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
## Computational engine: glmnet
```

Tuning hyperparameters

setting the grid for tuning then tune the model

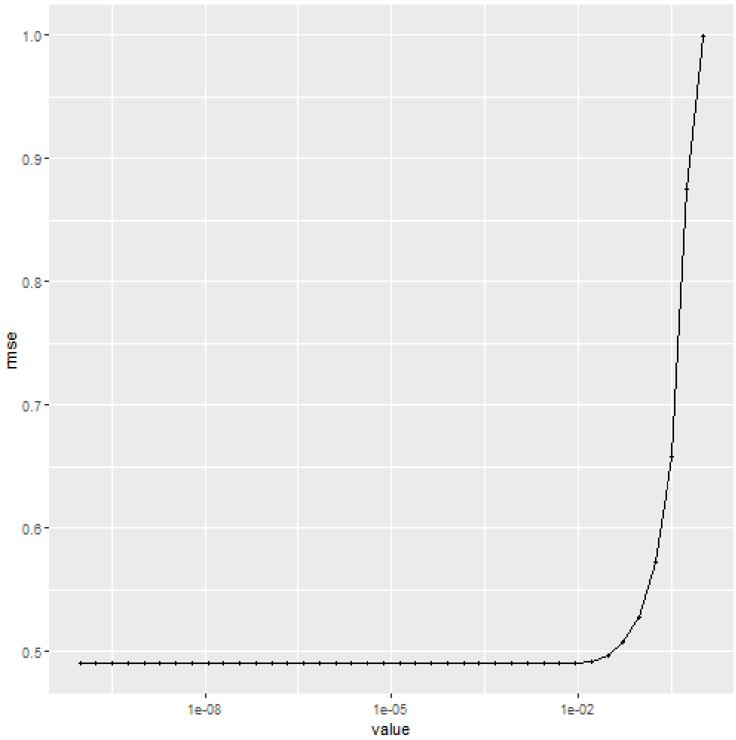
```
lambda_grid <- grid_regular(penalty(), levels = 40)

set.seed(123)

lasso_tune <- tune_grid(
  lasso_wf,
  resamples = ames_cv,
  grid = lambda_grid,
  metrics = metric_set(rmse, rsq)
)
```

Plot tuning performance results over iterations

```
autoplot(lasso_tune, metric = "rmse")
```



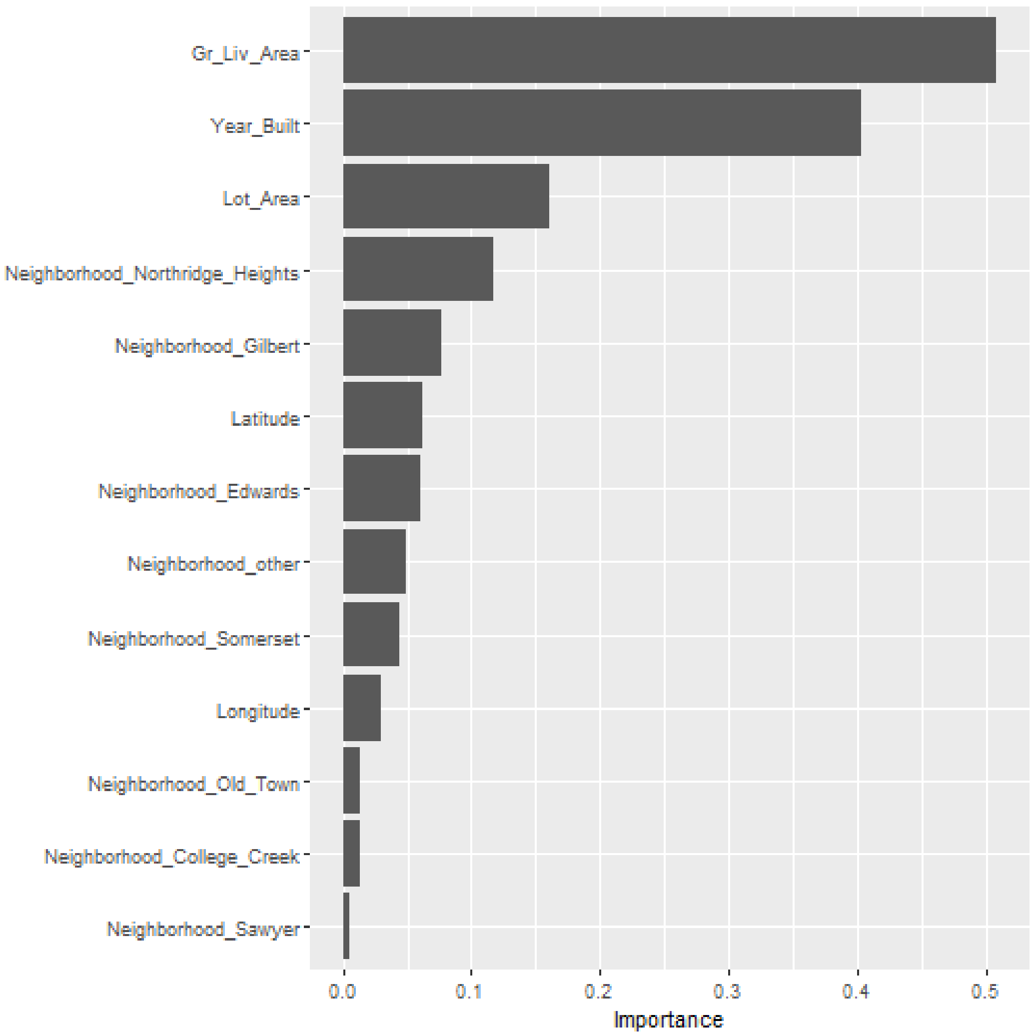
Collected metrics

```
## # A tibble: 80 x 7
##      penalty .metric .estimator  mean     n std_err .config
##        <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 1.00e-10 rmse    standard   0.491    10  0.0193 Model01
##  2 1.00e-10 rsq     standard   0.758    10  0.0134 Model01
##  3 1.80e-10 rmse    standard   0.491    10  0.0193 Model02
##  4 1.80e-10 rsq     standard   0.758    10  0.0134 Model02
##  5 3.26e-10 rmse    standard   0.491    10  0.0193 Model03
##  6 3.26e-10 rsq     standard   0.758    10  0.0134 Model03
##  7 5.88e-10 rmse    standard   0.491    10  0.0193 Model04
##  8 5.88e-10 rsq     standard   0.758    10  0.0134 Model04
##  9 1.06e- 9 rmse    standard   0.491    10  0.0193 Model05
## 10 1.06e- 9 rsq     standard   0.758    10  0.0134 Model05
## # ... with 70 more rows
```

Inspection of the final `lasso` model

```
## == Workflow ==================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------
## 6 Recipe Steps
##
## * step_log()
## * step_YeoJohnson()
## * step_other()
## * step_dummy()
## * step_zv()
## * step_normalize()
##
## -- Model -----------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = 1e-10
##   mixture = 1
##
## Computational engine: glmnet
```

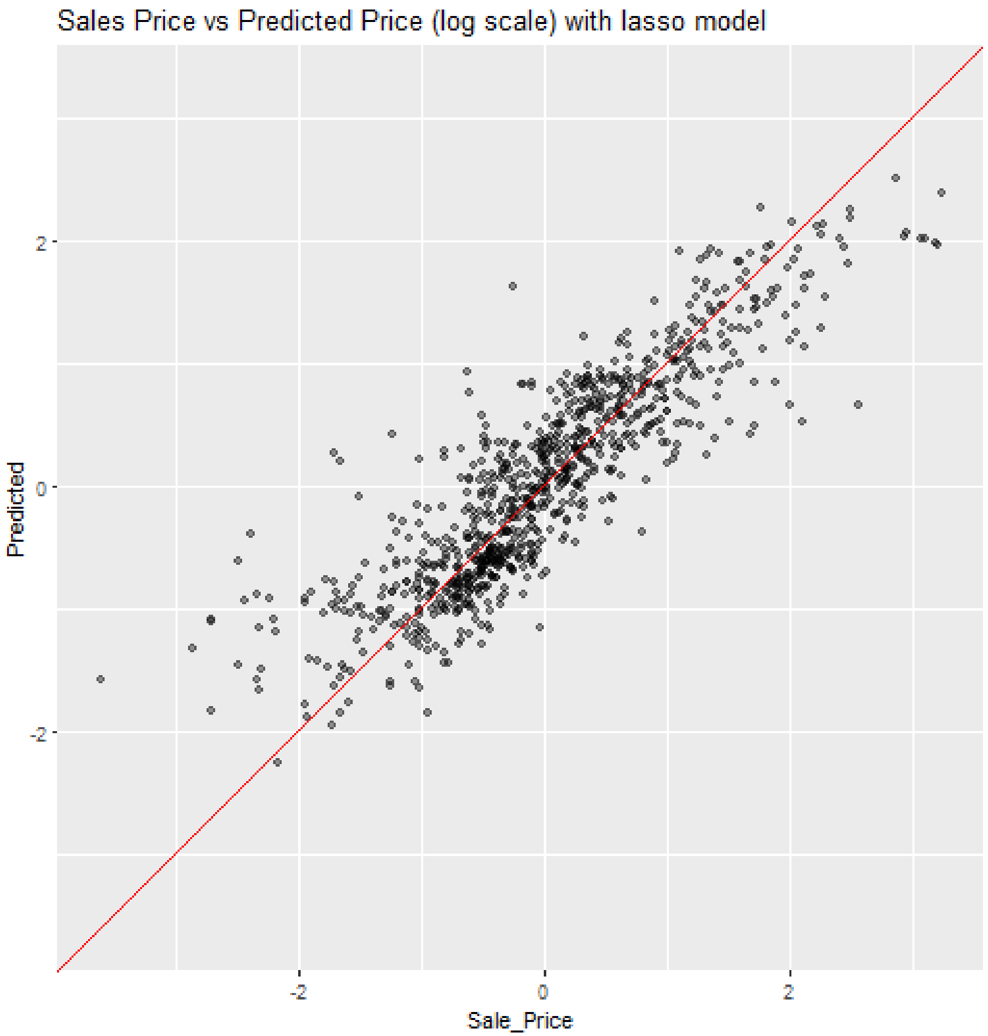
Important variables in the model



```r
lasso_final <- last_fit(final_lasso, ames_split)

lf <- lasso_final %>%
  collect_metrics()
lf
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.476
## 2 rsq     standard       0.771

#extract the test set predictions themselves
lasso_test_predictions <- lasso_final %>% collect_predictions()
lasso_test_predictions
## # A tibble: 879 x 4
##    id                 .pred  .row Sale_Price
##    <chr>              <dbl> <int>      <dbl>
##  1 train/test split -0.787      2     -1.13
##  2 train/test split  0.0678     8      0.345
##  3 train/test split  0.492      9      0.862
##  4 train/test split  0.388     10      0.313
##  5 train/test split  0.0822    13      0.199
##  6 train/test split  2.50      16      2.87
##  7 train/test split  1.12      18      2.11
##  8 train/test split -0.485     24     -0.269
##  9 train/test split -0.779     31     -1.11
## 10 train/test split  1.60      39      2.12
```


Sales Price vs Predicted Price (log scale) with lasso model
Predicted
2
0
-2
-2
0
2
Sale_Price

Comparison of model results

	knn	rf	lasso
rmse	0.434	0.416	0.476
rsq	0.809	0.825	0.771

Showing 1 to 2 of 2 entries

Which model would you chose??

Are we finished modelling the Ames data?

Most assuredly not!

What are some of the things we might do?

1) Add more specific variables or look at the entire set of variables

2) Spend more time doing feature engineering

3) Tune the models more carefully

4) Run other types of models - maybe even try other types of ensemble modelling

Now with the time remaining, let's move to a **classification** problem

I'll do this in real time with a R markdown file-"silge9_multinom.Rmd".

Otherwise, you can execute the code chunks on your own time.