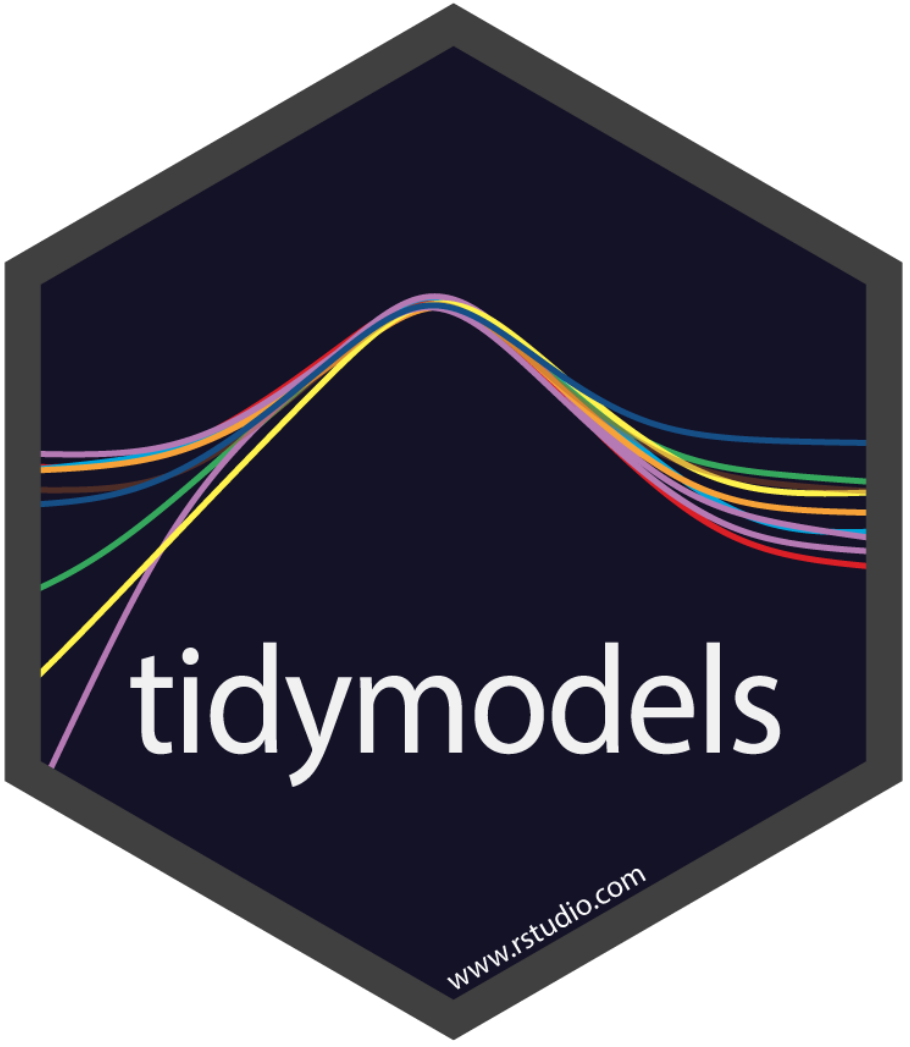
Coding Machine Learning Models with R

Meet Tidymodels

John Lewis

2020/04/08 (updated: 2020-08-11)



These slides and a pdf file can be found at (github.com/jelewis and repository:OpenGeoHubSlides1)

Why Tidymodels??

Fundamentally, tidymodels is an ecosystem of packages which is specifically designed with common APIs and a shared philosophy.

R has a consistency problem. Since everything was made by different people and using different principles, everything has a slightly different interface, and trying to keep everything in line can be frustrating. Therefore, to circumvent this problem 'tidymodels' was created by Max Kuhn at R Studio.

So the 'tidymodels' package is an integrated, modular, extendable set of packages that implement a framework that facilitates creating predicative statistical models. It adhere to tidyverse syntax and design principles that promote consistency and well-designed human interfaces over the speed of code execution.

However, it has built-in capabilities for parallel execution tasks to help with resampling, cross validation and parameter tuning. 'Tidymodels' works through the steps of the basic ML modelling and implements conceptual structures that make complex iterative workflows possible and reproducible.

*Paraphrased from: Joseph Rickert R Views 2020-04-21

Who am I?

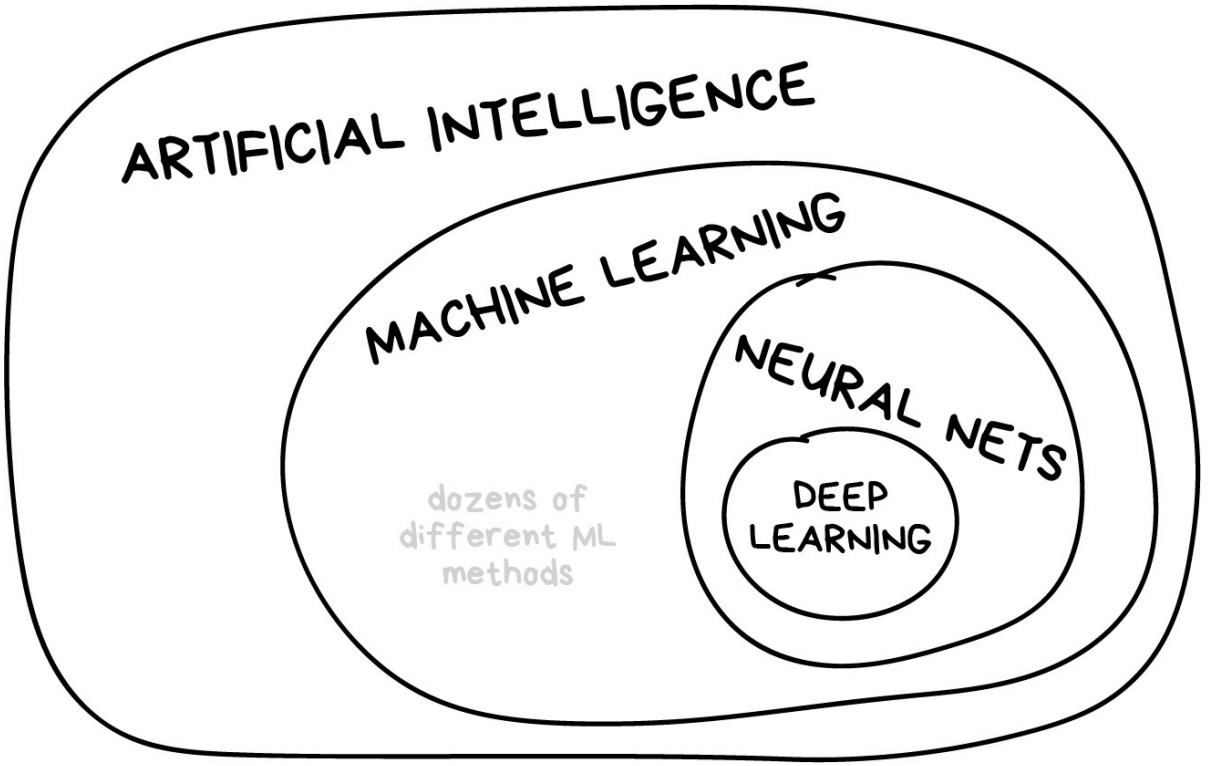
John Lewis

McGill University Professor (retired)

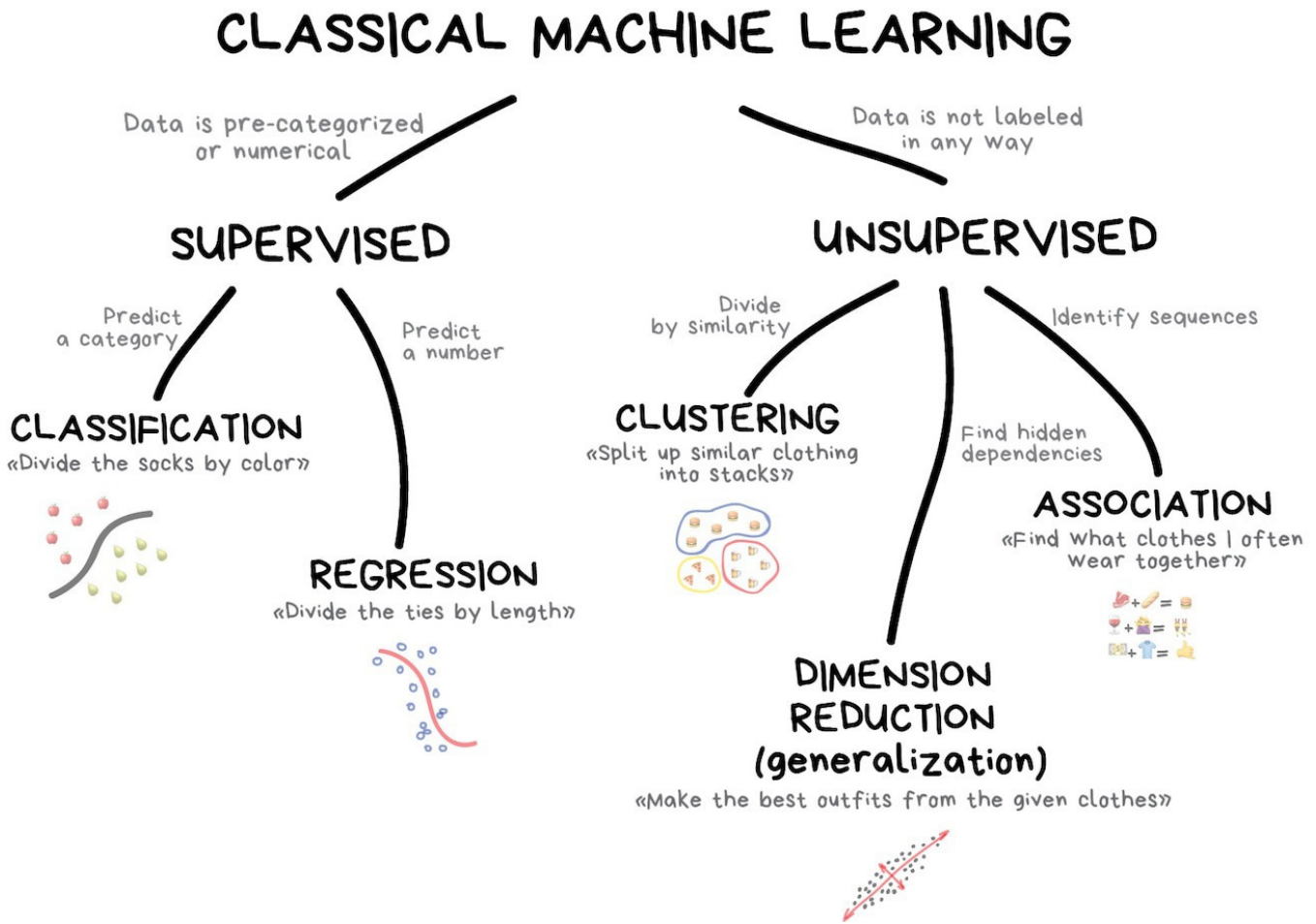
"Github" **http://github.com/jelewis**

"email" **jelewis02@gmail.com**

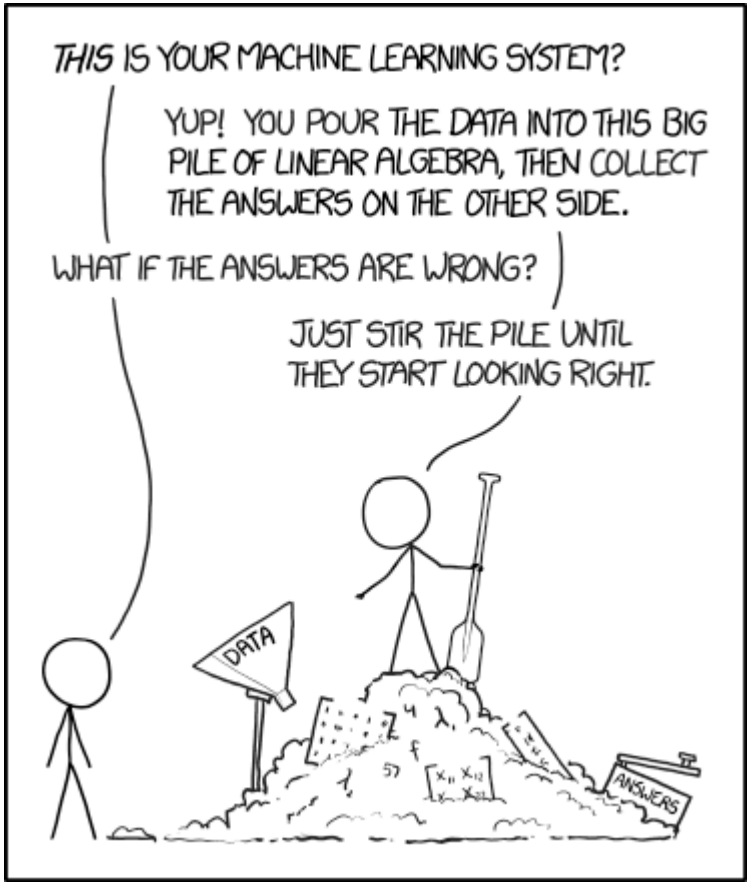
What is Machine Learning or the World of AI?



Source:https://vas3k.com/blog/machine_learning/



Source:https://vas3k.com/blog/machine_learning/



reference=@xkcd

A wise man once said "Just because you have a bag of hammers does not make every problem a nail"

Goals of this presentation

-explain key concepts of tidymodels

-use tidymodels and its companion packages to produce output for a ML regression model

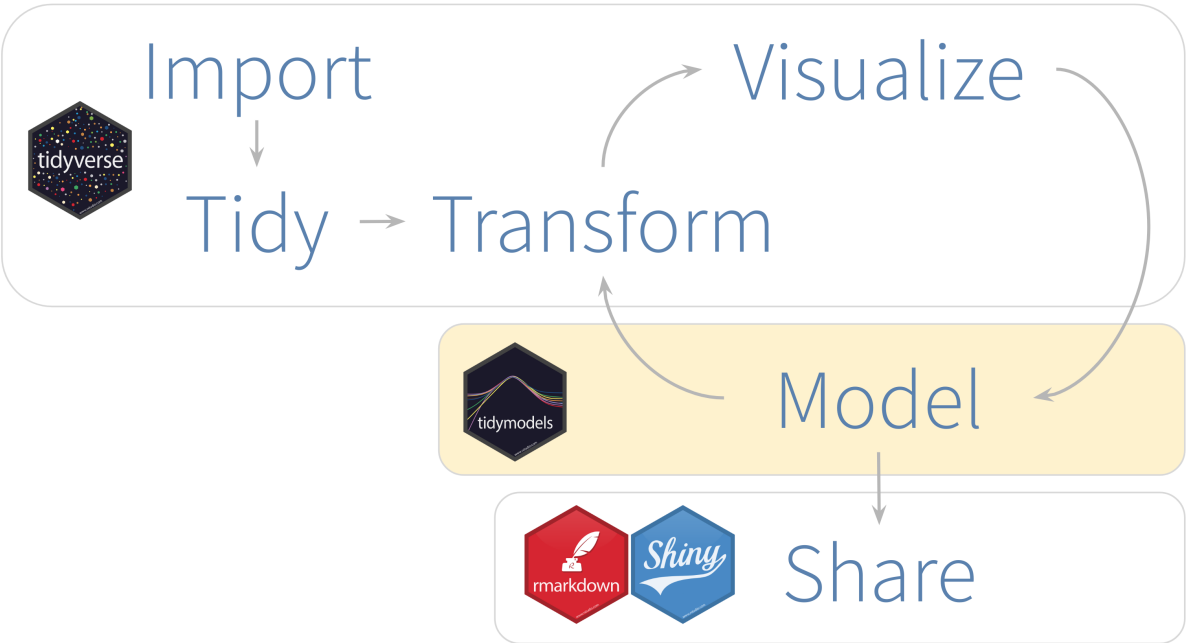
-illustrate with R code how to program the workflow sequence for ML models

Assumptions behind this talk

-Have a working knowledge of the R language

-Are somewhat familiar with machine learning material

A word about tidymodels within the tidyverse



https://rviews.rstudio.com/2019/06/19/a-gentle-intro-to-tidymodels/

An extra bit of motivation for Tidymodels

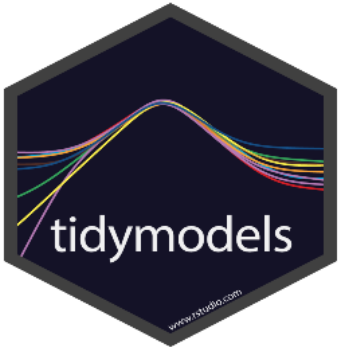
"Whether you are just starting out today or have years of experience with modeling, tidymodels offers a consistent, flexible framework for your work."

From - *Max Kuhn* originator of both 'tidymodels' and the 'carets' package in R

or

The **tidymodels** ecosystem bundles together a set of packages that work hand in hand to solve machine-learning problems from start to end. Together with the data-wrangling facilities in the **tidyverse** and the plotting tools from **ggplot2**, this makes for a rich toolbox for every data scientist working with R. (Hansjörg Plieninger, Blog, Feb. 2020)

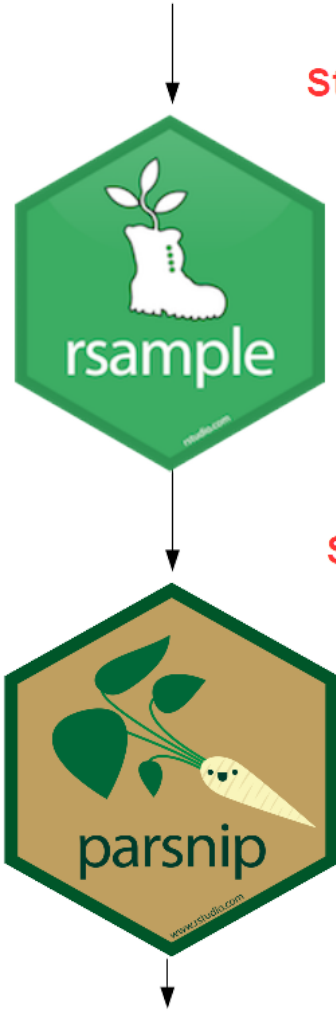
Tidymodels Program Flow



tidymodels is a meta-package that installs and load the core packages listed below that you need for modelling and machine learning

Step 1

recipes is a tidy interface to data pre-processing tools for feature engineering



rsample provides infrastructure for efficient data splitting and resampling

parsnip is a tidy, unified interface to models that can be used to try a range of models without getting bogged down in the syntactical minutiae of the underlying packages

Step 4

tune helps you optimize the hyperparameters of your model and chose pre-processing steps

Step 5

yardstick measures the effectiveness of models using performance metrics

Other important packages

workflows bundle your pre-processing, modeling, and post-processing together

dials creates and manages tuning parameters and parameter grids

broom converts the information in common statistical R objects into user-friendly, predictable formats

Other packages you might use

Learn more about the tidymodels packages and the metapackage itself please go to the following sites:

https://www.tidymodels.org/
https://tidymodels.github.io/tidymodels/

Getting set up

First we need to load some libraries: tidymodels and tidyverse.

load the relevant tidymodels libraries

```
library("tidymodels")
library("tidyverse")
```

If you don't already have the tidymodels library (or any of the other libraries) installed, then you'll need to install it (once only) using **install.packages("tidymodels")**

Loaded Packages in the 'tidyverse' and 'tidymodels' ecosystem

library(tidyverse)

```
-- Attaching packages ------------tidyverse 1.3.0 --
ggplot2 3.3.2    purrr 0.3.4
tibble 3.0.3    dplyr 1.0.0
tidyr 1.1.0    stringr 1.4.0
readr 1.3.1    forcats 0.5.0
```

library(tidymodels)

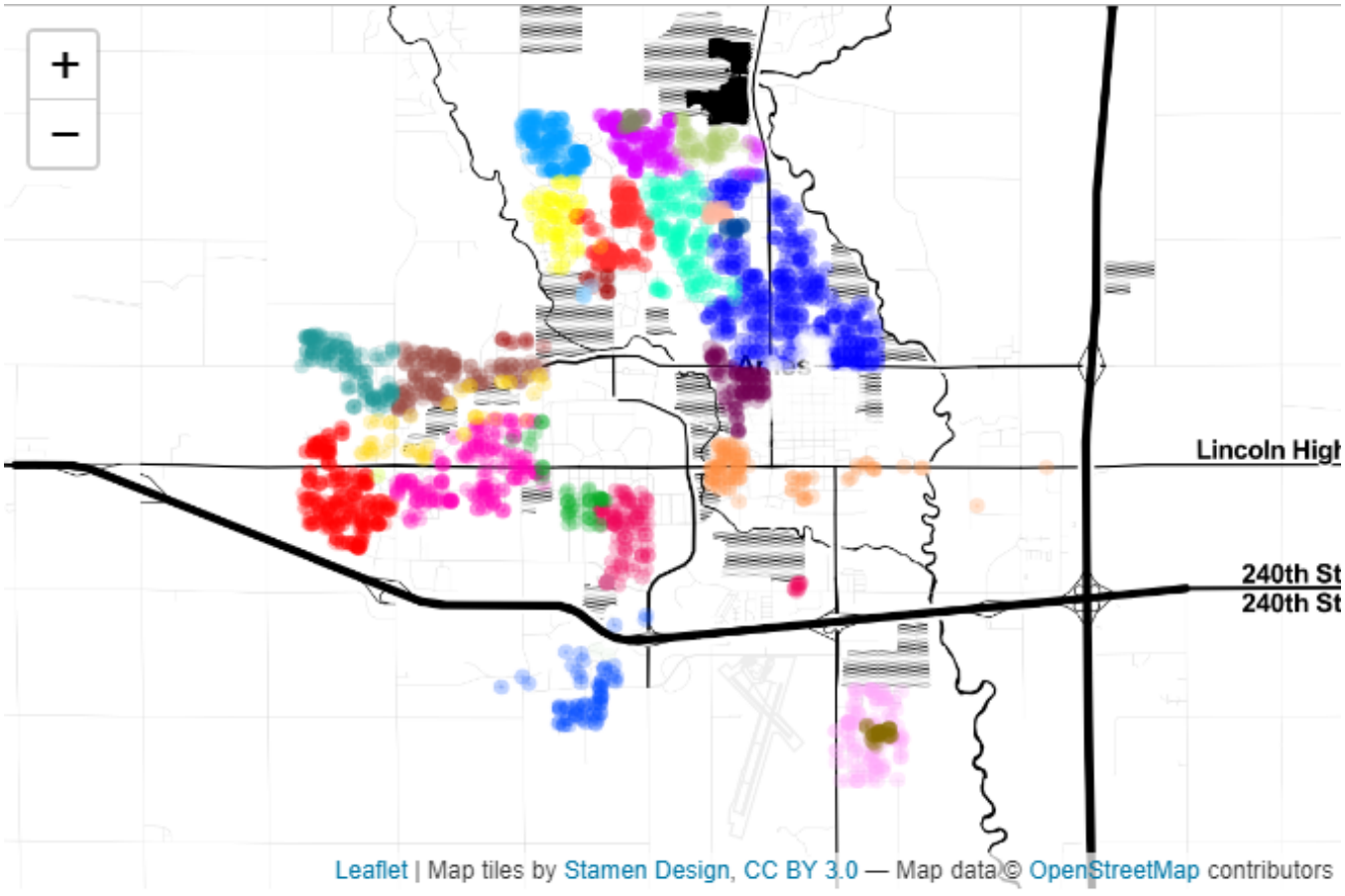
```
-- Attaching packages -------------tidymodels 0.1.1 --
broom 0.7.0    recipes 0.1.13
dials 0.0.8    rsample 0.0.7
infer 0.5.3    tune 0.1.1
modeldata 0.0.2    workflows 0.1.2
parsnip 0.1.2    yardstick 0.0.7
```

Loading the data

We'll be using the Ames Housing dataset which contains 81 variables and 2930 observations and our dependent variable/target outcome is **Sale_Price**. Obviously, in an actual analysis we would spend much more time exploring this dataset, but for sole purpose of demonstrating the {tidymodels} workflow, we'll just perform a variety of preprocessing, throw a relatively simple model at the data, use cross-validation and then tune the model.

```
data(ames, package = "modeldata")
```

Ames, Iowa



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

First look at the data

```
nrow(ames)
```

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

```
ncol(ames)
```

```
## [1] 81
```

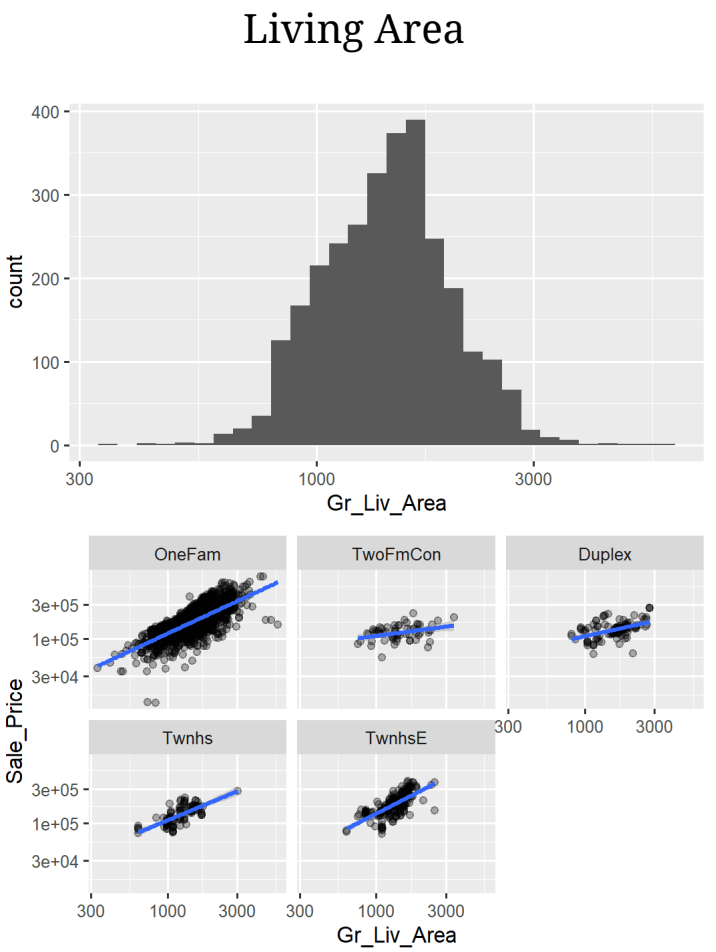
Another look: Column names

```
head(colnames(ames), n=69)
##  [1] "MS_SubClass"      "MS_Zoning"        "Lot_Frontage"
##  [4] "Lot_Area"         "Street"           "Alley"
##  [7] "Lot_Shape"        "Land_Contour"     "Utilities"
## [10] "Lot_Config"       "Land_Slope"       "Neighborhood"
## [13] "Condition_1"      "Condition_2"      "Bldg_Type"
## [16] "House_Style"      "Overall_Qual"     "Overall_Cond"
## [19] "Year_Built"       "Year_Remod_Add"   "Roof_Style"
## [22] "Roof_Matl"        "Exterior_1st"     "Exterior_2nd"
## [25] "Mas_Vnr_Type"     "Mas_Vnr_Area"     "Exter_Qual"
## [28] "Exter_Cond"       "Foundation"       "Bsmt_Qual"
## [31] "Bsmt_Cond"        "Bsmt_Exposure"    "BsmtFin_Type_1"
## [34] "BsmtFin_SF_1"     "BsmtFin_Type_2"   "BsmtFin_SF_2"
## [37] "Bsmt_Unf_SF"      "Total_Bsmt_SF"    "Heating"
## [40] "Heating_QC"       "Central_Air"      "Electrical"
## [43] "First_Flr_SF"     "Second_Flr_SF"    "Low_Qual_Fin_SF"
## [46] "Gr_Liv_Area"      "Bsmt_Full_Bath"   "Bsmt_Half_Bath"
## [49] "Full_Bath"        "Half_Bath"        "Bedroom_AbvGr"
## [52] "Kitchen_AbvGr"    "Kitchen_Qual"     "TotRms_AbvGrd"
## [55] "Functional"       "Fireplaces"       "Fireplace_Qu"
## [58] "Garage_Type"      "Garage_Finish"    "Garage_Cars"
## [61] "Garage_Area"      "Garage_Qual"      "Garage_Cond"
## [64] "Paved_Drive"      "Wood_Deck_SF"     "Open_Porch_SF"
```

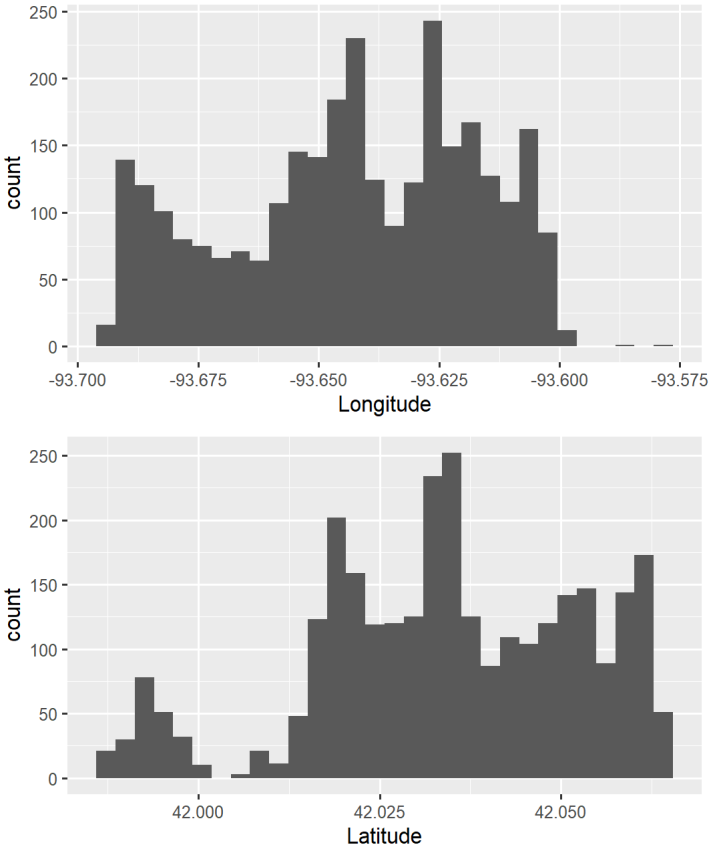
An another look: Data types & values

```
glimpse(ames)
## Rows: 2,930
## Columns: 81
## $ MS_SubClass        <fct> One_Story_1946_and_Newer_All_Styles, Or
## $ MS_Zoning          <fct> Residential_Low_Density, Residential_H
## $ Lot_Frontage       <dbl> 141, 80, 81, 93, 74, 78, 41, 43, 39, 6
## $ Lot_Area           <int> 31770, 11622, 14267, 11160, 13830, 997
## $ Street             <fct> Pave, Pave, Pave, Pave, Pave, Pave, Pav
## $ Alley              <fct> No_Alley_Access, No_Alley_Access, No_A
## $ Lot_Shape          <fct> Slightly_Irregular, Regular, Slightly_I
## $ Land_Contour       <fct> Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, HLS,
## $ Utilities          <fct> AllPub, AllPub, AllPub, AllPub, AllPub,
## $ Lot_Config         <fct> Corner, Inside, Corner, Corner, Inside,
## $ Land_Slope         <fct> Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl,
## $ Neighborhood       <fct> North_Ames, North_Ames, North_Ames, Nor
## $ Condition_1        <fct> Norm, Feedr, Norm, Norm, Norm, Norm, No
## $ Condition_2        <fct> Norm, Norm, Norm, Norm, Norm, Norm, Nor
## $ Bldg_Type          <fct> OneFam, OneFam, OneFam, OneFam, OneFam,
## $ House_Style        <fct> One_Story, One_Story, One_Story, One_St
## $ Overall_Qual       <fct> Above_Average, Average, Above_Average,
## $ Overall_Cond       <fct> Average, Above_Average, Above_Average,
## $ Year_Built         <int> 1960, 1961, 1958, 1968, 1997, 1998, 200
## $ Year_Remod_Add     <int> 1960, 1961, 1958, 1968, 1998, 1998, 200
```

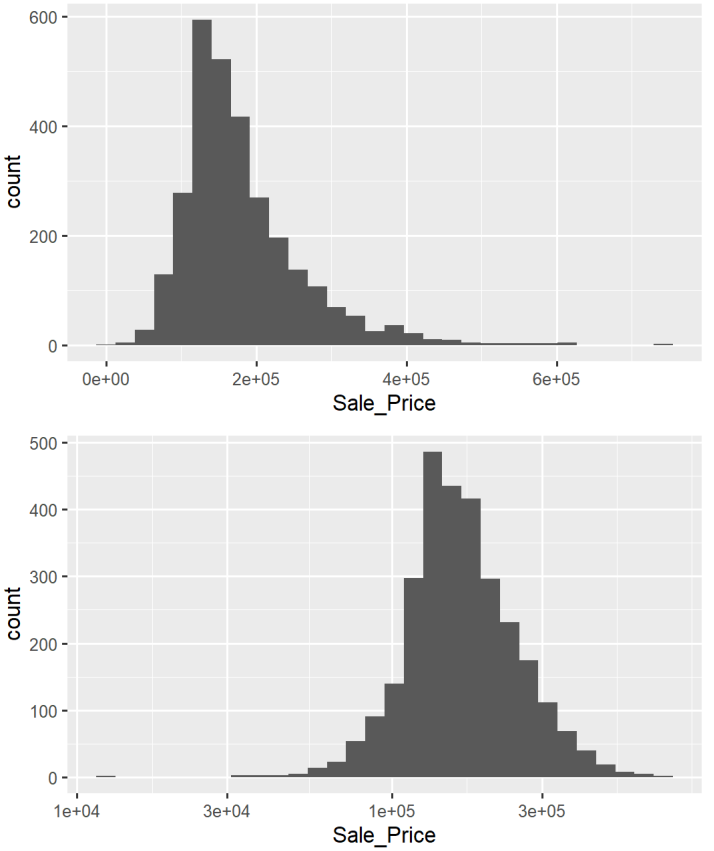
Take a look at some data plots - an example of potential one predictor variable



We could use all the variables but for this example I am only using two



The response/target variable (Sales Price) we want to predict



(log10 transform)

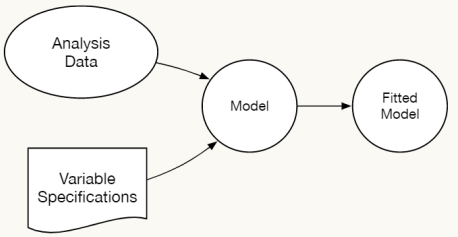
Focus variables

```
## # A tibble: 5 x 3
##   Latitude Longitude Sale_Price
##      <dbl>     <dbl>      <int>
## 1     42.1     -93.6     215000
## 2     42.1     -93.6     105000
## 3     42.1     -93.6     172000
## 4     42.1     -93.6     244000
## 5     42.1     -93.6     189900
```

What Are We Doing with the Data?

We often think of the model as the *only* real data analysis step in this process.

However, there are other procedures that are often applied before or after the model fit that are data-driven and have an impact.



If we only think of the model as being important, we might end up accidentally overfitting to the data in-hand. This is very similar to the problems of "the garden of forking paths" and "p-hacking".

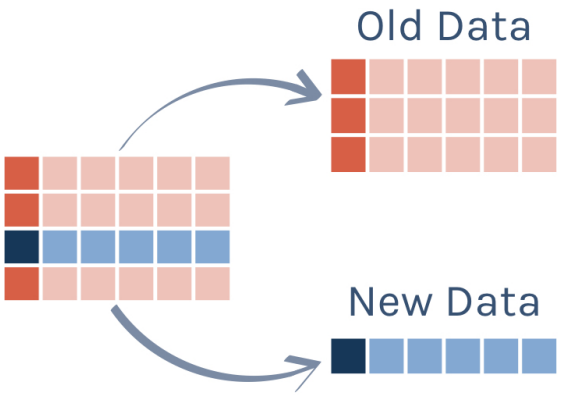
Source: Max Kuhn & Davis Vaughan https://github.com/rstudio-conf-2020/applied-ml/blob/master/Part_1.pdf

Now we start the modelling workflow-

First we want to split the data into a training set and a test set

For that we use the Tidymodels package "rsample"

Step 1: Split the data



Source:https://alison.rbind.io/post/2019-12-23-learning-to-teach-machines-to-learn/

Data Partitioning

Use the package "rsample"


```
set.seed(7014) #for reproducability
ames_split <- initial_split(ames, prop = .70)
#prop defines the amount of split
ames_split
```

```
## <Training/Validation/Total>
## <2051/879/2930>
```

```
ames_train <- training(ames_split)
#
ames_cv <- vfold_cv(ames_train)# for later to do cross-validation
```

Next step - Define a recipe

Recipes allow you to specify the role of each variable as an outcome or a predictor variable (using a “formula”), and any preprocessing steps you want to conduct (such as transforms, normalization, imputation, PCA, etc)

Creating a recipe has two parts:

1. Specify the formula (recipe()): specify the outcome variable and predictor variables.

2. Specify preprocessing steps (step_*()): define the preprocessing steps,such as imputation, creating dummy variables, scaling, and more

Preprocessing

Use the package "recipes"


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

We could have use many more variables with much more prepossessing as this recipe shows:

```
ames_rec <- recipe(Sale_Price ~ ., data = ames_train) %>%

step_log(Sale_Price, base = 10) %>%
step_YeoJohnson(Lot_Area, Gr_Liv_Area) %>%
step_other(Neighborhood, threshold = .1) %>%
step_dummy(all_nominal()) %>%
step_zv(all_predictors()) %>%
step_ns(Longitude, deg_free = tune("lon")) %>%
step_ns(Latitude, deg_free = tune("lat"))
```

The full list of preprocessing steps available can be found here:

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

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.**
- **Scaling of the predictors using a transformation can lead to a big improvements.**
- **Many models cannot cope with missing data so imputation strategies might be necessary.**
- **In some cases, the data can be encoded in a way that maximizes its effect on the model such as the date for the day of the week can be very effective for modeling air pollution data.**

Source:Max Kuhn Workshop Presentation at the 'nyr-2020' Meeting, August 2020

Model Training & Tuning

Parsnip offers a unified interface for the "massive" variety of models that exist in R.

This means that you only have to learn one way of specifying a model, and you can use this specification and have it generate a linear model, a random forest model, a support vector machine model, and more with only a few lines of code.

Specifying the model:

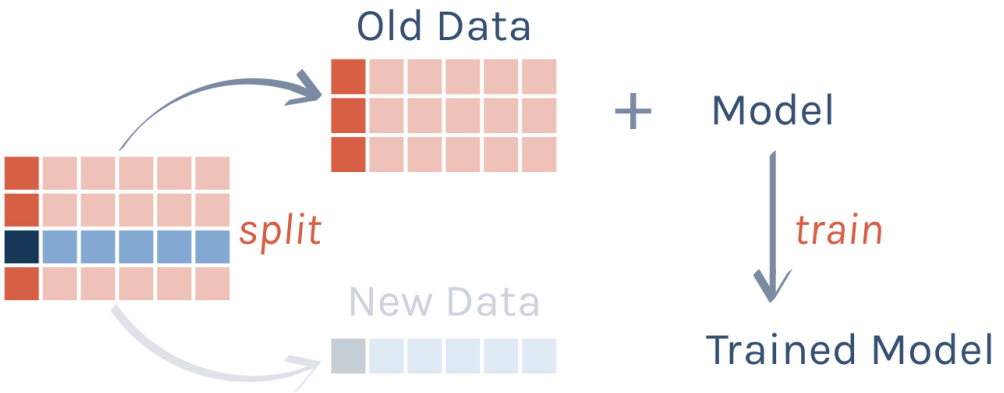
1) *The model type*: what kind of model you want to fit, set with different function of the specific model, such as rand_forest() for random forest, logistic_reg() for logistic regression etc.

2) *The engine*: the underlying package the model should come from (e.g.*ranger* for the ranger implementation of Random Forest), set using *set_engine()*.

3) *The mode*: the type of prediction - since several packages can do both classification (binary/categorical prediction) and regression (continuous prediction), set using *set_mode()*.

4) *The arguments*: the model parameter values (now consistently named across different models), set using *set_args()*.

Step 2: Train the model



Source:https://alison.rbind.io/post/2019-12-23-learning-to-teach-machines-to-learn/

A list of models accessible via **parsnip**

The mode of a model is related to its goal-*regression* and *classification*

common models: boost_tree(), decision_tree(), mars(), mlp(), nearest_neighbor(), rand_forest(), svm_poly(), svm_rbf (), ...

classification: logistic_reg(),multinom_reg()

regression: linear_reg(), surv_reg()

Source: vignettes/articles/Models.Rmd

For more of a complete list please see:

https://www.tidymodels.org/find/parsnip/

Use "parsnip"

We can easily fit a 5-nearest neighbor model with the following code:

```
fit_knn <-
  nearest_neighbor(mode = "regression", neighbors = 5) %>%
  set_engine("kknn") %>%
  fit(log10(Sale_Price) ~ Longitude + Latitude, data = ames_train)
fit_knn
```

```
## parsnip model object
##
## Fit time:  40ms
##
## Call:
## kknn::train.kknn(formula = log10(Sale_Price) ~ Longitude + Latitude,     d
##
## Type of response variable: continuous
## minimal mean absolute error: 0.07027179
## Minimal mean squared error: 0.009850297
## Best kernel: optimal
## Best k: 5
```

A few words about kNN - nearest neighbor model

k-nearest neighbor is a relatively simple supervised algorithm where each observation is predicted based on its "similarity" to its nearest neighbors. Its particular strengths are:

1) the algorithm is easy to understand which enables better interpretation of your results

2) it makes no assumptions about the data, such as its distributional form

3) it may not provide the best prediction accuracy but it works well for a wide variety of datasets

4) it can be very useful for preprocessing purposes

Define a knn regression model with the tuning parameters empty for later tuning.

```
knn_mod <-
  #specify the model as nearest neighbor
  nearest_neighbor() %>%
  #set tuning parameters-we chose 2 from the knn model - no. of neigh
  #and distance exponent
  set_args(neighbors = tune(),dist_power = tune()) %>%
  # set R package that is associated with the model
  set_engine("kknn")%>%
  set_mode("regression")
```

Putting it all together with workflow

Now I am going to diverge from the modelling steps I showed during the introduction.

This is to introduce a new package entitled "workflow".

`*workflow*` is a package that can bundle together your preprocessing, modeling, and post processing requests. For example, you don't have to keep track of separate objects in your workspace and model fitting can be executed using a single call to *fit()*.

Construct a workflow that combines your recipe and your model

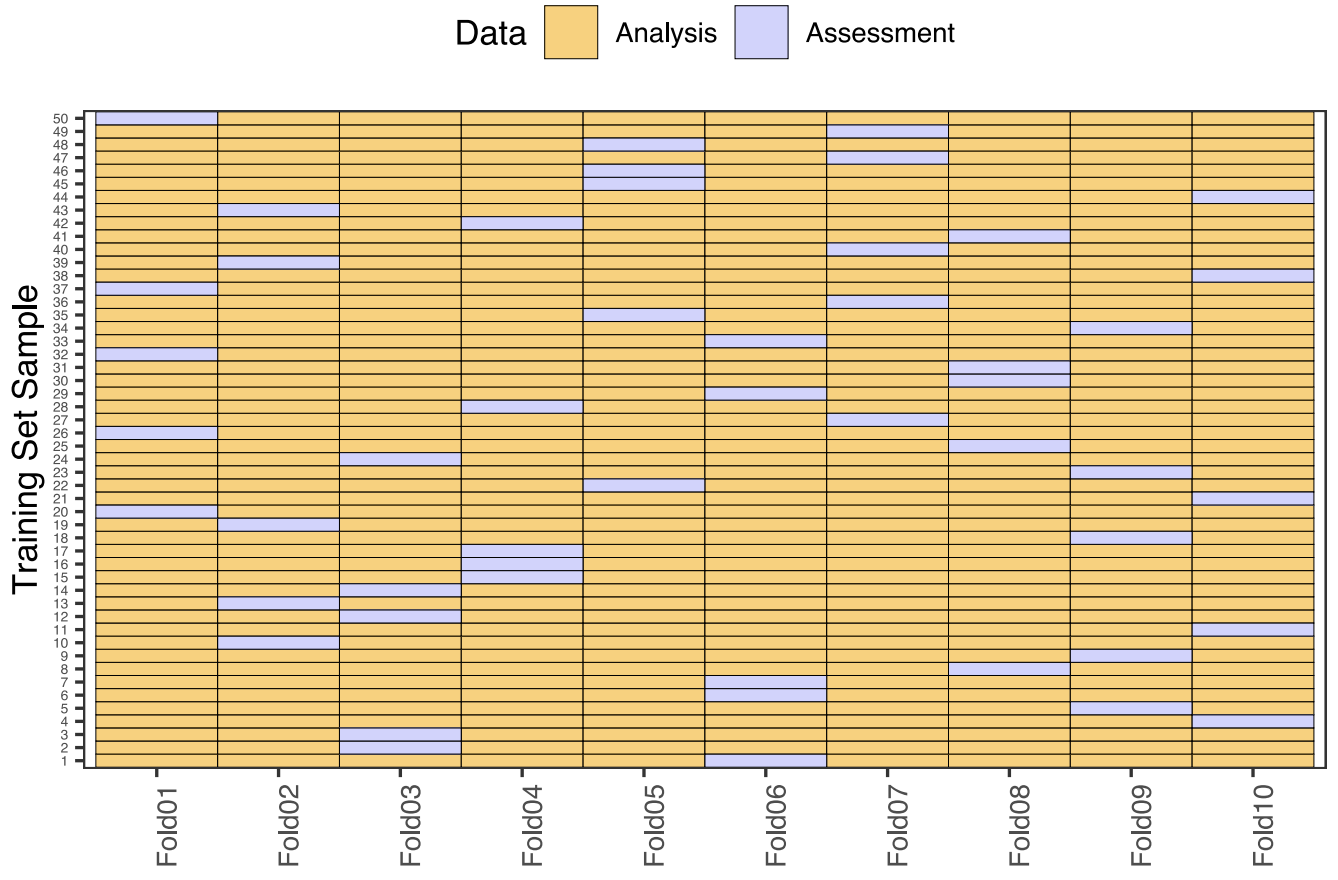
```
ml_wflow <-
  workflow() %>%
  #add the recipe
  add_recipe(mod_rec) %>%
  #add the model
  add_model(knn_mod)
```

Tune the hyperparameters

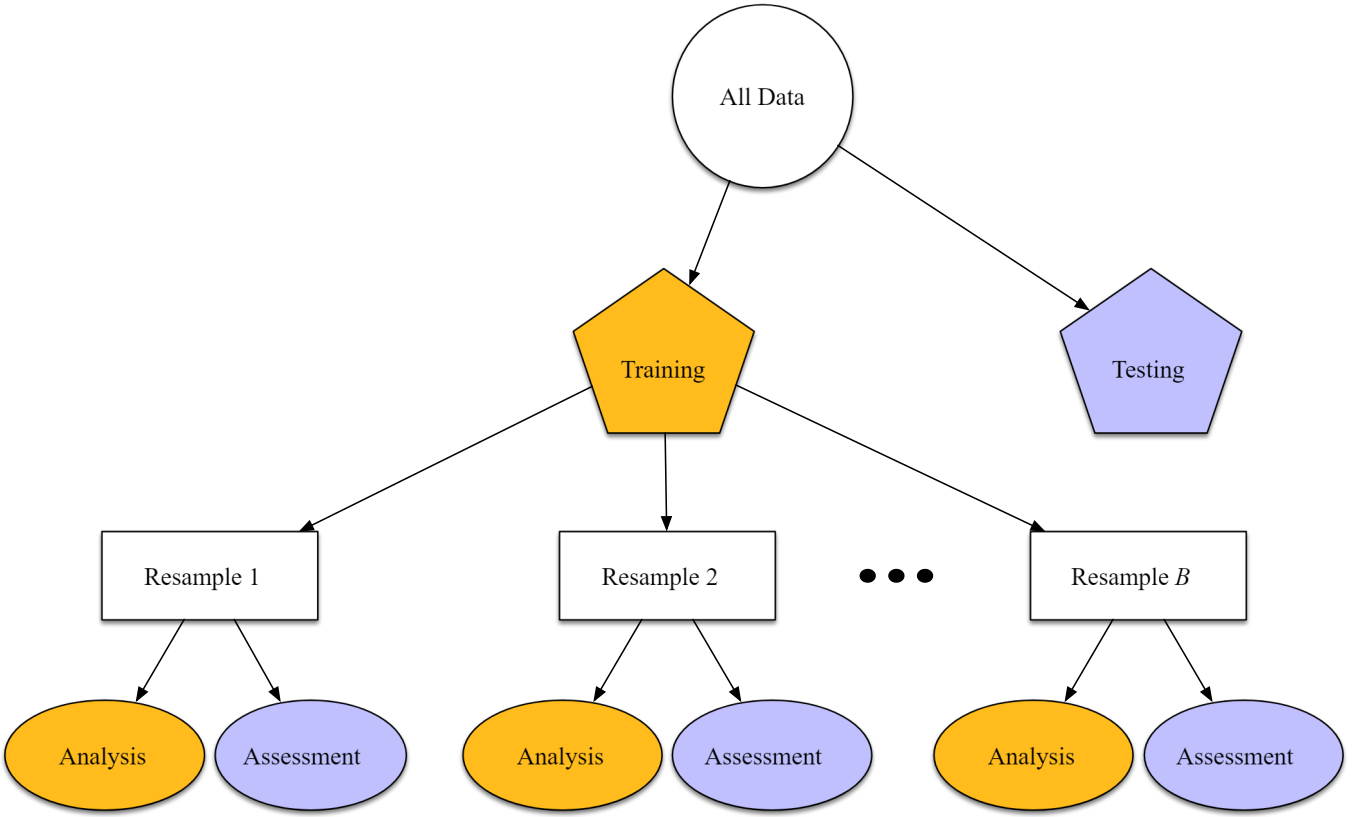
We need to tune the model (i.e. choose the hyperparameter value(s) that leads to the best performance) before fitting our final model. If there are no hyperparameters to tune, you can skip this step.

In this example, we will do the tuning using the vfold cross-validation (vfold=10) object that we created previously then specify the range for k neighbors and distanced squared exponent after which we add a tuning layer to our workflow using the function *tune_grid()*

10 fold cross validation example



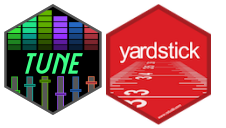
Source:Max Kuhn & Davis Vaughan https://github.com/rstudio-conf-2020/applied-ml/blob/master/Part_4.pdf



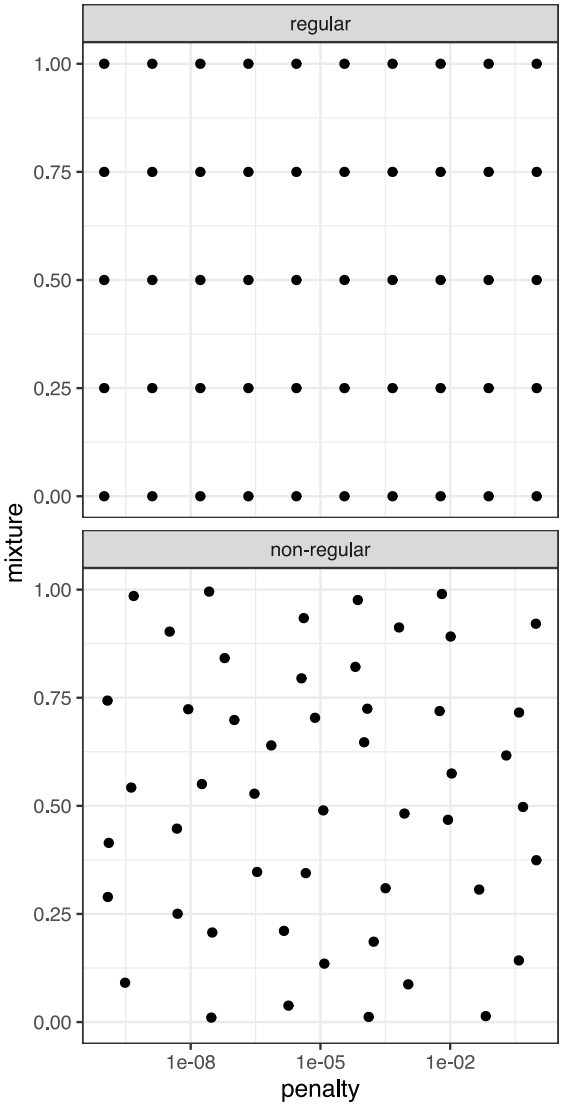
source:https://rviews.rstudio.com/2020/04/21/the-case-for-tidymodels/

Use "tune" and "yardstick" package to find best model

Objective: find the best model




```
ml_wflow_tune <-
  ml_wflow %>%
  tune_grid(resamples = ames_cv, # cv object
                  grid = 10, # grid values
                  metrics = metric_set(rmse)) # performance metric of
```


regular
non-regular
mixture
1.00
0.75
0.50
0.25
0.00
1e-08
1e-05
1e-02
penalty

Pre-Process → Train → Validate

https://rviews.rstudio.com/2019/06/19/a-gentle-intro-to-tidymodels/

A word about yardstick

yardstick is a package to estimate how well models are working

There are a large number of different metrics that one can use for assessment. The three main groups are:

1) *class metrics*, such as accuracy, sensitivity, kappa matrix ...,

2) *probability metrics* such as, area under the receiver operator curve(ROC)...,

3) *regression metrics* that has root mean squared error, R^2, mean absolute error and many more.

For a complete list of metric choices please see:

https://tidymodels.github.io/yardstick/reference/index.html

collecting the results of the tuning

```
res  <- ml_wflow_tune %>%
  collect_metrics()
```

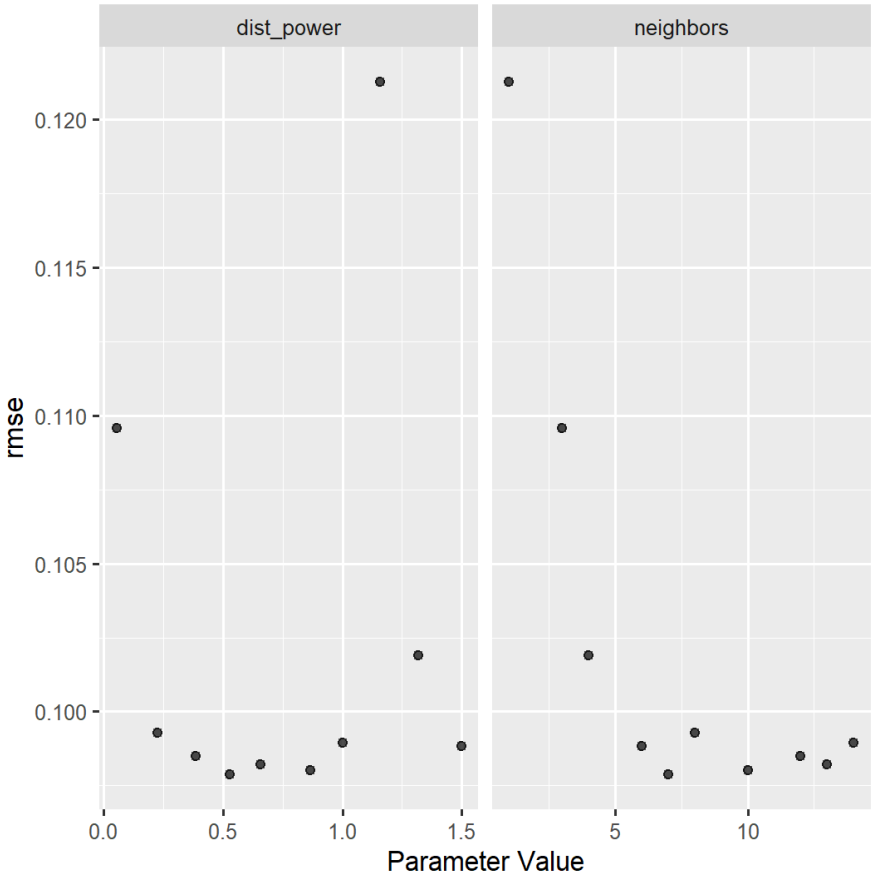
printing the results

```
res
```

```
## # A tibble: 10 x 7
##    neighbors dist_power .metric .estimator   mean     n std_err
##        <int>      <dbl> <chr>   <chr>       <dbl> <int>   <dbl>
##  1         1     1.16   rmse    standard   0.121     10 0.00519
##  2         3     0.0548 rmse    standard   0.110     10 0.00507
##  3         4     1.32   rmse    standard   0.102     10 0.00452
##  4         6     1.50   rmse    standard   0.0988    10 0.00451
##  5         7     0.530  rmse    standard   0.0979    10 0.00484
##  6         8     0.227  rmse    standard   0.0993    10 0.00486
##  7        10     0.866  rmse    standard   0.0980    10 0.00493
##  8        12     0.387  rmse    standard   0.0985    10 0.00491
##  9        13     0.659  rmse    standard   0.0982    10 0.00493
## 10        14     1.00   rmse    standard   0.0989    10 0.00496
```

Plot performance over iterations for cross-validation

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



Finalize the workflow

Now need to add a layer to our workflow that adds the tuned parameter, i.e. k neighbors for the value that yielded the best results.

We extract the best value for the accuracy metric by applying the *select_best()* function to the tune object.

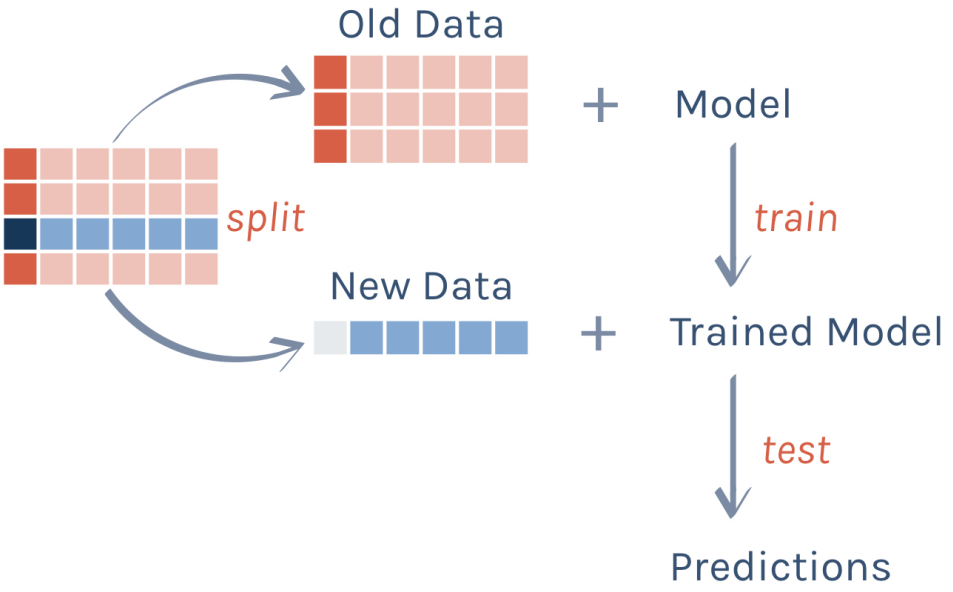
Select best parameters

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

Finalize workflow

```
# Now add this parameter (best_params) to the workflow using the
# finalize_workflow() function.
ames_reg_res <-
  ml_wflow %>%
  finalize_workflow(best_params)
```

Step 3: Make predictions



Source:https://alison.rbind.io/post/2019-12-23-learning-to-teach-machines-to-learn/

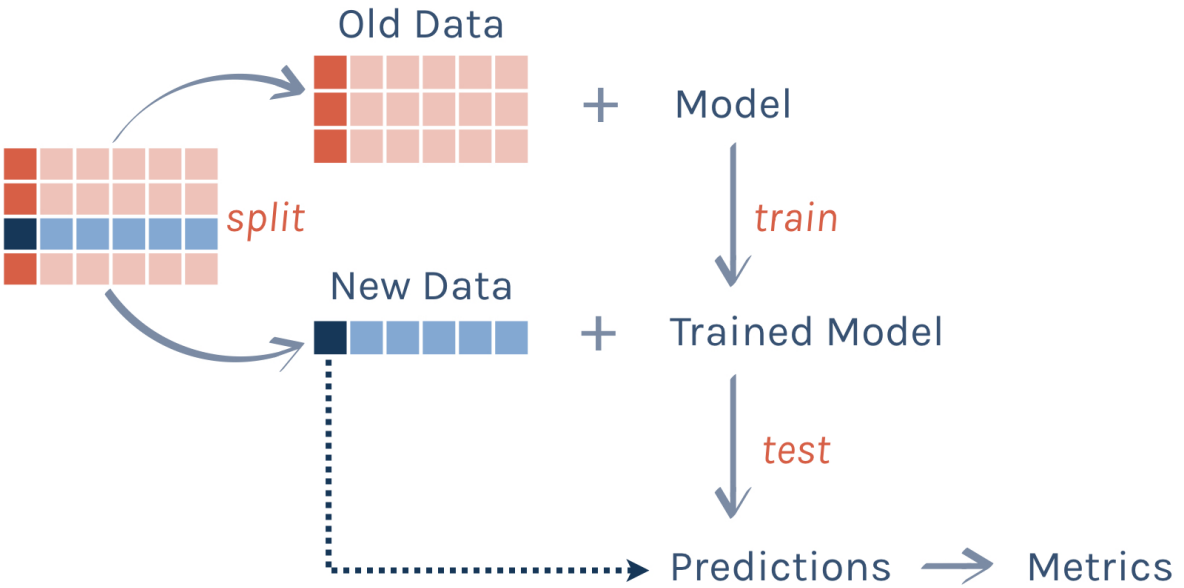
Fit the final model

Now that we've defined our recipe, our model, and tuned the model's parameters, we're ready to actually fit the final model. Since all of this information is contained within the workflow object, we just apply the *last_fit()* function to our workflow and the train/test split object. This will automatically train the model specified by the workflow using the training data, and produce evaluations based on the *test* set.

Fit using the entire training data

```
ames_wfl_fit <- ames_reg_res %>%
  last_fit(ames_split)
```

Step 4: Compute metrics



Source:https://alison.rbind.io/post/2019-12-23-learning-to-teach-machines-to-learn/

Test Performance

Since we supplied the train/test object when we fit the workflow object, the metrics are evaluated on the test set. Now when we use the *collect_metrics()* function, it extracts the performance of the final model (since *ames_wfl_fit* now consists of a single final model) applied to the test set.

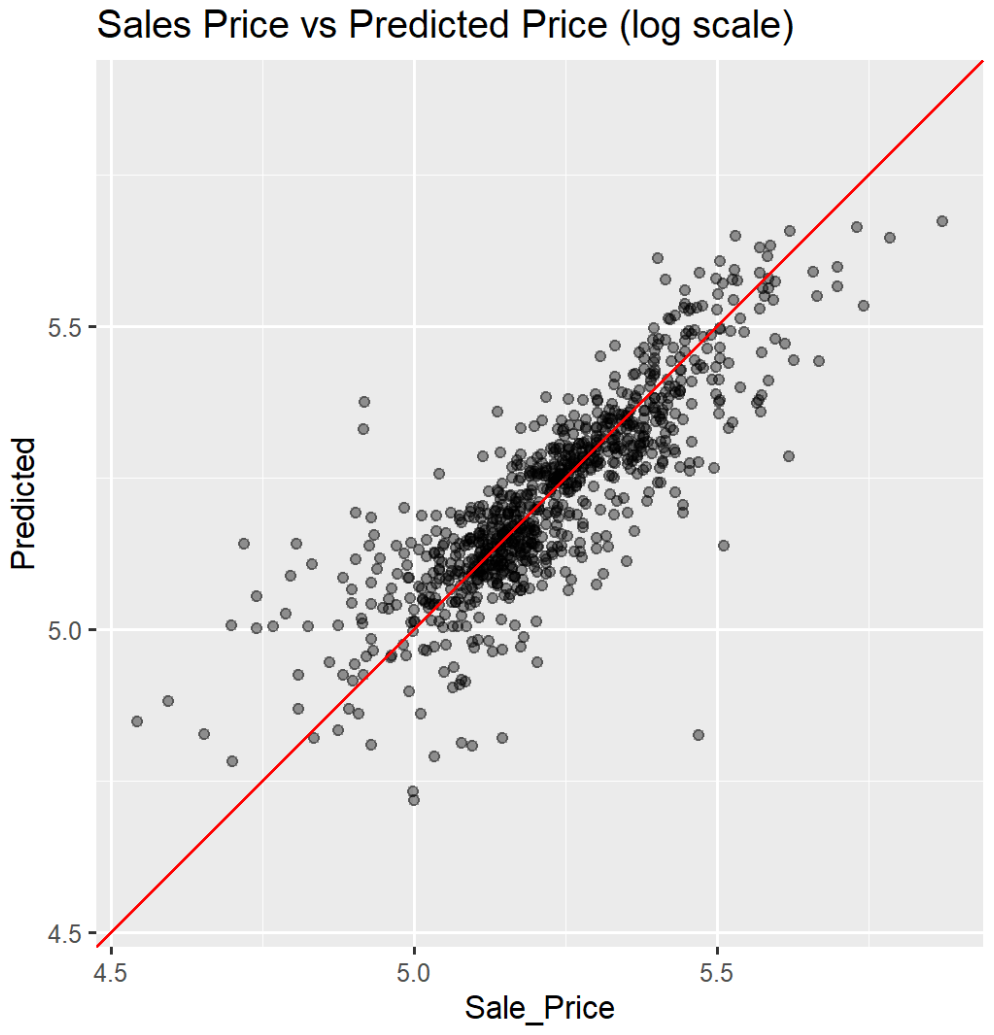
```
test_performance <- ames_wfl_fit %>%
  collect_metrics()
test_performance #print the results
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0959
## 2 rsq     standard      0.703
```

extract the test set predictions themselves

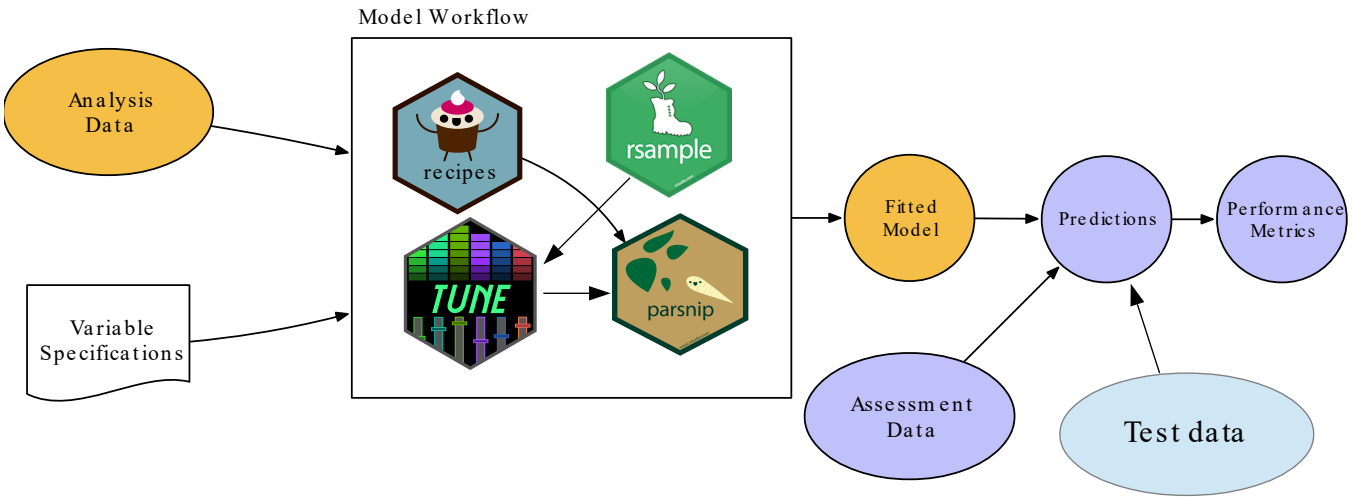
```
test_predictions <- ames_wfl_fit %>%
  collect_predictions()
test_predictions #print the results
```

```
## # A tibble: 879 x 4
##    id                .pred  .row Sale_Price
##    <chr>             <dbl> <int>      <dbl>
##  1 train/test split  5.17      3       5.24
##  2 train/test split  5.27      5       5.28
##  3 train/test split  5.27      6       5.29
##  4 train/test split  5.31      9       5.37
##  5 train/test split  5.26     11       5.25
##  6 train/test split  5.24     12       5.27
##  7 train/test split  5.22     14       5.23
##  8 train/test split  5.66     16       5.73
##  9 train/test split  5.15     24       5.17
## 10 train/test split  5.17     25       5.18
## # ... with 869 more rows
```


Sales Price vs Predicted Price (log scale)
Predicted
5.5
5.0
4.5
4.5
5.0
5.5
Sale_Price

The results of the model performance are not great but remember in this example we only used 2 predictors (longitude and latitude) which accounted for ~70% of the variance. In some ways the real job of modelling house sale prices starts now!

Finally to sum up a tidymodels workflow



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

A test-Match tasks to packages

- Fit a K-NN model
- Extract holidays from dates
- Make a training/test split
- Bundle a recipe and model
- Is high in vitamin A
- Compute R2
- Bin a predictor (but seriously,...don't)
- workflows
- yardstick
- carrot
- recipes
- parsnip
- rsample
- ggvis

Source:Max Kuhn Workshop Presentation at the 'nyr-2020' Meeting, August 2020

For more information and code you can find examples of these topic on my Github site

- build another kNN model but with different predictors in addition build a random forest model and lasso model
- compare the different model results for the Ames data
- look more deeply into cross-validation techniques and other methods of tuning hyperparameters
- also we investigate the implications of the bias-variance tradeoff and the problem of overfitting
- discuss aspects of model performance assessment in relation to a TidyTuesday dataset-global volcano data. You can run this with the supplied Rmd file.

Some references you might find useful

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

- Burkov, A., 2019: The Hundred-Page Machine Learning Book, self-published,(email: author@thermlbook.com)

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

Further Resources

*** For further information about each of the tidymodels packages, I recommend the vignettes/articles on the respective package homepage: (e.g., https://tidymodels.github.io/recipes/ or https://tune.tidymodels.org/articles/getting_started.html).**

*** Variable importance (plots) are provided by the package 'vip', which works well in combination with tidymodels packages.**

*** Recipe steps for dealing with unbalanced data are provided by the 'themis' package.There are a few more tidymodels packages that are not cover herein, like 'infer' or 'embed'. Read more about these and other packages at https://tidymodels.tidymodels.org/.**

*** Also see my slide deck "Deeper Dive into Tidymodels for Machine Learning in R" and the Rmd file (pdf too) "Multinomial classification with tidymodels and TidyTuesday volcano data" in my github account. (github.com/jelewis and repository:OpenGeoHubSlides2)**

In space, no one can hear you scream.

– Alien (1979)

Luckily tidymodels is a friendlier place. Ease of adoption and ease of use are fundamental design principles for the packages within the tidymodels ecosystem.

The end

Thanks to Max Kuhn, Alison Hill, and Julia Silge for all the insightful material concerning 'Tidymodels' that they have made available on the internet. It has made my presentation possible.