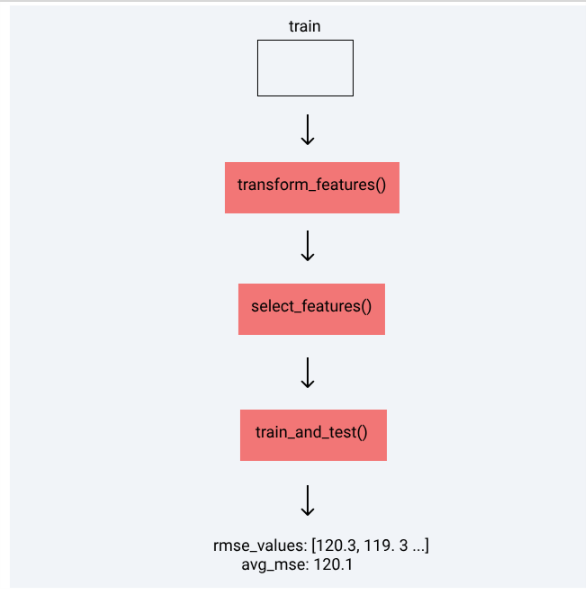
In this course, we started by building intuition for model based learning, explored how the linear regression model worked, understood how the two different approaches to model fitting worked, and some techniques for cleaning, transforming, and selecting features. In this guided project, you can practice what you learned in this course by exploring ways to improve the models we built.

You'll work with housing data for the city of Ames, Iowa, United States from 2006 to 2010. You can read more about why the data was collected [here](http://ww2.amstat.org/publications/jse/v19n3/decock.pdf). You can also read about the different columns in the data [here](https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt).

Let's start by setting up a pipeline of functions that will let us quickly iterate on different models.



You can find the solutions notebook for this guided project [here](https://github.com/dataquestio/solutions/blob/master/Mission240Solutions.ipynb).

Instructions

* Import pandas, matplotlib, and numpy into the environment. Import the classes you need from scikit-learn as well.
* Read AmesHousing.tsv into a pandas data frame.
* For the following functions, we recommend creating them in the first few cells in the notebook. This way, you can add cells to the end of the notebook to do experiments and update the functions in these cells.
  + Create a function named transform\_features() that, for now, just returns the traindata frame.
  + Create a function named select\_features() that, for now, just returns the Gr Liv Area and SalePrice columns from the train data frame.
  + Create a function named train\_and\_test() that, for now:
    - Selects the first 1460 rows from from data and assign to train.
    - Selects the remaining rows from data and assign to test.
    - Trains a model using all numerical columns except the SalePrice column (the target column) from the data frame returned from select\_features()
    - Tests the model on the test set and returns the RMSE value.

Let's now start to removing features with many missing values, diving deeper into potential categorical features, and transforming text and numerical columns. Update transform\_features() so that any column from the data frame with more than 25% (or another cutoff value) missing values is dropped. You also need to remove any columns that leak information about the sale (e.g. like the year the sale happened). In general, the goal of this function is to:

* remove features that we don't want to use in the model, just based on the number of missing values or data leakage
* transform features into the proper format (numerical to categorical, scaling numerical, filling in missing values, etc)
* create new features by combining other features

Next, you need to get more familiar with the remaining columns by reading the [data documentation](https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt) for each column, determining what transformations are necessary (if any), and more. As we mentioned earlier, succeeding in predictive modeling (and competitions like Kaggle) is highly dependent on the quality of features the model has. Libraries like scikit-learn has made it quick and easy to simply try and tweak many different models, but cleaning, selecting, and transforming features is still more of an art that requires a bit of human ingenuity.

Instructions

* As we mentioned earlier, we recommend adding some cells to explore and experiment with different features (before rewriting these functions).
* The transform\_features() function shouldn't modify the train data frame and instead return a new one entirely. This way, we can keep using train in the experimentation cells.
* Which columns contain less than 5% missing values?
  + For numerical columns that meet this criteria, let's fill in the missing values using the most popular value for that column.
* What new features can we create, that better capture the information in some of the features?
  + An example of this would be the years\_until\_remod feature we created in the last mission.
* Which columns need to be dropped for other reasons?
  + Which columns aren't useful for machine learning?
  + Which columns leak data about the final sale?

Now that we have cleaned and transformed a lot of the features in the data set, it's time to move on to feature selection for numerical features.

Instructions

* Generate a correlation heatmap matrix of the numerical features in the training data set.
  + Which features correlate strongly with our target column, SalePrice?
  + Calculate the correlation coefficients for the columns that seem to correlate well with SalePrice. Because we have a pipeline in place, it's easy to try different features and see which features result in a better cross validation score.
* Which columns in the data frame should be converted to the categorical data type? All of the columns marked as nominal from the [documentation](https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt) are candidates for being converted to categorical. Here are some other things you should think about:
  + If a categorical column has hundreds of unique values (or categories), should you keep it? When you dummy code this column, hundreds of columns will need to be added back to the data frame.
  + Which categorical columns have a few unique values but more than 95% of the values in the column belong to a specific category? This would be similar to a low variance numerical feature (no variability in the data for the model to capture).
* Which columns are currently numerical but need to be encoded as categorical instead (because the numbers don't have any semantic meaning)?
* What are some ways we can explore which categorical columns "correlate" well with SalePrice?
  + Read this post for some [potential strategies](http://machinelearningmastery.com/feature-selection-machine-learning-python/).
* Update the logic for the select\_features() function. This function should take in the new, modified train and test data frames that were returned from transform\_features().

Now for the final part of the pipeline, training and testing. When iterating on different features, using simple validation is a good idea. Let's add a parameter named k that controls the type of cross validation that occurs.

Instructions

* The optional k parameter should accept integer values, with a default value of 0.
* When k equals 0, perform holdout validation (what we already implemented):
  + Select the first 1460 rows and assign to train.
  + Select the remaining rows and assign to test.
  + Train on train and test on test.
  + Compute the RMSE and return.
* When k equals 1, perform simple cross validation:
  + Shuffle the ordering of the rows in the data frame.
  + Select the first 1460 rows and assign to fold\_one.
  + Select the remaining rows and assign to fold\_two.
  + Train on fold\_one and test on fold\_two.
  + Train on fold\_two and test on fold\_one.
  + Compute the average RMSE and return.
* When k is greater than 0, implement k-fold cross validation using k folds:
  + Perform k-fold cross validation using k folds.
  + Calculate the average RMSE value and return this value.

That's it for the guided steps. Here's some potenial next steps that you can take:

* Continue iteration on feature engineering:
  + Reserach some other approaches to feature engineering online around housing data.
  + Visit the [Kaggle kernels page](https://www.kaggle.com/c/house-prices-advanced-regression-techniques/kernels) for this dataset to see approaches others took.
* Improve your feature selection:
  + Research ways of doing feature selection better with categorical columns (something we didn't cover in this particular course).