# Analysis of College Graduate Earnings

Cristopher D. Benge (cbenge509), October 2017

## Executive Summary

This document presents a summary analysis of the observations and most relevant features pertaining to the prediction of future college graduate income in the Q3 2017 Data Science Capstone machine learning competition. All analysis was based on a modified / obfuscated derivative of the publicly available U.S. Department of Education: College Scorecard data provided solely for the competition.

Our task in this competition was to predict average graduate earnings some fix but unspecified period of years following enrollment in U.S. based institutions of higher education. My high-level plan of attack was as follows:

1. **Understand the problem**. The official problem description for the challenge outlined the objective to be a supervised, univariate regression with a target performance metric of root mean squared error:
   $$[\sum_{i=1}^{N} (z_{f_i} - z_{o_i})^2 / N]^{1/2}$$
   As such, most of the exploratory data analysis was dedicated to understanding statistical relationships between independent variables, the dependent variable, and a consideration of their philosophical meaning to the problem space.
2. **Identify starting assumptions**. I selected Python for the challenge as it has a wide variety of regression models to work with and broad community support. A quick survey of top 10 winners in recent Kaggle regression competitions reinforced my platform selection. As the feature count was particularly high (297) and most of the columns had missing values (288/297), I started with an assumption that a tree model would likely be most useful to begin analysis and baselining.
3. **Dependent variable research**. I focused initial analysis on the dependent variable ('income'), trying to understand more about its distribution and identify any idiosyncrasies that might pose a challenge to prediction.
4. **Independent variable analysis**. This phase, which was iterated over many times throughout the challenge period, was used to understand as much as possible about the most important predictors and how changes in their values affect the dependent variable. Feature importance was derived through a combination of Gini importance index analysis from simple tree-based estimators and through manual analysis of correlation to the dependent variable.
5. **Data pre-processing**. Pre-processing for both training and test values was completed to ensure the final model had an improved representation of the data as objectively measured through private and public RMSE score. This consisted principally of treatment for categorical variables, imputing and interpolating missing data, and feature engineering through (a) discretizing the dependent variable through a simple tree-based classifier, and (b) creation of synthetic features derived from the distance of quadratic roots between three independent features with the same base range.
6. **Testing assumptions and adjusting for best fit solution**. Initial benchmark/baseline was taken after only minimal data clean-up. This was a single decision-tree regressor which showed solid cross-validation and a decent starting score. Following baseline, all modifications to parameters, evaluation of new regressors, and additional pre-processing (such as feature synthesizing, scaling, and imputation) were treated as hyperparameter tuning. All experiments were cross-validated using the same random seed and shuffle split folds. Modifications were made iteratively, comparing both hold-out and cross-validated scores to prior runs, and noting impact on incremental changes to public leaderboard score as a means for gauging over-fitting. After individual models had been optimized and achieved reasonable scores, ensemble stacking and blending was employed to further improve final submissions.

# Initial Exploratory Data Analysis

Initial exploration of the scorecard data began with the application of simple and common data analysis principles outlined in Examining Your Data (Hair et. Al, 2013 – ref).

## General Overview from Initial Analysis

A quick survey of the provided training and test data revealed a starting-point of 297 features; 17,107 observations were provided in the training set, and 9,912 observations were provided for prediction.

I started my exploration by reviewing the base data types and reviewing the official data dictionary for ad-hoc, as-needed definition lookup. All but nine of the features provided were automatically interpreted as numeric:

```
In [8]: print(df_train.dtypes.value_counts())
        df_train.columns[(df_train.dtypes == object)]

        float64    287
        object       9
        int64        1
        dtype: int64

Out[8]: Index(['report_year', 'school__degrees_awarded_highest',
               'school__degrees_awarded_predominant',
               'school__institutional_characteristics_level', 'school__main_campus',
               'school__online_only', 'school__ownership', 'school__region_id',
               'school__state'],
              dtype='object')
```

| Column Name | Rows Missing | Pct. Missing |
|---|---|---|
| student__share_firstgeneration_parents_middleschool | 17023 | 0.99509 |
| admissions__act_scores_75th_percentile_writing | 16677 | 0.974864 |
| admissions__act_scores_midpoint_writing | 16677 | 0.974864 |
| admissions__act_scores_25th_percentile_writing | 16676 | 0.974806 |
| admissions__sat_scores_25th_percentile_writing | 15120 | 0.883849 |
| admissions__sat_scores_75th_percentile_writing | 15120 | 0.883849 |
| admissions__sat_scores_midpoint_writing | 15120 | 0.883849 |
| admissions__act_scores_25th_percentile_math | 13973 | 0.8168 |
| admissions__act_scores_75th_percentile_math | 13973 | 0.8168 |
| admissions__act_scores_midpoint_math | 13973 | 0.8168 |
| admissions__act_scores_25th_percentile_english | 13969 | 0.816566 |
| admissions__act_scores_75th_percentile_english | 13969 | 0.816566 |
| admissions__act_scores_midpoint_english | 13969 | 0.816566 |
| admissions__sat_scores_25th_percentile_critical_reading | 13523 | 0.790495 |
| admissions__sat_scores_75th_percentile_critical_reading | 13523 | 0.790495 |
| admissions__sat_scores_midpoint_critical_reading | 13523 | 0.790495 |
| admissions__sat_scores_25th_percentile_math | 13477 | 0.787806 |
| admissions__sat_scores_75th_percentile_math | 13477 | 0.787806 |
| admissions__sat_scores_midpoint_math | 13477 | 0.787806 |
| admissions__act_scores_25th_percentile_cumulative | 13412 | 0.784007 |
| admissions__act_scores_75th_percentile_cumulative | 13412 | 0.784007 |
| admissions__act_scores_midpoint_cumulative | 13412 | 0.784007 |
| admissions__sat_scores_average_overall | 13082 | 0.764716 |
| student__retention_rate_four_year_part_time | 12935 | 0.756123 |
| admissions__sat_scores_average_by_ope_id | 12768 | 0.746361 |
| cost__tuition_program_year | 12141 | 0.709709 |
| student__retention_rate_lt_four_year_part_time | 11653 | 0.681183 |
| completion__completion_rate_4yr_100nt | 11492 | 0.671772 |
| completion__completion_cohort_4yr_100nt | 11489 | 0.671596 |
| completion__transfer_rate_4yr_full_time | 10910 | 0.637751 |
| completion__transfer_rate_cohort_4yr_full_time | 10910 | 0.637751 |
| student__retention_rate_four_year_full_time | 10903 | 0.637341 |
| completion__completion_rate_less_than_4yr_100nt | 10468 | 0.611913 |
| completion__completion_cohort_less_than_4yr_100nt | 10467 | 0.611855 |
| admissions__admission_rate_overall | 10428 | 0.609575 |
| admissions__admission_rate_by_ope_id | 9623 | 0.562518 |
| school__online_only | 8639 | 0.504998 |
| student__retention_rate_lt_four_year_full_time | 8423 | 0.492372 |
| completion__transfer_rate_cohort_less_than_4yr_full_time | 8312 | 0.485883 |
| completion__transfer_rate_less_than_4yr_full_time | 8312 | 0.485883 |
| student__share_first_time_full_time | 6959 | 0.406793 |
| student__demographics_veteran | 6394 | 0.373765 |
| cost__tuition_out_of_state | 5990 | 0.350149 |
| school__ft_faculty_rate | 5921 | 0.346116 |
| school__faculty_salary | 5784 | 0.338107 |
| cost__tuition_in_state | 5659 | 0.3308 |
| student__share_25_older | 5126 | 0.299643 |

Aside from noting the nine character-based categorical columns for later analysis and encoding, I performed further research into two general categories of missing data: (1) data with missing values, and (2) categorical data present in the test set but not found in the training set. The latter analysis turned out to be important, as values not represented in the training data could make for more difficult estimation. For one-hot encoding, this would lead to a mismatch between fitting and predicting if not properly addressed.

The data in the provided training and test data sets are very sparse, which presented its own challenges. Only nine columns out of 297 were completely populated – eight of which were in the above mentioned, string-based categorical values (all but 'school__online_only').

The only ordinal value to contain a fully populated data-set was: 'school__degrees_awarded_predominant_recoded'. Of the remaining 288 columns with missing values, 47 were missing 25% or more data (4 of which were missing over 97%) – left image.

Note: (left image)) Columns with green flag indicators next to them denote features that are identified to be in the Top 20 features most useful, by Gini index, to predicting the label.

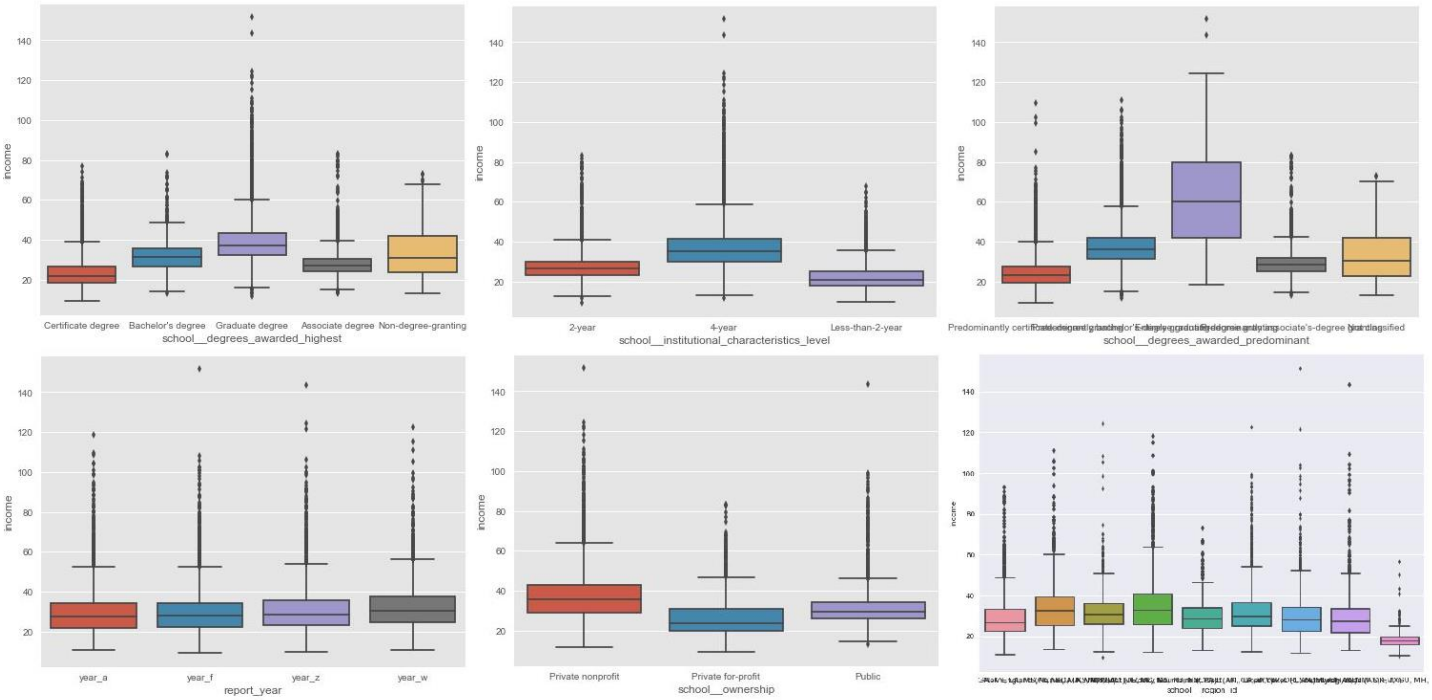
Additionally, *potentially duplicate* rows were identified:

```
In [14]: print(X[X.duplicated(keep=False)].shape)
(459, 297)
```

Unfortunately, most of the rows were predominantly sparse, making it difficult to ascertain whether they were true duplicates or unique rows with important differentiators missing. This issue appears in both the training and test sets, and all reasonable attempts to treat these rows – whether through 'keep first', 'keep last', 'keep none' or 'generate a mean average label and impute a single replacement row' – always resulted in higher error rates in cross-validation. Ultimately, this data was imputed as all other rows were and left in the model as-is.

## Categorical Data Analysis and Treatment

I approached analysis and treatment of categorical data using the following general guideline: *Wherein a linear order is apparent, and where it seems sensible given the purpose of the data*, *encode a numerical order on that data*. For all other cases, use a traditional one-hot encoding scheme. There is a risk of inadvertently adding false information to your model using this approach, but a reward of lower rate of error when correctly applied.




Boxplots were created for categorical values, and a numerical encoding order was derived for eight features (six examples depicted above). For all remaining categorical values, a one-hot encoding method was used to pivot all values to binary indicators.

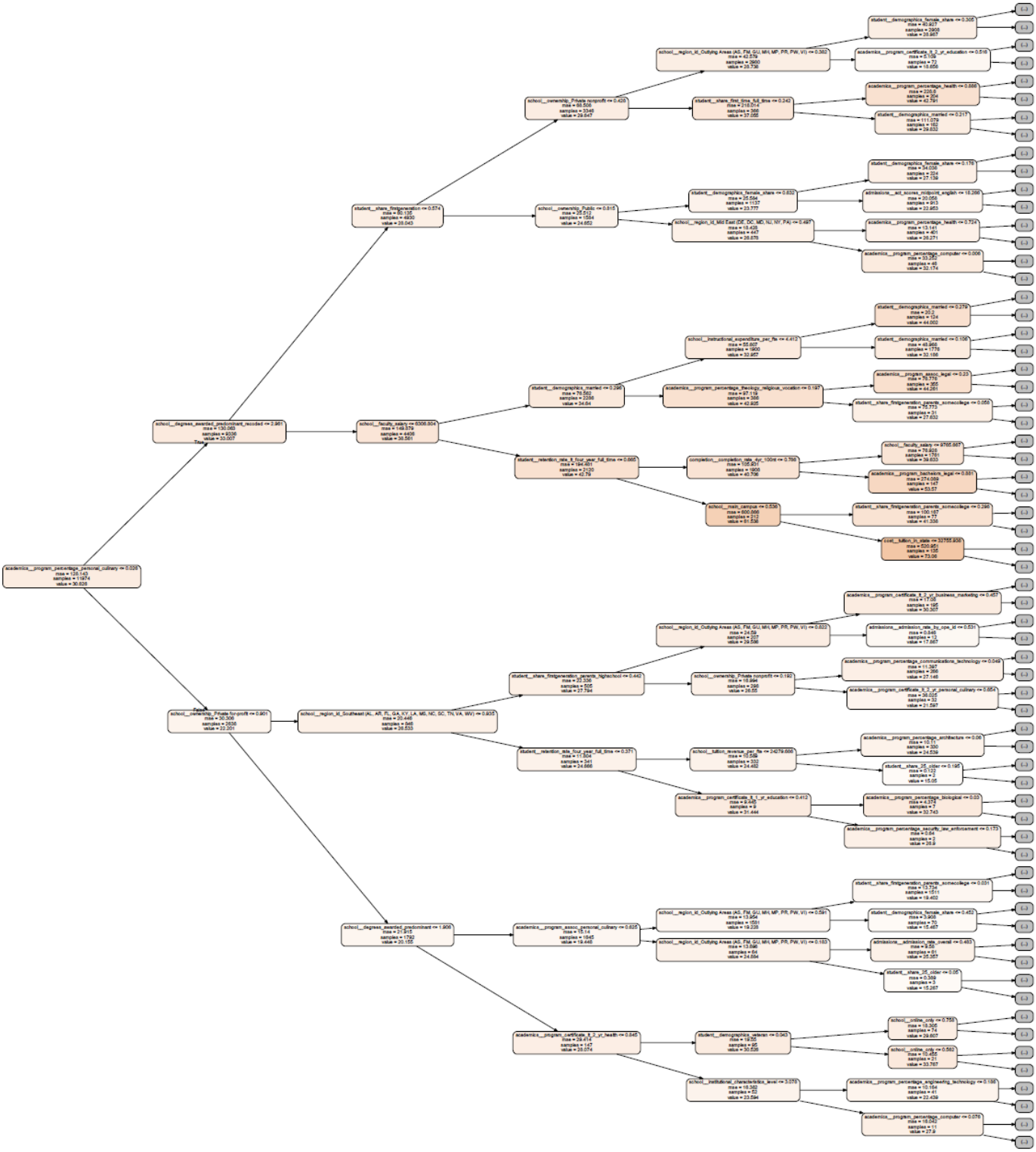
A significant number of trials were run to determine the best approach for treating missing values. The following, uniform methods were tested and rejected after consistently failing to improve local scores (both cross-validated and hold-out) and public leaderboard RMSE:

- Mean average imputing
- Median average imputing
- Most frequent value imputing
- Removal of sparse rows and/or sparse columns

In the final, best-performing model, a simple 'fill with 0' method was used for all but a few cases.

## Categorical Data Interpolation

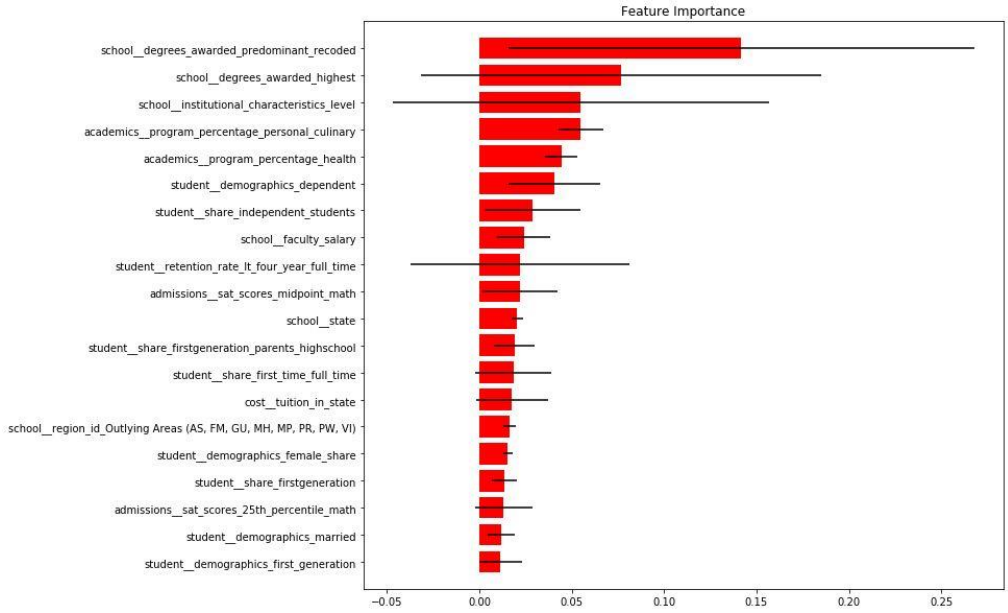
Several categorical values were interpolated using an ensemble of extra-tree regressors to predict and back-fill missing values. This turned out to be a critical move in reducing error deviation for the worst performing rows in the hold-out set (i.e. those with the highest residual $|\hat{y}_i - y_i|^2$). Analysis was performed manually in Excel after conventional hyperparameter tuning failed to produce improved CV scores. Graphical weak learner decision tree reports, like the example below, were generated on the discrepancy rows and analyzed to better ascertain influencers in the feature space, and values were treated in a way that best supported more accurate predictions on a case-by-case basis.



## Categorical Data Interpolation (Contd.)

The process of either assessing the impact of all features and/or the efficacy of tuning randomly selected features was impractical given time constraints and scope of the challenge feature space. Consequentially, selection of tuning targets was approached in a step-wise fashion through: (1) Construction of simple tree models with cross-validation, (2) capture of private RMSE and examination of 'top 20' features ranked by Gini importance, (3) deep-dive analysis and tuning experimentation, (4) and re-run of cross-validation to capture change in score and feature importance. This iterative process was time consuming, but worth several tenths off RMSE:

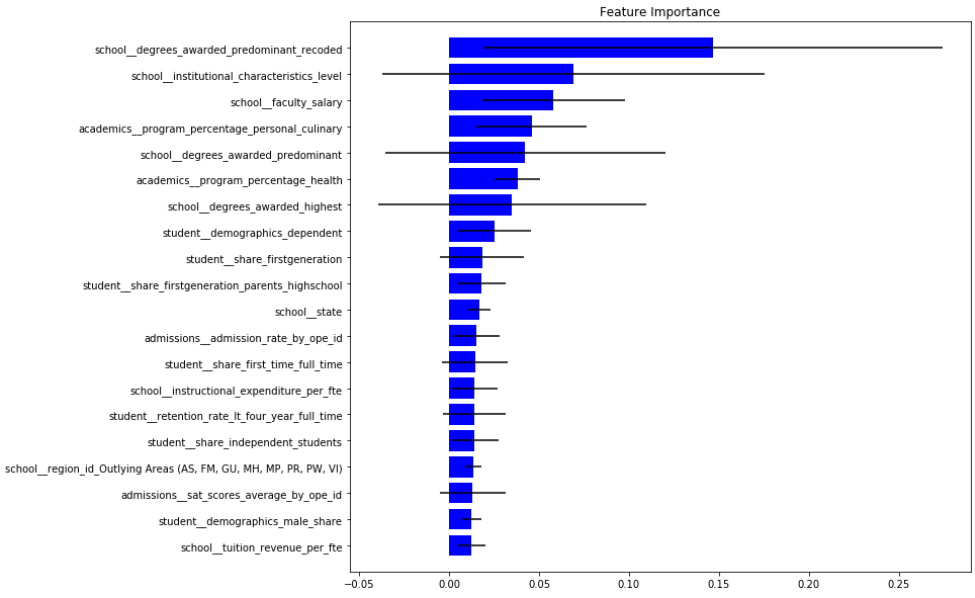
**Feature Importance (Starting Point)**




*Nine of the most important features are missing 25% or more of their values. These turned out to be prime candidates for interpolation, using [tuned] Extra Trees Regressors.*

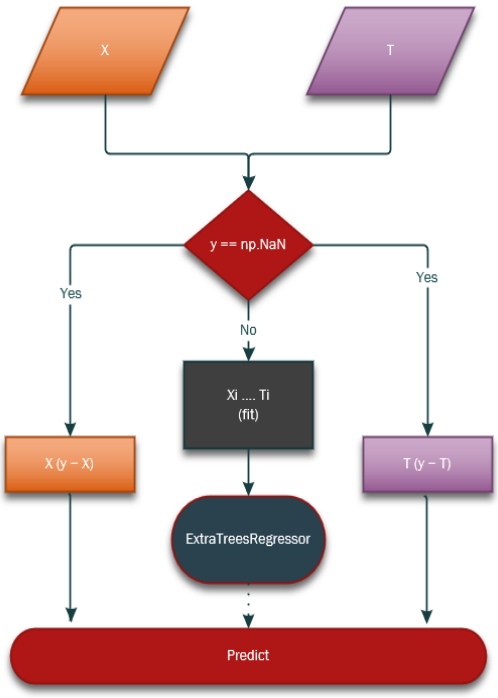
*Several features moved up in rank importance or into the top 20 list after regression interpolation.*

**Feature Importance (Final Model)**



## Categorical Data Interpolation (Contd.)

Interpolation of manually selected 'important' features was implemented through a process of combining the provided training and test data sets where a value was provided, fitting a tuned ensemble tree-based ensemble of regressors on the combined data, and predicting for the missing values on the combined model. Cross-validation and hold-out scoring was measured against each interpolated set, and only those that improved error were retained. In general, features with a higher rate of missing values (>=5,000) tended to fare better than those with only a few hundred missing.




## Synthesized and Engineered Features

Though labor-intensive, feature engineering machine learning applications may lead to improved accuracy or error-based scores when the added features provide a good representation of the feature vector. In general, tree-based machine learning models tend to be better at synthesizing common mathematical equations (Heaton, Jan. 2017 – *An Empirical Analysis of Feature Engineering for Predictive Modeling*- ref). In my testing, the addition of a single synthesized feature that reflected the distance in roots of quadratic equations between 3 similarly scaled independent variables had a nominal effect on RMSE (formula below).

During initial exploratory analysis of the data, I noted existence of a demographics_female_share feature in the student domain but absence of a correlated non-female share feature. Addition of this feature, as well as populating missing values with a 50/50 split, resulted in a very small improvement in predictive force.

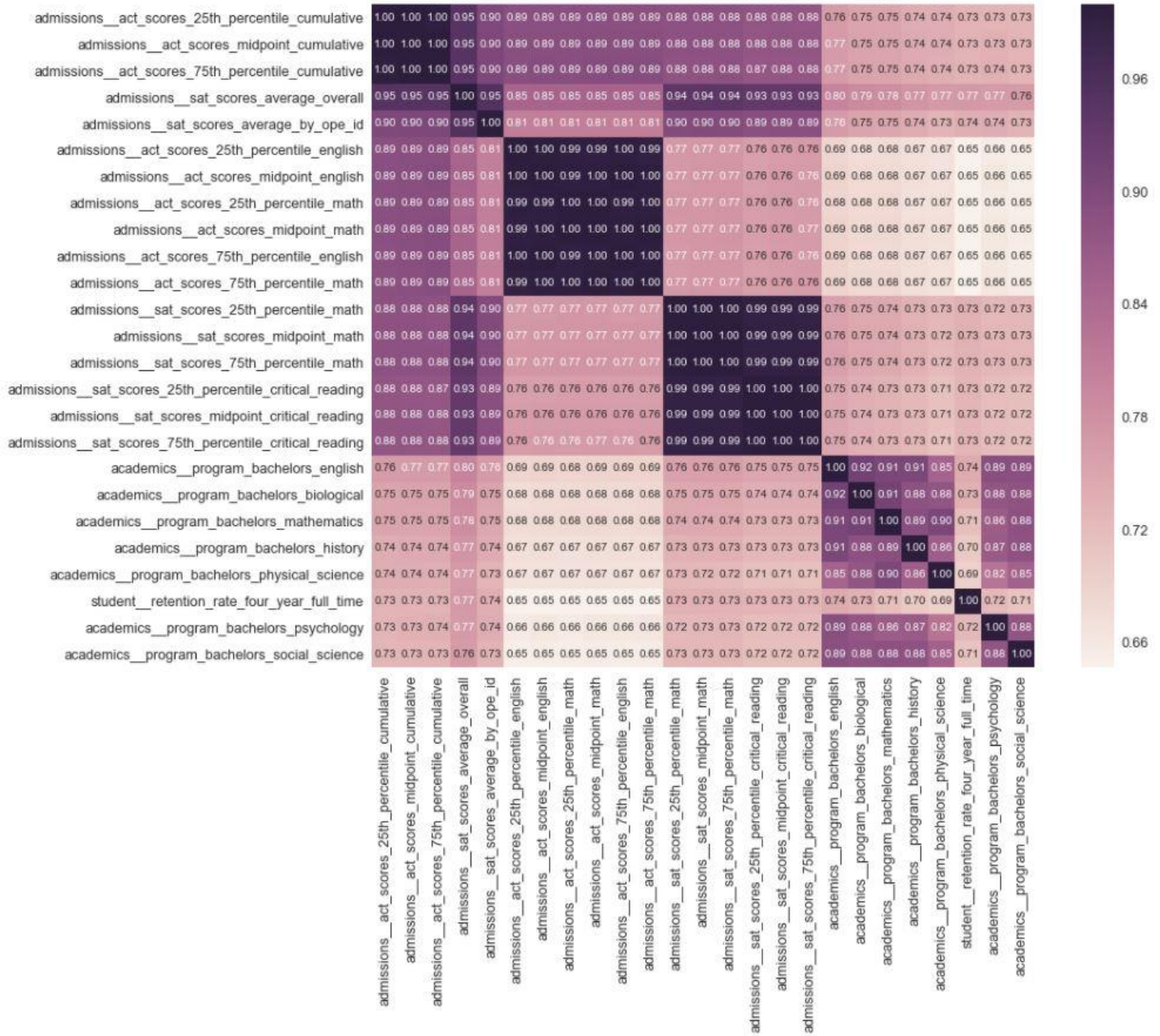
$$y = \left| \frac{-b + \sqrt{b^2 - 4ac}}{2a} - \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right|$$

```
def synthesize(X, colA, colB, colC, newColName, category):

    X[newColName] = abs(
            ((-X[colB] + sqrt(abs(X[colB]**2 - (4 * X[colA] * X[colC])))) / (2 * X[colA])) -
            ((-X[colB] - sqrt(abs(X[colB]**2 + (4 * X[colA] * X[colC])))) / (2 * X[colA]))
        )

    print("column [%s] created for %s." % (newColName, category))
    return X
```

## Feature Reduction

Virtually all attempts to remove seemingly irrelevant or objectively low-ranking [vis-à-vis Gini importance score] features resulted in an increased RMSE on cross-validation, hold-out and public leaderboard submissions. A noteworthy and surprising mention in this category is feature student__share_firstgeneration_parents-middleschool, with an astounding missing-value rate of 99.51%. All models tested with this feature removed always resulted in higher error rates than those with experiments including the sparse feature. The addition of adding 'isMissing'-style binary indicators had no noticeable effect on the RMSE for various features tested.

In the end, the only columns removed from the winning models were those that were perfectly, or near perfectly, correlated with other features – rendering them redundant to the machine learning application:



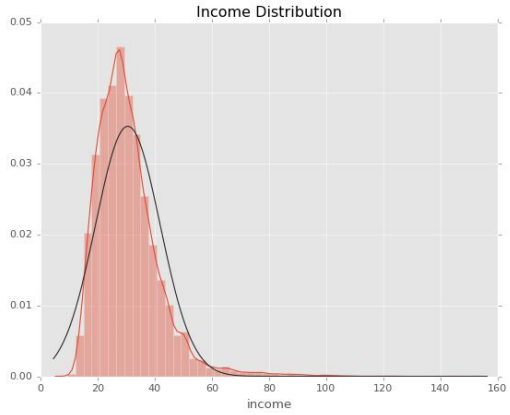

*Note the features with perfect correlation; most of these were removed from the final model.*

## Feature Encoding

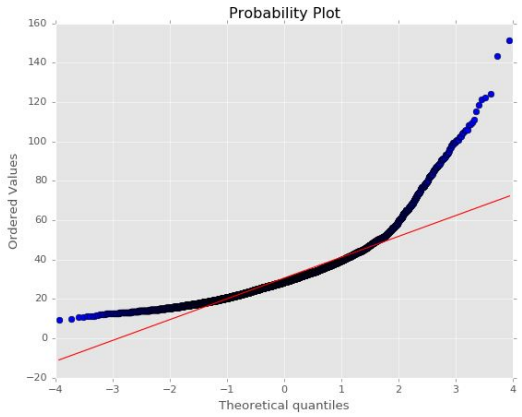
Several features in the data-set presented as categorical and logically ordinal. For these features, mapping was encoded to ensure the order was preserved numerically. By way of example, associate degrees were ranked: 2, bachelors: 4, graduate: 5, etc. Some features that were not logically ordered, but nonetheless exhibit order, a posteriori, due to the specifics of the application in this challenge were encoded ordinally (i.e. state codes). All other categorical data utilized one-hot encoding.

## Dependent Variable Analysis

Income in the training dataset is positively skewed (skw 1.956 / kur 7.97) with a range of 9.4 to 151.5 and an IQR of 12.6. The most frequent value was 28.2.

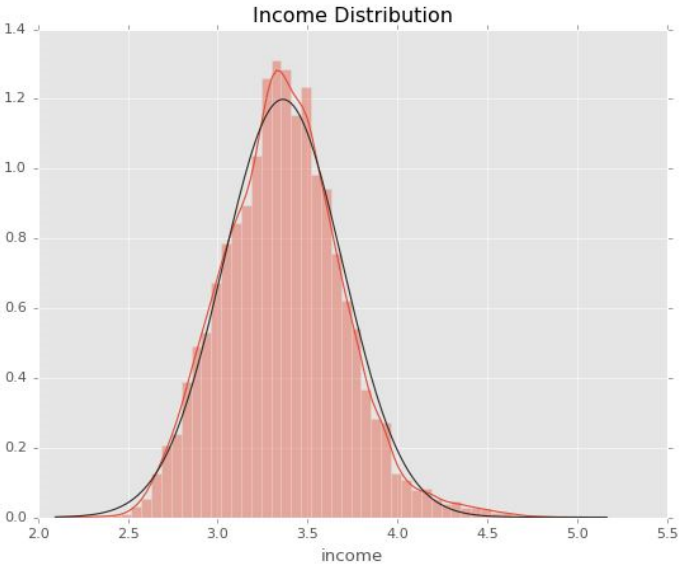


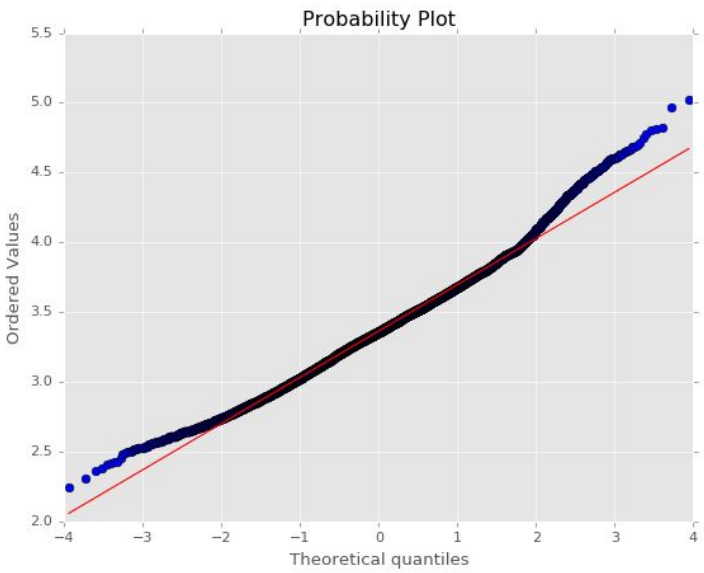

```
Number of unique values: 721,  Number of NULLs: 0
count    17107.000000
mean        30.592003
std         11.302597
min          9.400000
25%         23.000000
50%         28.700000
75%         35.600000
max        151.500000

Skewness: 1.956283
Kurtosis: 7.975607
Mode: 28.200000
```




Natural logarithm smoothing of the label was tested in all 'best score' models, and was found to have either no effect or very minor adverse effect in public leaderboard submissions. Due to scaling of the label range, local CV and hold-out scores were in the RMSE 0.08 to 0.10 +/-.03 range, but end-result of application resulted in nominally worse scores after deconversion. Normalcy of the dependent variable, it seems, is not critical for tree-based machine learning models.

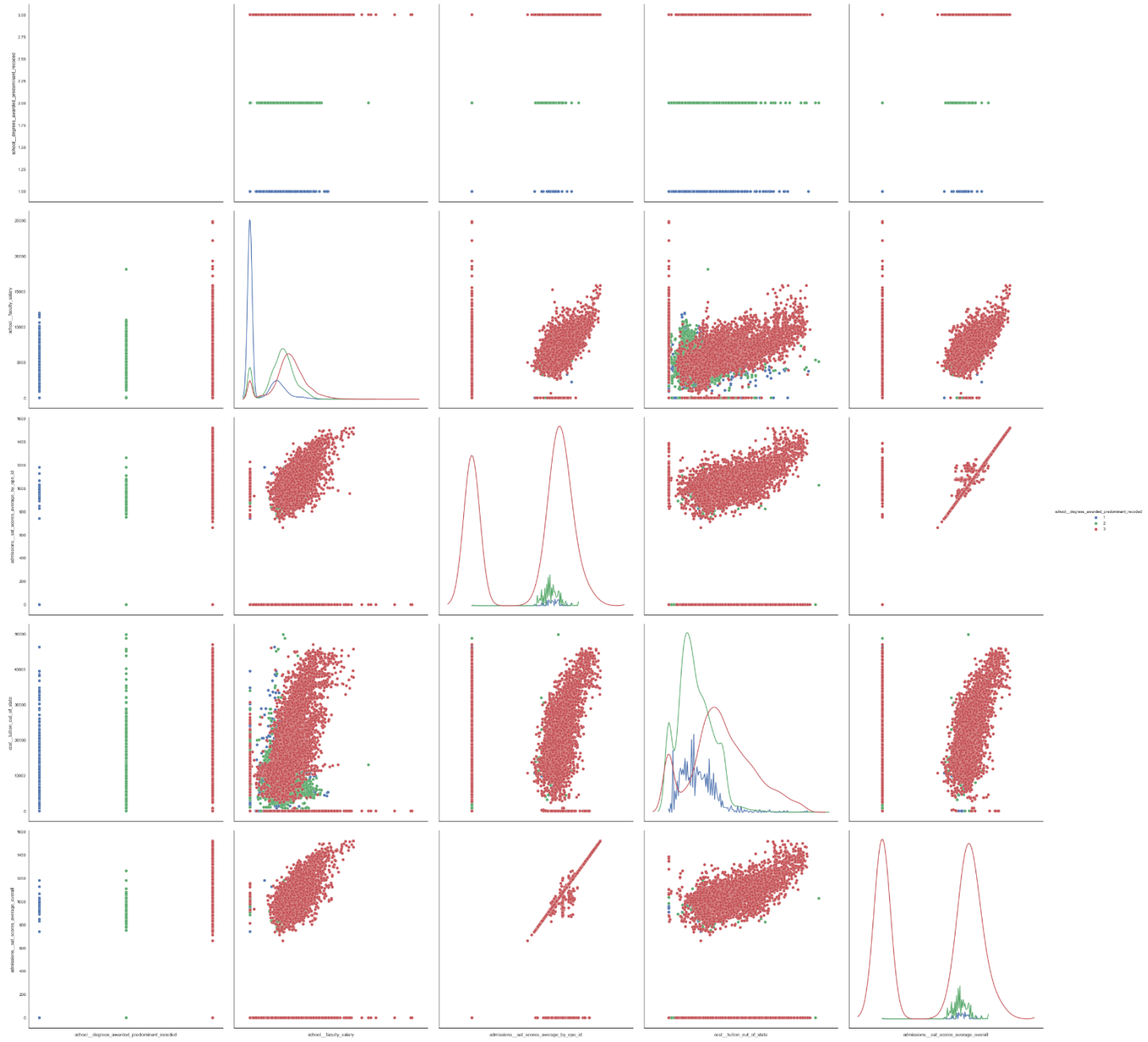
*Natural logarithm scaling improves normalcy of distribution, but has adverse or no effect on leaderboard scores:*



## Dependent Variable Analysis (Contd.)

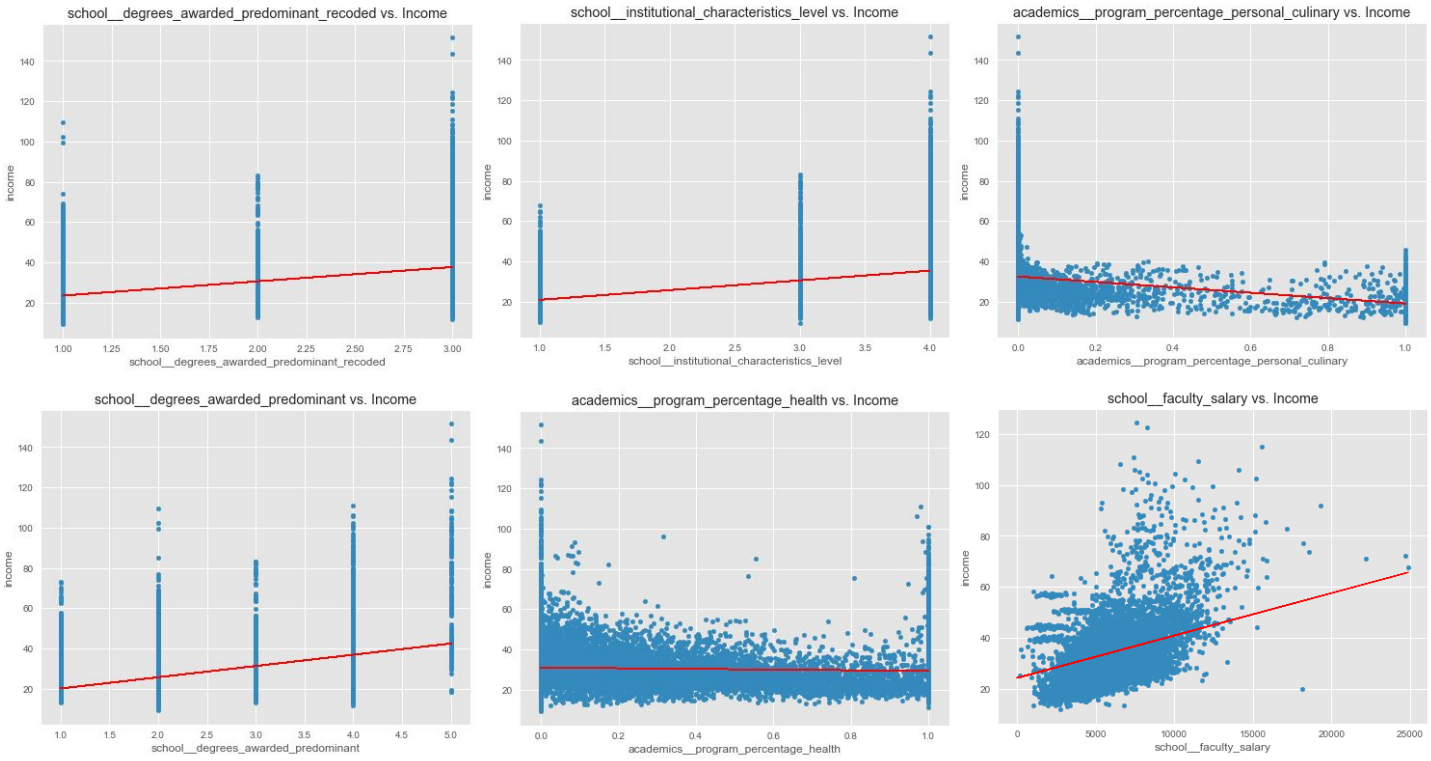
Pairplots were generated against the [initially identified] top 5 'important' features; due to scaling differences some comparisons are not visually optimal. Regardless, I chose to retain the native scale during analysis for three reasons: (1) scaling makes comparisons more accessible, but remove scope in difference from the visuals, (2) No scaling was employed as tree-based models were used for prediction and feature importance analysis, and (3) sufficient information for interpretation was available without modification.



*Pairplot of initial top 5 features, color-coded by most prominent feature: degrees_awarded_predominant_recoded. The impact of the highest encoding value of this feature on other critical predictors, like faculty salary and average SAT scores, is self-evident.*

## Dependent Variable Analysis (Contd.)

The general relationship between the dependent variable and all of the top ranked features were assessed to identify any problem areas in the data that might be good targets for data cleaning. Many features, like faculty salary and demographics age entry, have strong apparent predictive force but were missing thousands of values in the raw data set. Some of these variables were addressed via interpolation, others were given an arbitrary value, and many were left as 0. The key to method selection was to rely solely on empirical results from cross-validation and hold-out scoring, and check the strongest bets against the public leaderboard score.

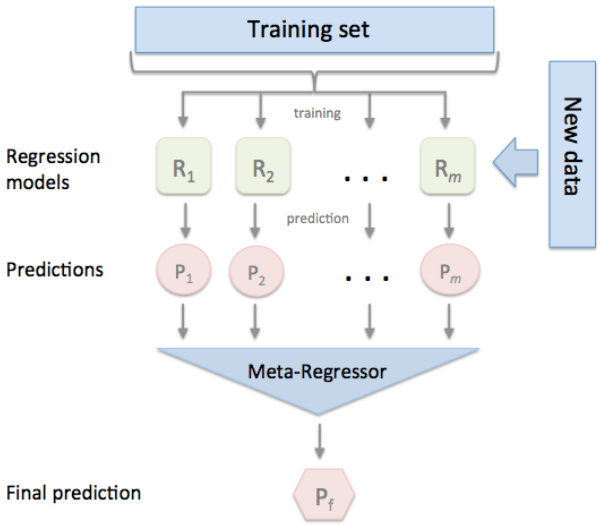



## Predictive Modeling – Baseline and Bagging

Modeling began with producing an initial baseline score against the base data with only light treatment through categorical one-hot encoding and replacing all missing values with 0. A decision-tree was used with default parameters, which produced a private RMSE of ~5.2. From this point forward, a process of evaluating different models (including ensembles), tuning them through parameter grid search, evaluation of Gini scores and iteratively treating the features to improve model performance was used. After reaching a point of producing an RMSE 3.2465 with a tuned extra-trees regressor, I was able to whittle down to an RMSE of 3.1916 through bootstrap aggregating, or "bagging," my three top-performing individual models. Bagging of submissions is a tactic of producing a single output (typically, mean-average) results from the base individual models. This method derives from the subset of coding theory referred to as error correction. There are several common methods one can use for bagging submissions: majority vote, weighted voting, averaging least correlated, etc. but I selected simple mean averaging and managed to adjust my submission RMSE by -0.0549.

## Predictive Modeling – Stacking generalization and Blending

After many attempts to further lower the score through addition of models, further tuning of parameters, minor adjustments to the underlying training data – I could not attain a lower public leaderboard score of RMSE 3.1916 through bagging of submissions. A new approach beyond a crude averaging would be required to make further progress; the next logical progression was to change my approach entirely, leveraging stacked generalization and multiple regressors (ideally, those that produced low Pearson correlation in their outputs and were able to use differing features more effectively). Popular non-linear algorithms for stacking are GBM, KNN, NN, RF and ET (all tested).




Breiman, Leo. "Stacked regressions." Machine learning 24.1 (1996): 49-64. Image source: mlxtend.

My first stacking generalization consisted of an ensemble of three individual regression models, each individually tuned. All three models were trained on the entire training data set, then a simple [untuned, default values] Linear regression was fitted to the outputs (meta-features) of the three individual models.

This approach is similar to bagging, but with regression applied to the predicted labels rather than averaging or voting. This produced a public leaderboard RMSE of 3.1359, an improvement of -0.0557 over simple-average bagging.

Minor score improvement continued through a process of iteration and adding more individually-tuned regressors, at a cost of increased complexity and [drastically] increased execution time. After reaching a best RMSE of 3.0881, a change to the approach was made to further reduce over-fitting and improve error rates against the test data.

In the first stacking approach, all regressors were fit to the same training data, potentially leading to overfitting. The second stacking approach employed a type of blending, where k-1 folds of the data are trained against the individual regressors, but only output of prediction against the k hold-out "validation" fold are added to the level-1 predictions before being fit and final predictions are made.

Once all predictions are made, the output is passed to a simple meta-regressor (in my case, linear regression) for final predictions. This approach ultimately achieved a final RMSE of 2.8992, my top score and an improvement of -0.2367 over the simpler stacking generalizer. The final model was comprised of seven individually tuned regressors, and one meta-estimator with 15 k-folds used, taking just over six hours run-time to produce final predictions.

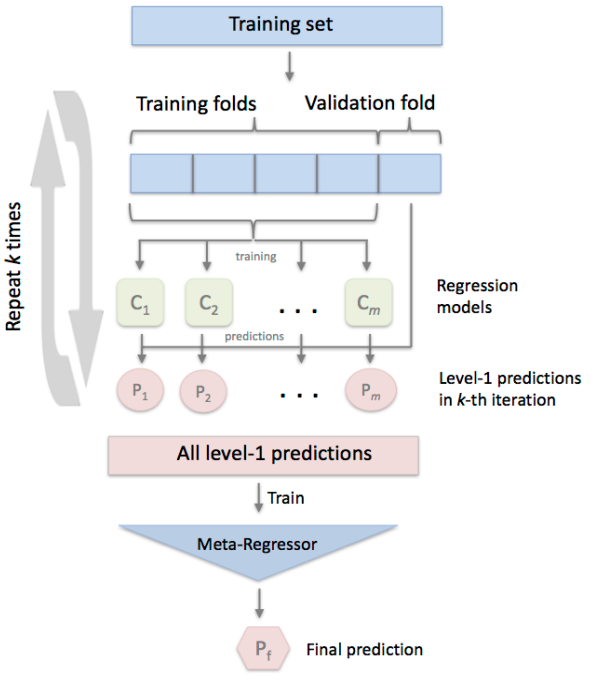



Image source: mlxtend.

## Predictive Modeling – Stacking classifiers with regressors

"*Sometimes, it is useful to allow XGBoost to see what a KNN-classifier sees.*" – WW #2 Kaggle competitor, Marios Michalilidis.

One way of improving regression problems is to provide it with synthesized features that better represent the underlying data. Using a classifier, you can transform the target y-label into evenly spaced bins that may help the regressors make better predictions. For this challenge, I found the optimal number of bins to be three:

- Income values of 56.76 and below are in bin 1
- Income values greater than 56.76 but less than or equal to 104.12 are in bin 2
- Income values greater than 104.12 are in bin 3

A tuned extra-trees classifier was cross-validated and achieved an accuracy rate of ~95% on training data. The income bins were created programmatically through simple discretization (via pandas qcut()), and then the data was fitted against the ET classifier. Predictions were made and saved to the test data as a new feature.

This method produced a minor improvement in the score, from 2.9000 to 2.8992 (Δ -0.0008), but was enough to tip the score into the 2.8XXX range.

## Predictive Modeling – Conclusion

The final model consisted of one classifier algorithm for income binning, seven individually tuned tree-based regressors for regression (below), and one untuned / default linear regression meta-estimator. The results show that graduate income can reasonably be inferred from messy and mostly sparse college scorecard data; I am confident that with more training data and perhaps more time spent interpolating additional missing features, a final RMSE could be reduced into the mid 2-range.

```
Ada-DTR: 3.752920 (*/- 0.292905)
ETR: 3.864760 (*/- 0.208437)
XGB: 3.760834 (*/- 0.165358)
RF: 4.133912 (*/- 0.195558)
gbr: 3.875963 (*/- 0.201890)
Ada-ETr: 4.029058 (*/- 0.198895)
lbgm: 4.038213 (*/- 0.103282)
```