1 How To Use This Document

Highly regulated industries, such as banking and insurance, must comply with government regulations for model validation before a model can be put into production. This includes creating robust model development documentation. DataRobot automates the generation of model documentation, expediting the process required for regulatory compliance and following best practice for reducing model risk.

This document is split into two components: those sections that are automatically produced by DataRobot and those that require further input by the user. The sections in blue italicized font include specific instructions for the documenter and require additional user input of organization-specific information, such as business use cases, data sources, and implementation details. Once the sections are complete, remove the instructions. The remaining sections in non-blue italicized font are automatically populated by DataRobot and require no further input.

Copyright ©2022, DataRobot, Inc.



Table of Contents

* 1 How To Use This Document
* 2 DataRobot Model Development Documentation
* 3 Executive Summary and Model Overview
* 3.1 Model Stakeholders
* 3.2 Model Development Purpose and Intended Use
* 3.3 Model Description and Overview
* 3.4 Overview of Model Results
* 3.5 Model Interdependencies
* 4 Model Data Overview
* 4.1 Feature Association
* 4.2 Data Source Overview and Appropriateness
* 4.3 Input Data Extraction, Preparation, and Quality & Completeness
* 4.4 Data Assumptions
* 5 Model Theoretical Framework and Methodology
* 5.1 Model Development Overview
* 5.2 Model Assumptions
* 5.3 Model Methodology
* 5.3.1 Ordinal scale converter of categorical features
* 5.3.2 Median value-based numeric imputation (V2 with quick median algorithm)
* 5.3.3 Extreme Gradient Boosting Regressor with Grid Search and Early Stopping support
* 5.4 Literature Review and References
* 5.5 Alternative Model Frameworks and Theories Considered
* 5.6 Variable Selection
* 5.6.1 DataRobot Quantitative Analysis
* 5.6.2 Expert Judgement and Variable Selection
* 5.6.3 Final Model Variables
* 5.6.3.1 Model Features and Summary Statistics
* 5.6.3.2 Data Quality Handling Report
* 6 Model Performance and Stability
* 6.1 Model Validation Stability
* 6.1.1 Cross Validation Scores
* 6.1.2 Data Partitioning Methodology
* 6.2 Model Performance (Sample Scores)
* 6.3 Sensitivity Testing and Analysis
* 6.3.1 Lift Chart
* 6.3.2 Key Relationships
* 6.3.3 Sensitivity Analysis (Partial Dependence)
* 7 Model Implementation and Output Reporting
* 7.1 Version Control

2 DataRobot Model Development Documentation

A key component of effective model risk management is sufficiently detailed documentation for model development, implementation, and use, so that reasonable parties unfamiliar with a model can understand how the model operates, its limitations, and its key assumptions. Additionally, model documentation should contain enough detail for an independent party (e.g., independent model validation) to replicate all aspects of the underlying modeling process.

The purpose of this document is not to be prescriptive in format and content, but rather to serve as a guide in creating sufficiently rigorous model development, implementation, and use documentation. The documentation should provide enough evidence to show that the components of the model work as intended, the model is appropriate for its intended business purpose, and that it is conceptually sound.

3 Executive Summary and Model Overview

3.1 Model Stakeholders

Describe the model's purpose and its intended business use. Describe all stakeholders of this model, including their role, line-of-business, and team. This should include stakeholders of model ownership, model development, model implementation, and model risk management.

Model Owner(s): The individual who owns the business risk addressed by the model and provides approval for the model to be used within the line-of-business or enterprise function.

Model Developer(s): The individual responsible for building new models with DataRobot or maintaining existing models.

Model User(s): Those teams who will use the model output as part of their ongoing business operations.

Model Validator(s): The validators are responsible for independent model review and approval prior to its first use.

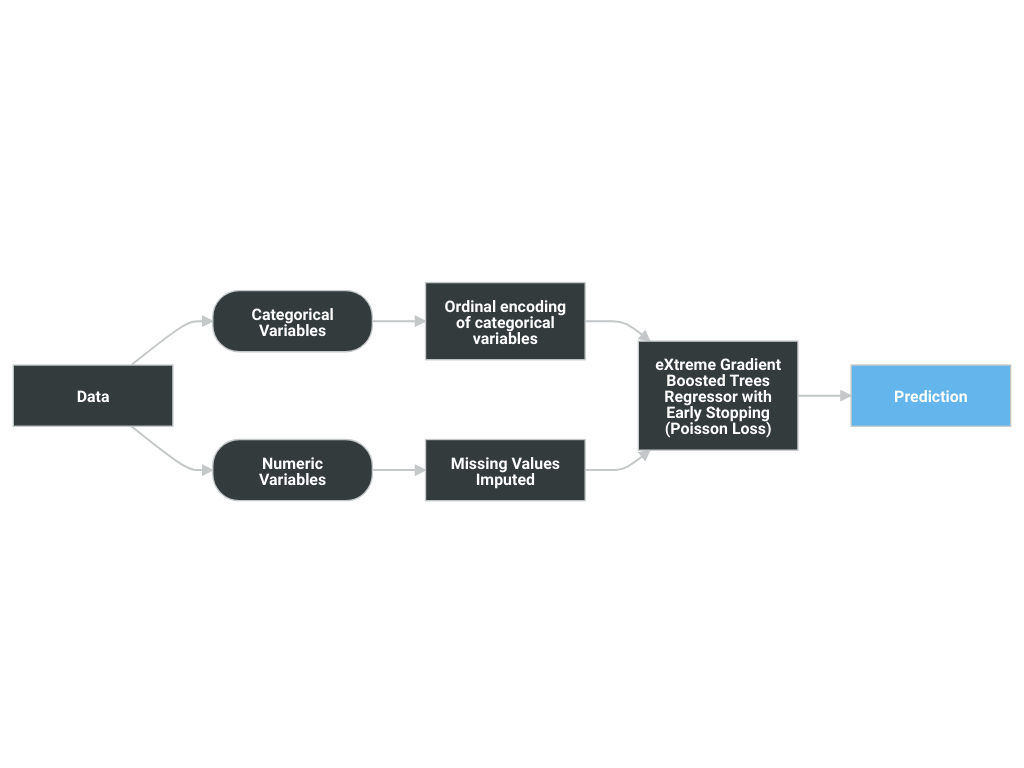
3.2 Model Development Purpose and Intended Use

Describe the model's purpose, including a summary of the business need for this particular model. Concisely describe how the model will be used to address this business problem. Furthermore, describe with great precision all model uses covered by this document. These descriptions will address this statement made in regulatory guidance, FRB SR-11-7, "Even a fundamentally sound model producing accurate outputs consistent with the design objective of the model may exhibit high model risk if it is misapplied or misused."

3.3 Model Description and Overview

The particular model referenced in this document: eXtreme Gradient Boosted Trees Regressor with Early Stopping (Poisson Loss). This model was developed in a project created with v7.3.2 of DataRobot. This model is denoted within DataRobot by the Project ID: 630b857eea4658a27df30253 and the Model ID: 630bef0d83a0e326b957a791. The project was created on 2022-08-28 15:10:54.

The model development workflow process (i.e., the model blueprint) is detailed in the figure below.



A Blueprint represents the high-level end-to-end procedure for fitting the model, including any preprocessing steps, algorithms, and post-processing. It illustrates the many steps involved in transforming input predictors and targets into a model. Each element (or, “node”) in a blueprint can represent multiple steps.

The following elements connect to visualize the blueprint:

* Ordinal encoding of categorical variables
* Missing Values Imputed
* eXtreme Gradient Boosted Trees Regressor with Early Stopping (Poisson Loss)

3.4 Overview of Model Results

DataRobot runs performance testing during the model development process to evaluate model results and reliability. The validation, cross-validation, and holdout (if applicable) out-of-sample performance scores are presented below, as well as the number of observations for each partition. The performance metric used for this project was Gamma Deviance and the project included a total of 1,460 observations. An asterisk (\*) next to a score, whether validation or holdout, indicates that DataRobot used in-sample predictions to derive the score. (In-samples predictions are those that include data from the validation or holdout partitions due to sample size used to build the model.)

|  |  |
| --- | --- |
| Scoring Type | Score (Gamma Deviance) |
| cross\_validation | 0.0127\* |
| holdout | 0.0169\* |
| validation | 0.0089\* |

3.5 Model Interdependencies

Understanding interdependent relationships allows for an enhanced understanding of, and improved ability to manage and aggregate model risk at the company. Explain how this model is interconnected with other models in the model inventory. If the output of this model feeds an interdependent model then the direction of that relationship is "downstream" otherwise it is "upstream." In addition to the directional relationship, also provide a brief description of each interconnected model.

4 Model Data Overview

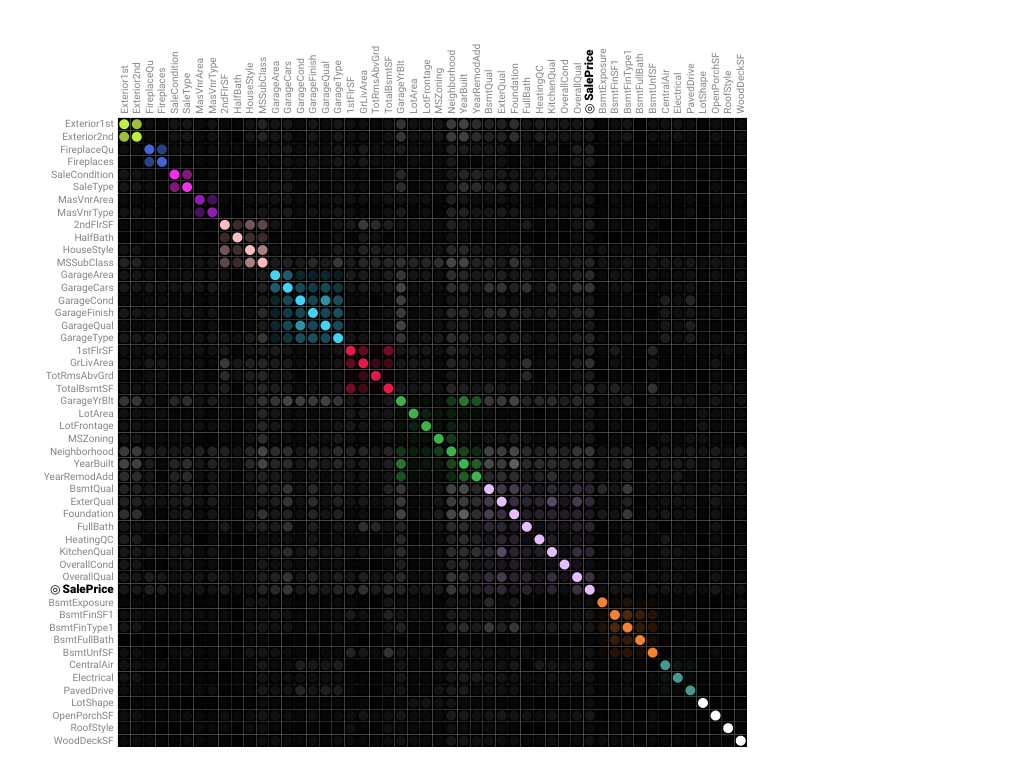
4.1 Feature Association

DataRobot’s Feature Association Matrix is populated by default by features from DataRobot’s Informative Features feature list. The Feature Associations matrix provides information on association strength between pairs of numeric and categorical features that are visually denoted by the opacity of the color (that is, num/cat, num/num, cat/cat, where lighter shades indicate weaker association and vice versa) and feature clusters. Clusters, families of features denoted by color on the matrix, are features partitioned into groups based on their association structure.

Some of the noted benefits of the Feature Association Matrix include:

* Understand the strength and nature of associations within the data;
* Detect families of pairwise association clusters; and,
* Identify clusters of high-association features prior to model building.

The Feature Association Matrix lists up to the top 50 features, selected by Importance Score, on both the X and Y axes, where the intersection of a feature pair provides an indication of their level of association. By default, the matrix displays by the Mutual Information values and sorts by the cluster.



The following are some general takeaways from looking at the matrix above:

* Each dot represents the association between two features (a feature pair), where the opacity of the color denotes the pair-wise strength of association.
* Each cluster is represented by a different color.
* The opacity of color indicates the level of association 0 to 1, between the feature pair. Levels are measured by the set metric, either mutual information or Cramer’s V.
* Shaded gray dots indicate that the two features, while showing some association, are not in the same cluster.
* White dots represent features that were not categorized into a cluster.
* The target feature, if present, is indicated by two small concentric circles next to the feature name.

4.2 Data Source Overview and Appropriateness

Explain how the data is suitable and relevant for the business problem and model use. For example:

Describe how, and from where, the data was obtained.

Provide a detailed description of the data source and its relevance to the business problem being addressed by this model.

Assess whether the data used for model development is appropriate given the populations to which the model will be applied.

If the model development and model implementation data sources differ, provide a detailed explanation justifying the use of different data sources.

4.3 Input Data Extraction, Preparation, and Quality & Completeness

Provide a detailed description of the data extraction and preparation process, and discuss any analysis conducted to confirm the data are complete and of sufficient quality (e.g., data validation). Include a detailed description of the data extraction process, hierarchical by extraction and preparation stage, and calling sequence. Provide data extraction code (e.g., SQL, Spark, etc.) in the Appendix.

Review and comment on any data weaknesses and limitations and their probable potential effects on the model. For example, data truncation, extraction timing, through-the-cycle data, and data exclusions could potentially cause unintended effects on the model.

4.4 Data Assumptions

Comment on data assumptions, the potential effects on the model, and any mitigating data controls. For example, assumptions related to data truncation, extraction timing, through-the-cycle data, reliability of source system, manual data overrides or imputation, and data exclusions could potentially cause unintended effects on the model.

5 Model Theoretical Framework and Methodology

5.1 Model Development Overview

DataRobot simplifies model development by performing a parallel heuristic search for the best model or ensemble of models, based on both the characteristics of the data and the prediction target. While some machine learning techniques tend to consistently outperform others, it is rarely possible to say in advance which will perform best for a given business problem. Therefore, during the modeling process, DataRobot develops dozens of independent challenger models, exposes the details of how these models were built and how they perform, and enables the user to select the best model for the particular business problem being addressed.

The fundamental workflow within DataRobot for model development is as follows:

* Rapid Data Ingestion: User creates a modeling dataset that includes the prediction target and loads into DataRobot
* Target Selection: User selects the prediction target; DataRobot detects whether the target is categorical or continuous. If the target is categorical, DataRobot selects and builds classification blueprints. If the target is continuous, DataRobot selects and builds regression blueprints. DataRobot also selects an optimization performance metric based on the type of supervised learning problem, which can also be changed by the user
* Automated Data Preparation: DataRobot analyzes the input data and automatically performs advanced preprocessing steps that are discussed in detail in this document. DataRobot also automatically partitions the input dataset into learning, validation and holdout dataset; these can also be defined by the user.
* DataRobot uses information about the selected target variable and predictors to define a set of candidate blueprints for analysis. It then trains models for each blueprint and ranks them on the model Leaderboard based on an out-of-sample validation accuracy score.
* Transparent Model Evaluation and Selection: DataRobot has built-in diagnostic tools to assess model accuracy and performance. Once DataRobot has trained and tested models, users can access them from the Leaderboard. From there, users can review model accuracy and, using built-in model diagnostic tools, understand how each independently built model performs. DataRobot provides many metrics for evaluating model accuracy, such as AUC, Log-Loss and RMSE. DataRobot's Leaderboard actively tracks performance of candidate models using out-of-sample data for comparison purposes.
* Model Deployment and Monitoring: Once the final model is selected, DataRobot provides efficient solutions for deployment (i.e., model implementation) and monitoring. These features enable the model owner to effectively manage model controls in accordance with Model Risk Management standards and policies.

5.2 Model Assumptions

This section should include model limitations, potential effects, and any mitigating controls in place. Limitations come in part from weaknesses in the model due to its various shortcomings, approximations, and uncertainties. Regulatory guidance refers to limitations as "...a consequence of assumptions underlying a model that may restrict the scope to a limited set of specific circumstances and situations." This section should include model limitations, potential effects, and any mitigating controls in place. Also include details here about the implementation of the models, what data will be used for scoring and why it is reasonable to think that the training data and the scoring data will be similar.

Machine learning methods can produce more accurate predictive models than traditional statistical regression methods because they are more flexible and rely less on statistical assumptions than traditional regression methods. For instance, ordinary least squares regression requires that the Gauss Markov assumptions are supported, which ensures that the model is unbiased and efficient.

Traditional statistical regression techniques rely on formal hypothesis testing for variable significance and feature selection (e.g., t-test, p-value, standard error). These hypothesis tests tend to have distributional and independence assumptions that may not be supported by the data. Machine learning methods, on the other hand, offer more flexibility in defining the model structure, which typically results in better model performance. Because machine learning includes methods that do not rely on formal hypothesis testing to demonstrate model validity, and because heuristic-style feature selection methods (e.g., stepwise selection) are not used in most machine learning approaches, no such distributional assumptions are required. In this case, the only assumption being made is that the model training data is representative of the future scoring data. Of course, these assumptions must be closely monitored and tracked by the model's ongoing performance monitoring process.

A common limitation of machine learning methods is the potential for overfitting. Overfitting occurs when the model is trained too closely to the underlying training data and does not perform well out-of-sample. DataRobot utilizes a robust cross-validation and holdout methodology to ensure model performance is sound, reducing the risk of over-fitting.

5.3 Model Methodology

The modeling workflow consists of the following elements, which connect to visualize the modeling blueprint:

* Ordinal encoding of categorical variables
* Missing Values Imputed
* eXtreme Gradient Boosted Trees Regressor with Early Stopping (Poisson Loss)

The following subsections include details for each node of the modeling blueprint.

5.3.1 Ordinal scale converter of categorical features

For a categorical feature, convert categorical levels to an ordinal scale. The ordinal scale is 0 to (unique values of categorical\_var) - 1. Rare categories (=other) and missing values are encoded as -1 and -2, respectively. Mapping is based on the lexicographic ordering of the categorical values, the frequency of the levels, the response, or is done randomly.

Ordinal encoding is effective for tree-based models, as it usually performs as well as one-hot encoding but requires fewer computational resources (memory and cpu).

Ordinal encoding, however, does not work for linear methods.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| select | add\_cols\_metadata | If specified, add -cols to metadata. values: [False, True] | False |
| select | add\_maps\_metadata | If specified, add -maps to metadata. values: [False, True] | False |
| multi | card\_max | Maximum number of categorical feature levels allowed. If None, a feature with any number of levels is allowed. values: {'int': [1, 9999999], 'select': None} | None |
| select | method | Method used in the encoding. None: uses random\_scale. random: random ordering of levels, lex: lexicographical ordering by category level names, freq: frequency ordering from least frequent to most frequent, resp: response ordering. values: ['None', 'random', 'lex', 'freq', 'resp'] | freq |
| int | min\_support | Minimum number of levels required for a category to be represented on the ordinal scale. If a category level count is below the minimum, it will be grouped with other small cardinality levels or encoded as a missing value, depending on the value of other\_category. values: [1, 99999] | 5 |
| int | offset | Shift the ordinal scale of ordinal encoder values: [0, 99999] | 0 |
| bool | other\_category | If True, small cardinality values are mapped to a dedicated value (-1), otherwise they are encoded as missing values (-2). values: [False, True] | True |
| bool | random\_scale | Applies if method is None. If random\_scale is True, random ordering is used for the ordinal scale. If it is False, lexicographical ordering is used. values: [False, True] | True |
| int | seed | The RNG seed. values: [0, 99999] | 1234 |

5.3.2 Median value-based numeric imputation (V2 with quick median algorithm)

For a numeric feature, impute rows of missing values with median value (V2).

Impute missing values on numeric variables with their median and create indicator variables to identify records that were imputed. A quick median algorithm (based on np.partition) is implemented to compute median feature value.

Imputation strategy:

A numeric feature is imputed with the median value if there are enough finite values in the feature samples used to train a numeric imputation task (e.g., > t, default: 50) and there are rows with NaN or infinite values in the samples to be imputed.

After imputation, the imputed numeric features will be scaled if the argument s is set to True. The feature will use scaled rounding (i.e., rounding to a logarithmic scale).

Imputation indicator:

The indicator column (0, 1) is added to indicate imputed rows if the numeric feature is imputed with : 1) the median value and with least one row with NaN and 2) at least two unique values.

Example:

An imputation task is initialized with t=2.

Input numeric features of this task:

feature0, feature1, feature2, feature3

1, 2, np.nan, np.nan

2, 3, np.nan, 18

3, 2, np.nan, 16

4, 1, 13, 14

20, 1, 45, 46

Output numeric features of this task:

feature0, feature1, feature2, feature2-mi, feature3, feature3-mi

1, 2, 45, 1, 18, 1

2, 3, 45, 1, 18, 0

3, 2, 45, 1, 16, 0

4, 1, 13, 0, 14, 0

20, 1, 45, 0, 46, 0

In the imputation output, median value imputation is run on feature2 and feature3. The feature2-mi is the indicator column for the imputation on feature2. The feature3-mi is the indicator column for the imputation on feature3.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| bool | scale\_small | True if small values (range of the numeric variable is <= 1) are to be scaled. values: [False, True] | True |
| int | threshold | Minimum number of required finite elements in a column to impute the data onto NaNs and INFs. values: [1, 99999] | 10 |

5.3.3 Extreme Gradient Boosting Regressor with Grid Search and Early Stopping support

Gradient Boosting Machines:

Gradient Boosting Machines (or Generalized Boosted Models, depending on who you ask to explain the acronym ‘GBM’) are an advanced algorithm for fitting extremely accurate predictive models. GBMs have won a number of recent predictive modeling competitions and are considered by many data scientists to be the most versatile and useful predictive modeling algorithm. GBMs require very little preprocessing, elegantly handle missing data, strike a good balance between bias and variance, and are typically able to find complicated interaction terms, making them a useful “Swiss army knife” of predictive models.

GBMs are a generalization of Freund and Schapire’s adaboost algorithm (1995) that handles arbitrary loss functions. They are very similar in concept to random forests, in that they fit individual decision trees to random re-samples of input data, where each tree sees a bootstrap sample of the rows of the dataset and N arbitrarily chosen columns, where N is a configurable parameter of the model. GBMs differ from random forests in a single major aspect: rather than fitting the trees independently, the GBM fits each successive tree to the residual errors from all the previous trees combined. This is advantageous, as the model focuses each iteration on the examples that are most difficult to predict (and therefore most useful to get correct).

Due to their iterative nature, GBMs are almost guaranteed to overfit the training data, given enough iterations. Therefore, the 2 critical parameters of the algorithm are the learning rate (or how fast the model fits the data) and the number of trees the model is allowed to fit. It is critical to tune one of these 2 parameters, and when done correctly, GBMs are capable of finding the exact point in the training data where overfitting begins, and halt one iteration prior to that point. In this manner GBMs are usually capable of squeezing every last bit of information out of the training set and producing a model with the highest possible accuracy without overfitting.

Extreme Gradient Boosting (XGBoost) is a very efficient, parallel version of GBM that has won a large number of Kaggle competitions. The base algorithm is very similar to GBM in R or in Python, but it has been heavily optimized and tweaked for faster runtimes and higher predictive accuracy.

Loss Function:

The XGBoost regressor uses least-squares loss by default, but can also use: tweedie loss for zero-inflated positive distributions, poisson loss for count problems, and gamma loss for right skewed positive distributions.

Grid Search Support:

Grid search is supported in this task. During training, grid search is run to estimate the optimal XGBoost parameter values that yield the best performance (evaluated by the configured loss function ). The grid search runs on a 70/30 train/test split within the training data; the estimated score uses 30% of the training data split. After the grid search completes and best tuning parameters are found, the final model is retrained on 100% of training data. Validation scores of the final model are different from the validation scores of the grid search.

Grid search is run on the task parameter with one of the following types: ‘intgrid’, ‘floatgrid’, ‘listgrid(int)’, ‘listgrid(float)’, ‘selectgrid’, or ‘multi’. Refer to the Parameters section for details of task parameter definitions.

For each grid search parameter, the search space is defined by the parameter values. Refer to the Parameters section for details of task parameter definitions.

Early Stopping Support:

The Early Stopping Extreme Gradient Boosting model also uses early stopping to determine the best number of trees. Early stopping is a method for determining the number of trees to use for a boosted trees model. The training data is split into a training set and a test set, and at each iteration the model is scored on the test set. If test set performance decreases for 200 iterations (tunable in Advanced Tuning), the training procedure stops and the model returns the fit from the best tree seen so far. The approach saves time by not continuing past the point where it is clear that the model is overfitting and further trees will not result in more accuracy.

Note that the early stopping test set uses a 90/10 train/test split within the training data for a given model. For example, a 64% model on the Leaderboard will internally use 57.6% of the data for training, and 6.4% of the data for early stopping. A 100% model on the leaderboard will internally use 90% of the data for training and 10% of the data for early stopping.

Since the early stopping test set was used for early stopping, it cannot be used for training.

This limitation also applies to grid search: within the grid search train/test split, the model will use a 90/10 train/test split for early stopping.

Please note: DataRobot DOES NOT use the early stoping logic from open-source XGBoost (also known as early\_stopping\_rounds). DataRobot instead uses custom early stopping logic with 2 parameters: smooth\_interval and interval. DataRobot does a moving average of the loss over the last smooth\_interval iterations, and then stops after interval rounds of increasing loss.

To replicate behavior similar to open-source XGBoost, you can set smooth\_interval=2 and then interval=desired\_early\_stopping\_rounds. This will still apply some smoothing to the loss function, resulting in it stopping a little bit later than open-source XGBoost.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| select | base\_margin\_initialize | If True, the intercept is initialized to the log odds of the target. values: [False, True] | True |
| floatgrid | colsample\_bylevel | Subsample of the features before each split in a tree. values: [0.1,1] | 1.0 |
| floatgrid | colsample\_bytree | Subsample ratio of columns when constructing each tree. By default, the value of colsample\_bytree for XGBoost classes is 1.0. However, based on the training data, DataRobot may choose a different initial value for this parameter. values: [0,1] | 0.3 |
| int | interval | Interval for early stopping. Once the model has hit “smooth\_interval” iterations, the early stopping logic checks for errors increasing at a rate of “interval” iterations in a row. For example, if smooth\_interval=200 and interval=10, XGBoost will run at least 200 iterations, and will early stop if the moving average of loss for the last 200 iterations increases for 10 iterations in a row. Higher values will make XGBoost run longer. values: [2, 500] | 10 |
| floatgrid | learning\_rate | Shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate (lr) and n\_estimators(n). values: [5e-4,1] | 0.05 |
| select | loss | Loss function to be optimized. ‘ls’ refers to least squares regression. values: ['ls', 'labs', 'poisson', 'tweedie', 'gamma'] | poisson |
| int | max\_bin | Used when tree\_method is set to ‘hist’. Maximum number of discrete bins to bucket continuous features. Increasing this number improves the optimality of splits at the cost of higher computation time. values: [16, 2048] | 256 |
| floatgrid | max\_delta\_step | Maximum delta step allowed for each tree’s weight estimation. If the value is set to 0, there is no constraint. Setting to a positive value makes the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Setting it to a value of 1-10 might help control the delta step update. values: [0,100] | 0.0 |
| intgrid | max\_depth | Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for optimal performance; the best value depends on the interaction of the input variables. The deeper the tree, the more variable interactions the model can capture. For frozen models that have larger sample sizes than the parent model, the max\_depth value is increased to retain similar accuracy. values: [1, 16] | 3 |
| floatgrid | min\_child\_weight | Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min\_child\_weight, the building process will give up further partitioning. In linear regression mode, this simply corresponds to the minimum number of instances needed to be in each node. The larger the value, the more conservative the algorithm will be. values: [0.01,float(1e5)] | 1.0 |
| floatgrid | min\_split\_loss | Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger the value, the more conservative the algorithm will be. values: [0,1e5] | 0.01 |
| float | missing\_value | Float value that should be treated as a missing value. When mono\_up or mono\_down is set, missing value will be set to -9999.0. values: [float(-1e5),float(1e5)] | -9999.0 |
| string | mono\_down | ID of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. | no |
| string | mono\_up | ID of the featurelist that defines the set of features with a monotonically increasing relationship to the target. | no |
| int | n\_estimators | Number of boosting stages to perform. Gradient boosting is fairly robust to overfitting, so a larger number usually results in better performance. values: [1,20000] | 1000 |
| intgrid | num\_parallel\_tree | Number of parallel trees created in each boosting stage. When this value is greater than 1, the model becomes a gradient-boosted random forest with (num\_parallel\_tree \* n\_estimators) trees. values: [1,16] | 1 |
| intgrid | random\_state | Seed used in the random number generator 'values': [0, int(1e9)] | 1234 |
| multi | reg\_alpha | L1 regularization term on weights; increasing this value will make the model more conservative. values: {'floatgrid': [0, 1e6], 'select': ['auto']} | 0.0 |
| multi | reg\_lambda | L2 regularization term on weights. Increasing this value will make the model more conservative. values: {'floatgrid': [0, 1e6], 'select': ['auto']} | 1.0 |
| float | scale\_pos\_weight | Scaling factor for examples in the positive class. values: [0,float(1e9)] | 1.0 |
| int | smooth\_interval | Moving average interval for early stopping. Loss over the last n intervals is averaged for determining early stopping. For example, if smooth\_interval=200, XGBoost will not stop early until it hits at least 200 iterations. After 200 iterations, it will use a moving average of loss over the last 200 iterations to determine early stopping. This helps remove noise in the loss function. Higher values will make XGBoost run longer. values: [2, 1000] | 200 |
| floatgrid | subsample | Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees, which will prevent overfitting. | 0.7 |
| select | tree\_method | Tree construction algorithm to use. ‘auto’: Heuristic to choose the faster algortihm. For small to medium datasets (<4M rows), exact greedy will be used. For large datasets (>=4M rows), approximate algorithm will be used. ‘exact’:Exact greedy algorithm. ‘approx’:Approximate greedy algorithm using sketching and histogram. ‘hist’:Fast histogram-optimized approximate greedy algorithm. It uses some performance improvements (e.g., bin caching). values: ['auto', 'exact', 'approx', 'hist'] | auto |
| float | tweedie\_p | Power parameter of the tweedie distribution. Only applicable when Tweedie loss is used. values: [1, 2] | 1.5 |

5.4 Literature Review and References

* [1] Acuna, Edgar, and Caroline Rodriguez. “The treatment of missing values and its effect on classifier accuracy.” Classification, Clustering, and Data Mining Applications. Springer Berlin Heidelberg, 2004. 639-647. https://link.springer.com/chapter/10.1007/978-3-642-17103-1\_60
* [2] Feelders, Ad. “Handling missing data in trees: Surrogate splits or statistical imputation?” Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 1999. 329-334. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.7991&rep=rep1&type=pdf
* Chen, T, and He, T. “Higgs boson discovery with boosted trees.” Cowan et al., editor, JMLR: Workshop and Conference Proceedings. No. 42. 2015. http://www.jmlr.org/proceedings/papers/v42/chen14.pdf
* Freund, Yoav, and Robert E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting.” Journal of computer and system sciences 55.1 (1997): 119-139. https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic\_generalization.pdf
* Friedman, Jerome H. “Greedy Function Approximation: A Gradient Boosting Machine.” Annals of statistics (2001): 1189-1232. https://statweb.stanford.edu/~jhf/ftp/trebst.pdf
* Hastie, T., Tibshirani, R., and Friedman, J.. “Elements of Statistical Learning.” Springer, 2009. http://statweb.stanford.edu/~tibs/ElemStatLearn/
* Breiman, Leo. Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley, 1997. https://statistics.berkeley.edu/sites/default/files/tech-reports/486.pdf

5.5 Alternative Model Frameworks and Theories Considered

As stated by regulatory guidance, comparison with alternative theories and approaches provides guidance for final model selection and is a fundamental component of a sound modeling process.

DataRobot develops dozens of alternative models, exposes the details of how these models were built and how they perform, and enables the user to select the best model for the particular business problem being addressed.

During the model development process, DataRobot considered the following alternative models. The final model was selected based on model performance as well as an analysis of model diagnostics and expert business judgment.

The performance metric used for this project was Gamma Deviance. The model types considered during the model selection process included the following models, which are sorted by the Holdout score.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Validation Score | Cross Validation Score | Holdout Score | Sample Percentage |
| eXtreme Gradient Boosted Trees Regressor (Poisson Loss) | 0.0098 | 0.0136 | 0.018 | 80.0 |
| RuleFit Regressor | 0.0114 | 0.0144 | 0.0189 | 80.0 |
| eXtreme Gradient Boosted Trees Regressor (Gamma Loss) | 0.0095 | 0.0152 | 0.0181 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor with Early Stopping (Gamma Loss) - Forest (10x) | 0.0098 | 0.0151 | 0.0186 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor | 0.01 | 0.0145 | 0.0189 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor with Early Stopping (Gamma Loss) | 0.0093 | 0.0149 | 0.019 | 63.9727 |
| AVG Blender | 0.0087 | 0.0132 | 0.0191 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor (learning rate =0.01) | 0.0102 | 0.0143 | 0.0195 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor (Tweedie Loss: 1.5) | 0.0094 | 0.0146 | 0.0196 | 63.9727 |
| Support Vector Regressor (Radial Kernel) | 0.0106 | 0.0159 | 0.0198 | 63.9727 |
| Support Vector Regressor (Linear Kernel) | 0.0108 | 0.0154 | 0.0199 | 63.9727 |
| Light Gradient Boosting on ElasticNet Predictions (Gamma Loss) | 0.0101 | 0.0144 | 0.0201 | 63.9727 |
| Gradient Boosted Trees Regressor (Least-Squares Loss) | 0.0115 | 0.0165 | 0.0201 | 63.9727 |
| Elastic-Net Regressor (mixing alpha=0.5 / Gamma Deviance) | 0.0101 | 0.0149 | 0.0202 | 63.9727 |
| Light Gradient Boosted Trees Regressor with Early Stopping (Gamma Loss) | 0.0118 | 0.0167 | 0.0205 | 63.9727 |
| Elastic-Net Regressor (L2 / Gamma Deviance) | 0.0102 | 0.015 | 0.0205 | 63.9727 |
| eXtreme Gradient Boosted Trees Regressor (Gamma Loss) with Unsupervised Learning Features | 0.0108 | 0.0156 | 0.0206 | 63.9727 |
| Elastic-Net Regressor (L1 / Gamma Deviance) | 0.01 | 0.0149 | 0.0207 | 63.9727 |
| Nystroem Kernel SVM Regressor | 0.0104 | 0.015 | 0.0212 | 63.9727 |
| Ridge Regressor | 0.0136 | 0.0221 | 0.0215 | 63.9727 |
| Elastic-Net Regressor (L2 / Poisson Deviance) | 0.0108 | 0.0156 | 0.0216 | 63.9727 |
| Elastic-Net Regressor (L2 / Tweedie Deviance) | 0.0103 | 0.0158 | 0.0219 | 63.9727 |
| RandomForest Regressor | 0.0126 | N/A | 0.0219 | 63.9727 |
| Elastic-Net Regressor (L2 / Gamma Deviance) with Binned numeric features | 0.0117 | 0.0161 | 0.0221 | 63.9727 |
| RandomForest Regressor (Shallow) | 0.013 | 0.0192 | 0.0224 | 63.9727 |
| Generalized Additive Model (Gamma Loss) | 0.0124 | 0.0179 | 0.0224 | 63.9727 |
| ExtraTrees Regressor | 0.0131 | N/A | 0.0225 | 63.9727 |
| LightGBM Random Forest Regressor | 0.013 | 0.0187 | 0.0226 | 63.9727 |
| Ridge Regression | 0.011 | 0.0193 | 0.0232 | 63.9727 |
| Keras Residual AutoInt Regressor using Training Schedule (3 Attention Layers with 2 Heads, 2 Layers: 100, 100 Units) | 0.0114 | 0.0188 | 0.0235 | 63.9727 |
| Dropout Additive Regression Trees Regressor (Gamma Loss) (15 leaves) | 0.0155 | N/A | 0.0235 | 63.9727 |
| Gaussian Process Regressor with Matern Kernel (nu=0.5) | 0.0117 | 0.0186 | 0.0245 | 63.9727 |
| Adaboost Regressor | 0.0168 | N/A | 0.0248 | 63.9727 |
| Keras Residual Cross Network Regressor using Training Schedule (3 Cross Layers, 4 Layers: 100, 100, 100, 100 Units) | 0.0112 | 0.8468 | 0.0249 | 63.9727 |
| Keras Slim Residual Neural Network Regressor using Adaptive Training Schedule (1 Layer: 64 Units) | 0.0142 | 0.0197 | 0.026 | 63.9727 |
| Keras Deep Residual Neural Network Regressor using Training Schedule (3 Layers: 512, 64, 64 Units) | 0.0132 | 0.0187 | 0.0262 | 63.9727 |
| Keras Slim Residual Neural Network Regressor using Training Schedule (1 Layer: 64 Units) | 0.012 | 0.0174 | 0.0281 | 63.9727 |
| Generalized Additive2 Model (Gamma Loss) | 0.0156 | N/A | 0.0306 | 63.9727 |
| Keras Residual Neural Factorization Machine Regressor using Training Schedule (2 Layers: 100, 100 Units) | 0.0153 | 0.8782 | 0.0311 | 63.9727 |
| Gaussian Process Regressor with Radial Basis Function Kernel | 0.0141 | 6434.6088 | 0.0312 | 63.9727 |
| Keras Deep Residual Neural Network Regressor using Training Schedule (2 Layers: 512, 512 Units) | 0.0158 | N/A | 0.0312 | 63.9727 |
| Keras Deep Self-Normalizing Residual Neural Network Regressor using Training Schedule (3 Layers: 256, 128, 64 Units) | 0.0185 | N/A | 0.0331 | 63.9727 |
| Auto-tuned K-Nearest Neighbors Regressor (Euclidean Distance) | 0.0224 | N/A | 0.0352 | 63.9727 |
| Auto-tuned Stochastic Gradient Descent Regression | 0.0189 | N/A | 0.0404 | 63.9727 |
| Decision Tree Regressor | 0.0298 | N/A | 0.0416 | 63.9727 |
| Eureqa Generalized Additive Model (10000 Generations) (Gamma Loss) | 0.0309 | N/A | 0.0492 | 63.9727 |
| Eureqa Generalized Additive Model (1000 Generations) (Gamma Loss) | 0.0327 | N/A | 0.0501 | 63.9727 |
| Eureqa Generalized Additive Model (40 Generations) (Gamma Loss) | 0.0416 | N/A | 0.0662 | 63.9727 |
| Linear Regression | 62393.0381 | N/A | 0.0677 | 63.9727 |
| Gaussian Process Regressor with Matern Kernel (nu=1.5) | 0.1199 | N/A | 0.1654 | 63.9727 |
| Mean Response Regressor | 0.1199 | N/A | 0.1654 | 63.9727 |
| Keras Wide Residual Neural Network Regressor using Training Schedule (1 Layer: 1536 Units) | 0.1199 | N/A | 0.1654 | 63.9727 |
| Gaussian Process Regressor with Matern Kernel (nu=2.5) | 0.1199 | N/A | 0.1654 | 63.9727 |
| TensorFlow Deep Learning Regressor | 0.0832 | N/A | 1.1668 | 63.9727 |
| Eureqa Regressor (Instant Search: 40 Generations) | 0.0521 | N/A | 41780.8084 | 63.9727 |
| Eureqa Regressor (Default Search: 3000 Generations) | 0.0367 | N/A | 82876.8218 | 63.9727 |
| Eureqa Regressor (Quick Search: 250 Generations) | 0.0522 | N/A | 107062.065 | 63.9727 |

5.6 Variable Selection

The model's variable selection process includes a balance of quantitative analysis and key domain knowledge about the underlying business problem (i.e., expert judgment). The subsections below describe:

* DataRobot Quantitative Analysis: key components related to variable selection that are automated by DataRobot
* Expert Judgment and Variable Selection: summary of the expert judgment used during the variable selection process.
* Final Model Variables: final feature list chosen

5.6.1 DataRobot Quantitative Analysis

A feature list is a defined set of features (variables) that DataRobot can use for modeling. DataRobot automatically creates three feature lists (described below) for each project. Users, however, can create customized feature lists that contain a subset of the total feature set, and use the new list to train new, alternative models. The default lists are described below:

* Informative Features (default): Features that pass a "reasonableness" check that determines whether they contain useful information. For example, DataRobot excludes features it determines are low information, such as a column containing all ones, duplicate columns, or a feature with too few values. The Informative Features list is sorted by each feature's correlation with the target variable
* Raw Features: All features (variables) in the dataset, including those excluded from the Informative Features list.
* Univariate Selection: Features that meet a certain threshold for non-linear correlation with the selected target. DataRobot calculates, for each entry in the Informative features list, the feature's individual relationship against the target.

Users also have the option to create user-defined feature transformations, which can then be included in a feature list for model exploration and to determine relative feature importance. Importance is measured using the information content of the variable; the calculation is done independently for each feature in the dataset. Features are then ranked on the Leaderboard from most to least important. This score represents a measure of predictive power using only that variable to predict the target. The score is measured using the project's accuracy metric that is defined by either the user (i.e., Gamma Deviance) or the default assigned by DataRobot.

5.6.2 Expert Judgement and Variable Selection

This section should include additional detail regarding the variable selection process and any expert judgment used during feature selection.

5.6.3 Final Model Variables

Below are two tables. The first contains a list of the final set of model feature variables, as well as summary statistics for the eXtreme Gradient Boosted Trees Regressor with Early Stopping (Poisson Loss) model and the second table contains a detailed analysis of missing values.

The Model Features and Summary Statistics table provides a brief overview of the summary statistics of model features. This includes Feature Name, variable type (Var Type), number of unique values (Unique), Number of missing values (Missing), Mean, Standard Deviation (Std Dev), Median, Minimum Value (Min), Maximum Value (Max) and Assessment of target leakage risk (Target Leakage).

5.6.3.1 Model Features and Summary Statistics

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature Name | Var Type | Unique | Missing | Mean | Std Dev | Median | Min | Max | Target Leakage |
| Id | Numeric | 1168 | 0 | 730.47 | 420.6 | 733.5 | 2.0 | 1460.0 | Low |
| MSSubClass | Numeric | 15 | 0 | 57.21 | 42.73 | 50.0 | 20.0 | 190.0 | Low |
| MSZoning | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| LotFrontage | Numeric | 107 | 201 | 70.14 | 25.073 | 69.0 | 21.0 | 313.0 | Low |
| LotArea | Numeric | 882 | 0 | 10492.054 | 9133.26 | 9522.5 | 1477.0 | 164660.0 | Low |
| Street | Categorical | 2 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Alley | Categorical | 2 | 1094 | N/A | N/A | N/A | N/A | N/A | Low |
| LotShape | Categorical | 4 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| LandContour | Categorical | 4 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| LotConfig | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| LandSlope | Categorical | 3 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Neighborhood | Categorical | 25 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Condition1 | Categorical | 9 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Condition2 | Categorical | 7 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| BldgType | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| HouseStyle | Categorical | 8 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| OverallQual | Numeric | 10 | 0 | 6.098 | 1.38 | 6.0 | 1.0 | 10.0 | Low |
| OverallCond | Numeric | 8 | 0 | 5.58 | 1.13 | 5.0 | 2.0 | 9.0 | Low |
| YearBuilt | Numeric | 110 | 0 | 1971.36 | 30.19 | 1973.0 | 1872.0 | 2010.0 | Low |
| YearRemodAdd | Numeric | 61 | 0 | 1984.92 | 20.54 | 1993.0 | 1950.0 | 2010.0 | Low |
| RoofStyle | Categorical | 6 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| RoofMatl | Categorical | 8 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Exterior1st | Categorical | 15 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Exterior2nd | Categorical | 16 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| MasVnrType | Categorical | 3 | 695 | N/A | N/A | N/A | N/A | N/A | Low |
| MasVnrArea | Numeric | 294 | 6 | 106.057 | 186.077 | 0.0 | 0.0 | 1600.0 | Low |
| ExterQual | Categorical | 4 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| ExterCond | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Foundation | Categorical | 6 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtQual | Categorical | 4 | 28 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtCond | Categorical | 4 | 28 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtExposure | Categorical | 4 | 29 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtFinType1 | Categorical | 6 | 28 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtFinSF1 | Numeric | 556 | 0 | 451.29 | 466.19 | 386.5 | 0.0 | 5644.0 | Low |
| BsmtFinType2 | Categorical | 6 | 29 | N/A | N/A | N/A | N/A | N/A | Low |
| BsmtFinSF2 | Numeric | 121 | 0 | 46.077 | 158.78 | 0.0 | 0.0 | 1474.0 | Low |
| BsmtUnfSF | Numeric | 675 | 0 | 568.52 | 446.83 | 478.0 | 0.0 | 2336.0 | Low |
| TotalBsmtSF | Numeric | 626 | 0 | 1065.88 | 444.79 | 992.0 | 0.0 | 6110.0 | Low |
| Heating | Categorical | 6 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| HeatingQC | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| CentralAir | Categorical | 2 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Electrical | Categorical | 5 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| 1stFlrSF | Numeric | 655 | 0 | 1167.21 | 391.207 | 1088.0 | 334.0 | 4692.0 | Low |
| 2ndFlrSF | Numeric | 354 | 0 | 346.87 | 435.901 | 0.0 | 0.0 | 2065.0 | Low |
| LowQualFinSF | Numeric | 19 | 0 | 5.74 | 48.3005 | 0.0 | 0.0 | 572.0 | Low |
| GrLivArea | Numeric | 728 | 0 | 1519.83 | 533.18 | 1468.5 | 334.0 | 5642.0 | Low |
| BsmtFullBath | Numeric | 4 | 0 | 0.42 | 0.51 | 0.0 | 0.0 | 3.0 | Low |
| BsmtHalfBath | Numeric | 3 | 0 | 0.057 | 0.24 | 0.0 | 0.0 | 2.0 | Low |
| FullBath | Numeric | 4 | 0 | 1.57 | 0.55 | 2.0 | 0.0 | 3.0 | Low |
| HalfBath | Numeric | 3 | 0 | 0.38 | 0.5004 | 0.0 | 0.0 | 2.0 | Low |
| BedroomAbvGr | Numeric | 8 | 0 | 2.87 | 0.81 | 3.0 | 0.0 | 8.0 | Low |
| KitchenAbvGr | Numeric | 4 | 0 | 1.046 | 0.22 | 1.0 | 0.0 | 3.0 | Low |
| KitchenQual | Categorical | 4 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| TotRmsAbvGrd | Numeric | 12 | 0 | 6.54 | 1.63 | 6.0 | 2.0 | 14.0 | Low |
| Functional | Categorical | 7 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| Fireplaces | Numeric | 4 | 0 | 0.607 | 0.64 | 1.0 | 0.0 | 3.0 | Low |
| FireplaceQu | Categorical | 5 | 558 | N/A | N/A | N/A | N/A | N/A | Low |
| GarageType | Categorical | 6 | 59 | N/A | N/A | N/A | N/A | N/A | Low |
| GarageYrBlt | Numeric | 97 | 59 | 1978.91 | 24.403 | 1980.0 | 1900.0 | 2010.0 | Low |
| GarageFinish | Categorical | 3 | 59 | N/A | N/A | N/A | N/A | N/A | Low |
| GarageCars | Numeric | 5 | 0 | 1.78 | 0.74 | 2.0 | 0.0 | 4.0 | Low |
| GarageArea | Numeric | 397 | 0 | 479.078 | 216.48 | 480.0 | 0.0 | 1418.0 | Low |
| GarageQual | Categorical | 5 | 59 | N/A | N/A | N/A | N/A | N/A | Low |
| GarageCond | Categorical | 5 | 59 | N/A | N/A | N/A | N/A | N/A | Low |
| PavedDrive | Categorical | 3 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| WoodDeckSF | Numeric | 252 | 0 | 96.904 | 128.46 | 0.0 | 0.0 | 857.0 | Low |
| OpenPorchSF | Numeric | 183 | 0 | 47.46 | 66.78 | 25.0 | 0.0 | 547.0 | Low |
| EnclosedPorch | Numeric | 102 | 0 | 22.055 | 62.43 | 0.0 | 0.0 | 552.0 | Low |
| 3SsnPorch | Numeric | 18 | 0 | 3.93 | 31.69 | 0.0 | 0.0 | 508.0 | Low |
| ScreenPorch | Numeric | 66 | 0 | 15.606 | 57.36 | 0.0 | 0.0 | 480.0 | Low |
| PoolArea | Numeric | 6 | 0 | 2.32 | 35.64 | 0.0 | 0.0 | 648.0 | Low |
| PoolQC | Categorical | 3 | 1163 | N/A | N/A | N/A | N/A | N/A | Low |
| Fence | Categorical | 4 | 947 | N/A | N/A | N/A | N/A | N/A | Low |
| MiscFeature | Categorical | 4 | 1124 | N/A | N/A | N/A | N/A | N/A | Low |
| MiscVal | Numeric | 19 | 0 | 49.72 | 552.039 | 0.0 | 0.0 | 15500.0 | Low |
| MoSold | Numeric | 12 | 0 | 6.27 | 2.71 | 6.0 | 1.0 | 12.0 | Low |
| YrSold | Numeric | 5 | 0 | 2007.801 | 1.32 | 2008.0 | 2006.0 | 2010.0 | Low |
| SaleType | Categorical | 9 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| SaleCondition | Categorical | 6 | 0 | N/A | N/A | N/A | N/A | N/A | Low |
| SalePrice | Numeric | 575 | 0 | 181528.605 | 80675.22 | 162700.0 | 39300.0 | 755000.0 | N/A |

The last column in this table is an assessment of target leakage risk. DataRobot automatically tests for target leakage on a per- feature basis during the Autopilot process. Target leakage, sometimes called data leakage, occurs when a model is trained using a dataset that includes information that would not be available at the time of prediction. This can produce overly optimistic model performance results during training, given a feature will near-completely describe the target (e.g., the number of late payments on a loan as a predictor for loan default at loan application date.)

DataRobot tests for target leakage risk using Alternating Conditional Expectation (ACE) to measure the association between each feature and the target; the ACE score is normalized using the project optimization metric so that its value is in the range [0,1]. If above a certain threshold (see below), DataRobot will create a new feature list with those features flagged and possibly removed, and the user is notified by a banner in the user interface during modeling. Notably, because the definition of target leakage is directly tied with prediction time and not strength of association between a feature and the target, it's possible for DataRobot to not identify all sources of target leakage. Therefore, to reduce the risk for potential target leakage in the feature list, it's important to apply subject matter expertise.

The thresholds for target leakage risk are based on a normalized ACE score:

* High risk: > 0.975, flagged and removed
* Moderate risk: > 0.85, flagged but not removed
* Low risk: < 0.85, no action

The following table provides a summary of missing values. It includes the name of the feature, its type, a summary of the missing value count (both number of rows and as a percentage), and provides information on the type of imputation applied to the feature.

5.6.3.2 Data Quality Handling Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature Name | Var Type | Missing Count | Missing Percentage | Imputation Name | Imputation Description |
| PoolQC | Categorical | 1453 | 100 | Ordinal encoding of categorical variables | Imputed value: -2 |
| MiscFeature | Categorical | 1406 | 96 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Alley | Categorical | 1369 | 94 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Fence | Categorical | 1179 | 81 | Ordinal encoding of categorical variables | Imputed value: -2 |
| MasVnrType | Categorical | 872 | 60 | Ordinal encoding of categorical variables | Imputed value: -2 |
| FireplaceQu | Categorical | 690 | 47 | Ordinal encoding of categorical variables | Imputed value: -2 |
| LotFrontage | Numeric | 259 | 18 | Missing Values Imputed | Missing indicator treated as feature, Imputed value: 69 |
| GarageQual | Categorical | 81 | 6 | Ordinal encoding of categorical variables | Imputed value: -2 |
| GarageFinish | Categorical | 81 | 6 | Ordinal encoding of categorical variables | Imputed value: -2 |
| GarageCond | Categorical | 81 | 6 | Ordinal encoding of categorical variables | Imputed value: -2 |
| GarageYrBlt | Numeric | 81 | 6 | Missing Values Imputed | Missing indicator treated as feature, Imputed value: 1980 |
| GarageType | Categorical | 81 | 6 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtExposure | Categorical | 38 | 3 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtFinType2 | Categorical | 38 | 3 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtQual | Categorical | 37 | 3 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtCond | Categorical | 37 | 3 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtFinType1 | Categorical | 37 | 3 | Ordinal encoding of categorical variables | Imputed value: -2 |
| MasVnrArea | Numeric | 8 | 1 | Missing Values Imputed | Missing indicator treated as feature, Imputed value: 0 |
| Electrical | Categorical | 1 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| 3SsnPorch | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| LotConfig | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtUnfSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 479 |
| Exterior1st | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| 2ndFlrSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| HalfBath | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| MoSold | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 6 |
| OverallCond | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 5 |
| LotArea | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 9480 |
| HouseStyle | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| GrLivArea | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1464 |
| YearRemodAdd | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1994 |
| BsmtHalfBath | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| Foundation | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| ExterCond | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| HeatingQC | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| RoofStyle | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| MiscVal | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| CentralAir | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| OverallQual | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 6 |
| KitchenAbvGr | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1 |
| BsmtFinSF1 | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 384 |
| LotShape | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BsmtFullBath | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| TotalBsmtSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 992 |
| BsmtFinSF2 | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| TotRmsAbvGrd | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 6 |
| YrSold | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 2008 |
| PavedDrive | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| OpenPorchSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 25 |
| LandSlope | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BldgType | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| FullBath | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 2 |
| SaleType | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| ScreenPorch | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| Exterior2nd | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Fireplaces | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1 |
| Heating | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| RoofMatl | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| GarageCars | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 2 |
| WoodDeckSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| 1stFlrSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1088 |
| KitchenQual | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| PoolArea | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| Condition2 | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Condition1 | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| LandContour | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| MSSubClass | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 50 |
| Neighborhood | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| BedroomAbvGr | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 3 |
| SaleCondition | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| ExterQual | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Functional | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| Street | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| LowQualFinSF | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| GarageArea | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 480 |
| MSZoning | Categorical | 0 | 0 | Ordinal encoding of categorical variables | Imputed value: -2 |
| YearBuilt | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 1973 |
| EnclosedPorch | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 0 |
| Id | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 731 |

6 Model Performance and Stability

6.1 Model Validation Stability

To find patterns in a dataset from which it can make predictions, an algorithm must first learn from a historical example – typically from a historical dataset that contains the output variable you want to predict. However, if a model is trained too closely on its training data then it may be overfit. Overfitting is a modeling error that occurs when a model is too closely fit to training data and therefore performs poorly on out-of-sample data (data that was not used to train the model). Overfitting generally results in an overly complex model that explains idiosyncrasies and random noise in the training data, rather than the underlying trends that the model was intended to capture. To avoid overfitting, the best practice is to evaluate model performance on out-of-sample data. If the model performs very well on in-sample data, (the training data) but poorly on out-of-sample data, that may be an indication that the model is overfit.

DataRobot uses standard modeling techniques to validate model performance and ensure that overfitting does not occur. DataRobot used a robust model k-fold cross-validation framework to test the out-of-sample stability of a model's performance. In addition to the cross-validation partitioning, DataRobot uses a holdout sample to further test out-of-sample model performance and ensure the model is not overfit.

The following procedure was used during development to insure that overfitting did not occur:

* DataRobot set aside 20.0% of the training data as a holdout dataset. This dataset is used to verify that the final model performs well on data that has not been touched throughout the training process.
* For further model validation, the remainder of the data is divided into 5 cross validation partitions. To compensate for the overhead when working with large datasets, DataRobot first trains models on a smaller part of the data and uses only one cross-validation fold to evaluate model performance. Then, for the highest performing models, DataRobot increases the subset sizes. This results in only the best model being trained on the total cross-validation partition. For those models, DataRobot completes 5-fold cross-validation training and scoring. As a result, the mean score of complete model cross-validation is calculated across all folds. Those models that did not perform well will not have a cross-validation score. Instead, because they only had a "one-fold" validation, their score is reported in the Validation column.

The following figure summarizes the CV process used by DataRobot, where the blue denotes 80.0% of the data available for training, which is then divided into 5-folds for cross-validation and and red denotes the holdout sample.



DataRobot calculates the Cross Validation scores for each of the training data partitions or folds. The project metric used to calculate the score is Gamma Deviance.

6.1.1 Cross Validation Scores

|  |  |
| --- | --- |
| Fold | Cross Validation Score (Gamma Deviance) |
| Fold 1 | 0.0089 |
| Fold 2 | 0.01781 |
| Fold 3 | 0.01101 |
| Fold 4 | 0.01355 |
| Fold 5 | 0.01238 |

6.1.2 Data Partitioning Methodology

Data partitions were selected by means of random sampling.

6.2 Model Performance (Sample Scores)

As an additional layer of model validity, DataRobot not only evaluated the statistical metrics underlying the model, but also performed testing on in-sample records.

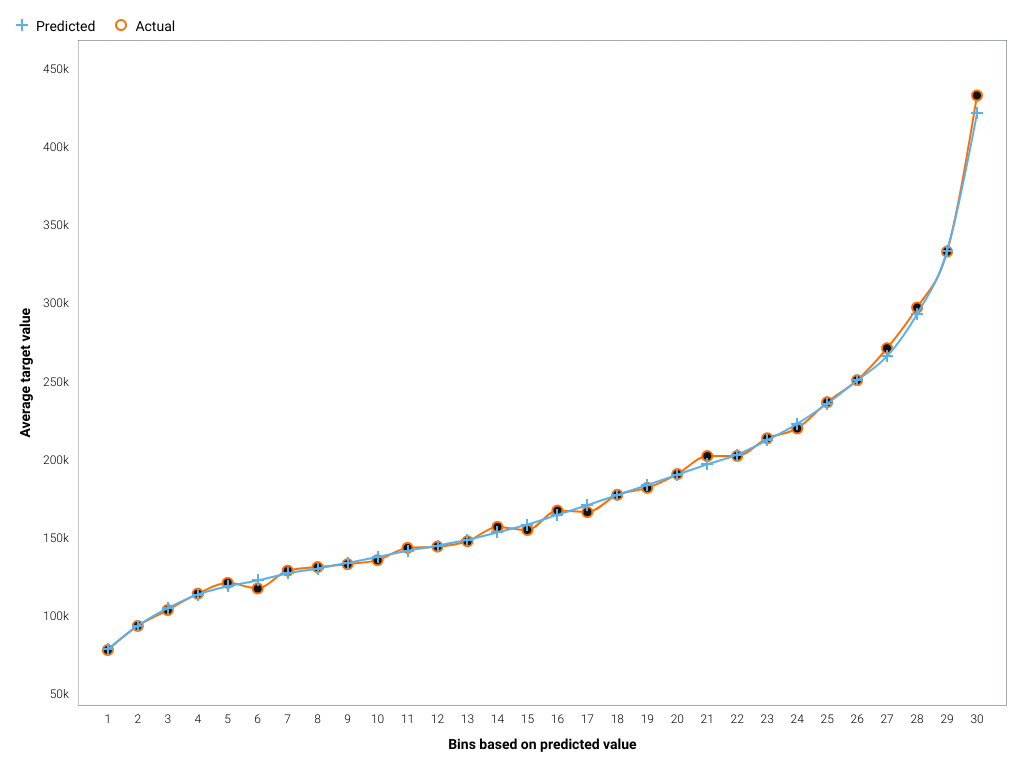
The performance metric used for this project was Gamma Deviance. The model performance results are presented below for in-sample testing:

|  |  |
| --- | --- |
| Scoring Type | Score (Gamma Deviance) |
| cross\_validation | 0.0127\* |
| holdout | 0.0169\* |
| validation | 0.0089\* |

6.3 Sensitivity Testing and Analysis

6.3.1 Lift Chart

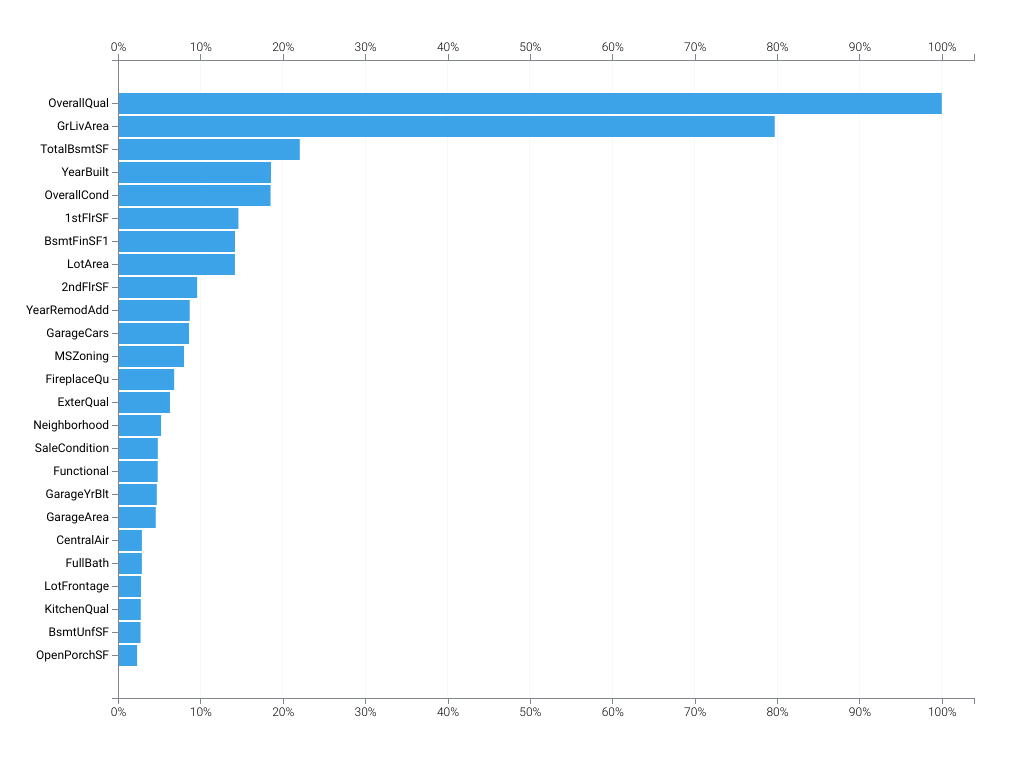
The Lift Chart sorts and groups numeric feature values into equal sized bins, depicting how well a model segments the target population and how capable it is of predicting the target, This helps the user to visualize model accuracy for each bin. The chart is sorted by predicted values -- lowest to highest predictions, for example -- which provides transparency to the model performance for different ranges of values of the target variable. Looking at the Lift Chart, the left side of the curve indicates where the model predicted a low score on one section of the population while the right side of the curve indicates where the model predicted a high score. The model Lift Chart is presented in the figure below.



The points on the Lift Chart indicate the average percentage in each bin. The "Predicted" blue line displays the average prediction score for the rows in that bin. The "Actual" orange line displays the actual percentage for the rows in that bin. In general, the steeper the Actual line is, and the more closely the Predicted line matches the actual line, the better the model. A close relationship between these two lines is indicative of the predictive accuracy of the model; a consistently increasing line is another good indicator of satisfactory model performance.

6.3.2 Key Relationships

Feature Impact, which is available for all model types, works by altering input data and observing the effect on a models score. This technique is sometimes called Permutation Importance. The Feature Impact for a given column measures how much worse a models error score would be if DataRobot made predictions after randomly shuffling that column (while leaving other columns unchanged). DataRobot normalizes the scores so that the value of the most important feature column is first and the other subsequent features are normalized to it.

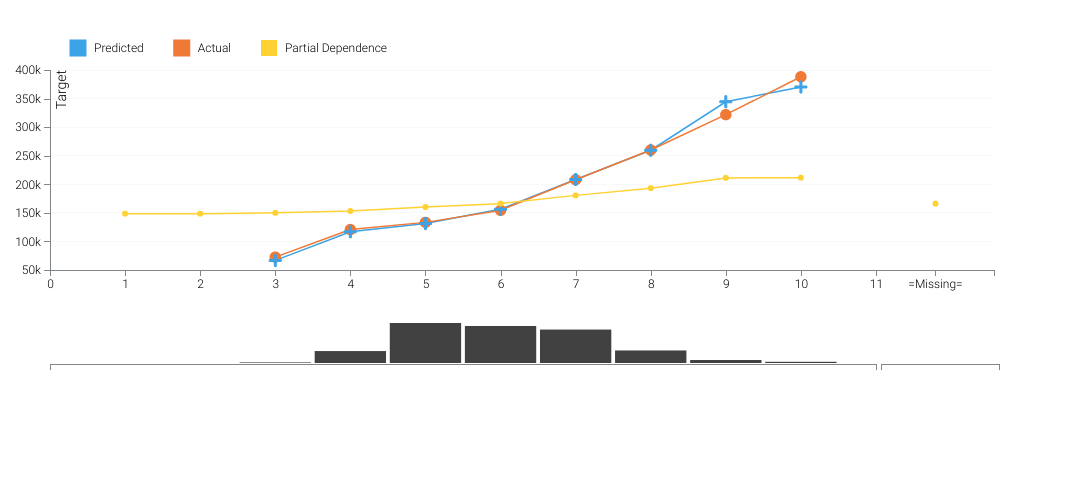


|  |  |  |
| --- | --- | --- |
| Feature Name | Impact Normalized | Impact Unnormalized |
| OverallQual | 1.0 | 0.0152 |
| GrLivArea | 0.7972 | 0.0121 |
| TotalBsmtSF | 0.2207 | 0.0034 |
| YearBuilt | 0.1857 | 0.0028 |
| OverallCond | 0.1852 | 0.0028 |
| 1stFlrSF | 0.1462 | 0.0022 |
| BsmtFinSF1 | 0.142 | 0.0022 |
| LotArea | 0.1419 | 0.0022 |
| 2ndFlrSF | 0.0961 | 0.0015 |
| YearRemodAdd | 0.087 | 0.0013 |
| GarageCars | 0.0862 | 0.0013 |
| MSZoning | 0.0802 | 0.0012 |
| FireplaceQu | 0.0682 | 0.001 |
| ExterQual | 0.0631 | 0.001 |
| Neighborhood | 0.0523 | 0.0008 |
| SaleCondition | 0.0483 | 0.0007 |
| Functional | 0.0482 | 0.0007 |
| GarageYrBlt | 0.0471 | 0.0007 |
| GarageArea | 0.0458 | 0.0007 |
| CentralAir | 0.029 | 0.0004 |
| FullBath | 0.029 | 0.0004 |
| LotFrontage | 0.0279 | 0.0004 |
| KitchenQual | 0.0276 | 0.0004 |
| BsmtUnfSF | 0.0274 | 0.0004 |
| OpenPorchSF | 0.0232 | 0.0004 |
| Condition1 | 0.023 | 0.0003 |
| Id | 0.0206 | 0.0003 |
| GarageType | 0.0185 | 0.0003 |
| ScreenPorch | 0.0166 | 0.0003 |
| Exterior1st | 0.0149 | 0.0002 |
| MoSold | 0.014 | 0.0002 |
| HalfBath | 0.0135 | 0.0002 |
| WoodDeckSF | 0.0134 | 0.0002 |
| BsmtExposure | 0.0116 | 0.0002 |
| Exterior2nd | 0.0112 | 0.0002 |
| BsmtFullBath | 0.0111 | 0.0002 |
| TotRmsAbvGrd | 0.0097 | 0.0001 |
| MasVnrArea | 0.0094 | 0.0001 |
| EnclosedPorch | 0.0092 | 0.0001 |
| Fireplaces | 0.009 | 0.0001 |
| BsmtFinType1 | 0.0082 | 0.0001 |
| YrSold | 0.0082 | 0.0001 |
| MSSubClass | 0.0079 | 0.0001 |
| HouseStyle | 0.0066 | 0.0001 |
| Foundation | 0.0064 | 0.0001 |
| SaleType | 0.0056 | 0.0001 |
| BsmtFinSF2 | 0.0055 | 0.0001 |
| HeatingQC | 0.0054 | 0.0001 |
| PavedDrive | 0.0054 | 0.0001 |
| BedroomAbvGr | 0.0054 | 0.0001 |
| GarageQual | 0.0052 | 0.0001 |
| KitchenAbvGr | 0.0049 | 0.0001 |
| BsmtCond | 0.0049 | 0.0001 |
| LotConfig | 0.0042 | 0.0001 |
| ExterCond | 0.0042 | 0.0001 |
| BsmtQual | 0.0042 | 0.0001 |
| GarageCond | 0.0042 | 0.0001 |
| LandContour | 0.0035 | 0.0001 |
| GarageFinish | 0.0029 | 0.0 |
| Electrical | 0.0028 | 0.0 |
| RoofStyle | 0.0027 | 0.0 |
| LotShape | 0.0026 | 0.0 |
| Heating | 0.0023 | 0.0 |
| LandSlope | 0.0019 | 0.0 |
| BldgType | 0.0015 | 0.0 |
| 3SsnPorch | 0.0015 | 0.0 |
| PoolArea | 0.0014 | 0.0 |
| MasVnrType | 0.0014 | 0.0 |
| Alley | 0.0014 | 0.0 |
| BsmtHalfBath | 0.0011 | 0.0 |
| LowQualFinSF | 0.0009 | 0.0 |
| MiscFeature | 0.0007 | 0.0 |
| Condition2 | 0.0007 | 0.0 |
| PoolQC | 0.0007 | 0.0 |
| Fence | 0.0005 | 0.0 |
| RoofMatl | 0.0005 | 0.0 |
| MiscVal | 0.0004 | 0.0 |
| BsmtFinType2 | 0.0003 | 0.0 |
| Street | 0.0002 | 0.0 |

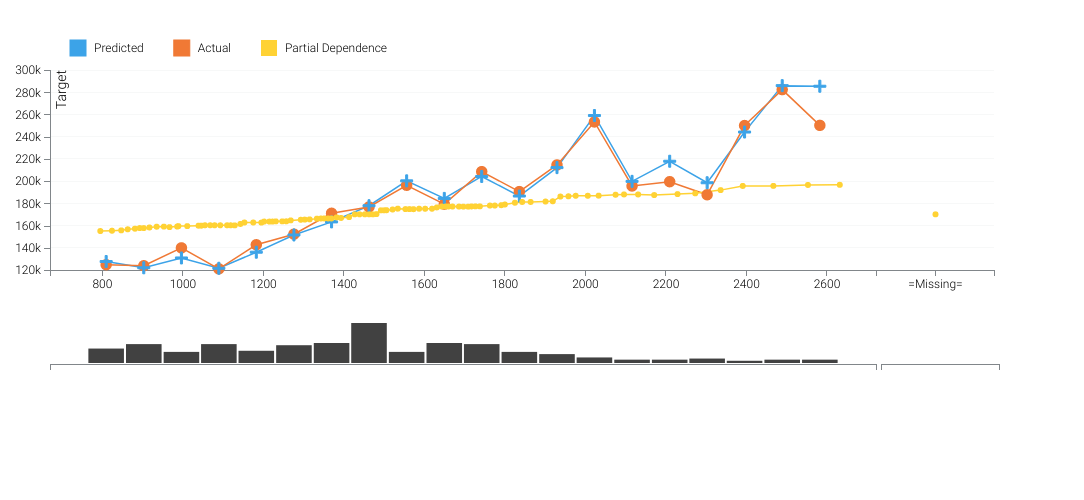
6.3.3 Sensitivity Analysis (Partial Dependence)

In the case of linear regression, we can gain considerable insight into the structure and interpretation of the model by examining its coefficients. For more complex models like support vector machines, random forests, or the blenders considered here, no comparably simple parametric description is available, making the interpretation of these models more difficult. To address this difficulty for his gradient boosting machine, Friedman (2001) proposed the use of partial dependence plots. Partial dependence plots show the average partial relationship between a set of predictors and the predicted response. The partial dependence plots below capture the top features in our model, as measured by Feature Impact.

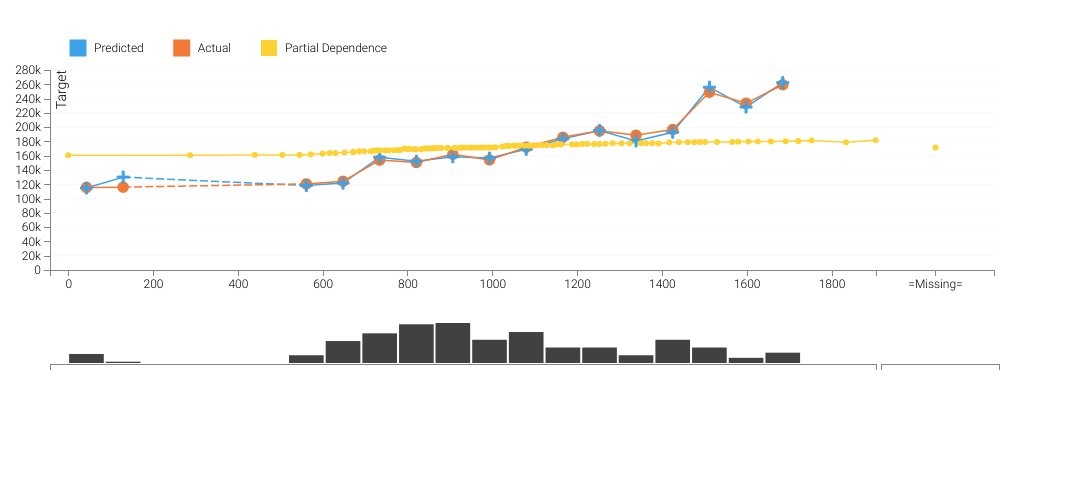
OverallQual



GrLivArea



TotalBsmtSF



The orange circles depict, for the selected feature, the average target value for the aggregated feature values. The blue crosses depict, for the selected feature, the average prediction for a specific value. From the graph you can see that DataRobot also averages the predicted feature values. Comparing the actual and predicted points can identify segments where model predictions differ from observed data. This typically occurs when the segment size is small. In those cases, for example, some models may predict closer to the overall average.

The yellow partial dependence data points depict the marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables constant, the value of this feature affects your prediction. DataRobot holds constant the values of all columns in the sample except the feature of interest. The value of the feature of interest is then reassigned to each possible value, calculating the average predictions for the sample at each setting. These values help determine how the value of each feature affects the target. The shape of the yellow data points describes the model's view of the marginal relationship between the selected feature and the target.

7 Model Implementation and Output Reporting

7.1 Version Control

DataRobot handles model and project version control automatically by tagging each model on the Leaderboard with a unique Model ID. The Model ID represents a single instance of a model type, feature list, sample size, and set of tuning parameter values. DataRobot also maintains unique Project IDs for each project, allowing accessibility to all models built for the project dataset. DataRobot's version control allows for reproducibility and traceability of the models it creates, which greatly increases the auditability of the model development process.

Users may also export scoring code for a DataRobot model in Java. You can download both a pre-compiled .jar file (with all dependencies included), plus the source code. Scoring code is easy to deploy, test, and maintain on a variety of platforms, and you can inspect the generated Java and Python code for complete transparency. DataRobot Scoring Code employs advanced features to ensure that predictions computed using generated Java code are the same as predictions computed inside DataRobot.