1 How To Use This Document

This document outlines the steps taken and modeling results for the US Department of Justice Recidivism Challenge. Overall, DataRobot's AutoML solution was used with some basic preprocessing code written in R to arrive at the best model submission described herein. It should be noted that this type of modeling exercise would require great care if put into production. Ethical and responsible AI means fully understanding the stakeholder impact, system's limitations, documenting risks and most importantly risk-mitigation based on severity and likelihood to protect vulnerable populations. The model described herein is NOT intended for production based on the ethical considerations and impacts that could be amplified without careful discretion. The extent of the model's utility is to understand and provide insights based on the data provided specifically in terms of accuracy and its relationship to protected features, race and gender, according to proportional parity.

Table of Contents

2 DataRobot Model Development Documentation

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 purpose, and that it is conceptually sound. As previously mentioned, the model does not purport to be an ethical implementation for the life-impacting use case described in the contest description.

3 Executive Summary and Model Overview

3.1 Model Stakeholders

The model was created as part of a "Recidivism Challenge" presented by the US Department of Justice.

Model Owner(s): Model ownership is determined by the rules of the publicly held competition.

Model Developer(s): Ted Kwartler, VP Trusted AI & Harvard University Extension School Adjunct Professor solely created the model.

Model User(s): It is likely the model will be reviewed, and interesting insights will be extracted by Department of Justice technical personnel rather than a production model or application be developed.

Model Validator(s): Model validation was performed by challenge administrators with results shared online.

3.2 Model Development Purpose and Intended Use

According to the contest website, The National Institute of Justice's (NIJ) "Recidivism Forecasting Challenge" (the Challenge) aims to increase public safety and improve the fair administration of justice across the United States. Results from the Challenge will provide critical information to community corrections departments that may help facilitate more successful reintegration into society for previously incarcerated persons and persons on parole.

Thus, it is likely not only the predictive power is of interest but also any insights stemming from the analysis may be of use in the academic-style research being conducted on recidivism to improve outcomes for affected people.

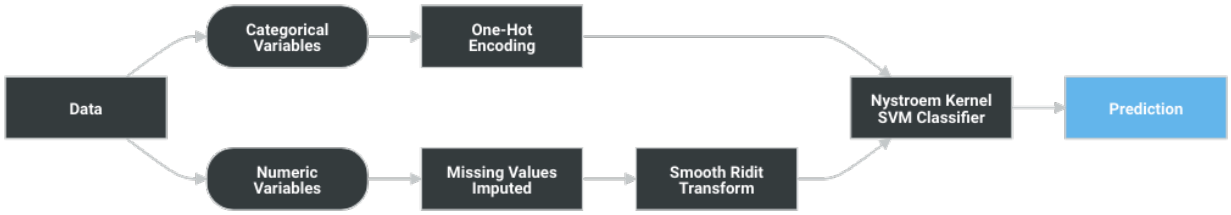
Data was prepared with ~50 lines of code written in R before being presented to an automated machine learning system. Within the automated machine learning application, 74 machine learning model types and accompanying tuning parameters were fit. These models were sorted according to LogLoss. The model described in this document represents the best model according to LogLoss among the 74, without any advanced feature engineering or enrichment which could have improved results but at the expense of time and effort. Additionally, an advanced performance indicator was selected to understand Bias and Fairness within the model. As a result, insights and results for race and gender according to proportional parity are also provided alongside standardized model KPIs.

Since the model is not intended to be productionalized, high risk could occur if the model is used to inform decision making.

3.3 Model Description and Overview

The particular model referenced in this document: Nystroem Kernel SVM Classifier. This model was developed in a project created with vf0e8f0f7a9b51308 of DataRobot. This model is denoted within DataRobot by the Project ID: 60b404c9d14c853b8cb15aee and the Model ID: 60b4ffaaeceb345397474965. The project was created on 2021-05-30 21:34:01.

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:

- One-Hot Encoding
- Missing Values Imputed
- Smooth Ridit Transform
- Nystroem Kernel SVM Classifier

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 LogLoss and the project included a total of 18,028 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 (LogLoss)
cross_validation	0.5558*
holdout	0.5444*
validation	0.5521*

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. *As constructed, there is no prediction stacking and subsequent "downstream" models. Prediction stacking was explored where sophisticated models impute missing values in the training data but the interdependent complexities weighed against the benefit made the exercise ineffective.*

4 Model Data Overview

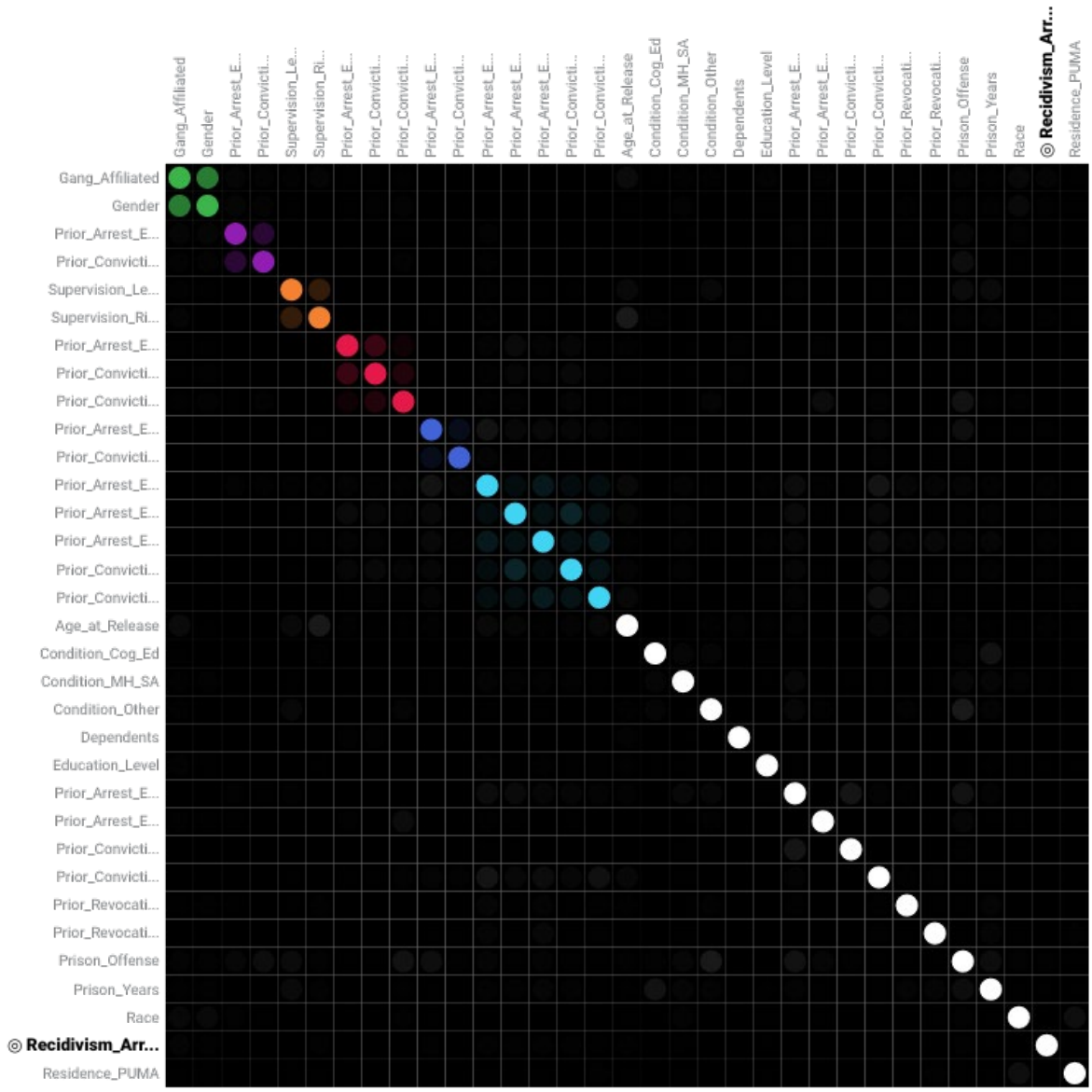
4.1 Feature Association

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.

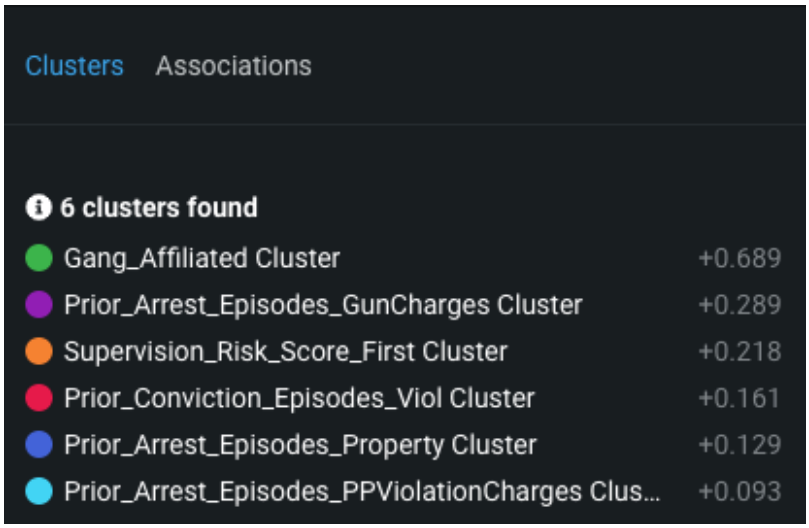
The Feature Association Matrix lists up to the top 50 features, selected by Importance Score, on both the X and Y axes, where the intersection of a feature pair provides an indication of their level of association. By default, the matrix displays by the Mutual Information values and sorts by the cluster.



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.

- Identified clusters include:



4.2 Data Source Overview and Appropriateness

Subject matter expertise has been determined by the contest administrators. The model was built with the assumption the data is correct, error-free and appropriate for the modeling task. It is likely the case, as was mentioned in the contest explanation that feature enrichment not just feature engineering could improve results. However, with anonymized data, enrichment is difficult or could encourage inadvertent bias by proxy.

The data was obtained from the contest website through webpage download. For example here,

https://data.ojp.usdoj.gov/stories/s/daxx-hznc

https://data.ojp.usdoj.gov/Corrections/NIJ-s-Recidivism-Challenge-Training-Dataset/8tjc-3ibv

---

4.3 Input Data Extraction, Preparation, and Quality & Completeness

The following R code represents the basic preprocessing of the raw data with comments included.

```
# Set the working directory
setwd("~/Desktop/nij")

# Identify and read in contest files.
train    <- list.files(pattern = 'Training')
test     <- list.files(pattern = 'Test')
```

```
testData <- read.csv(test )
trainingData <- read.csv(train )

# Intersect the training data columns with test set
keeps <- names(trainingData) %in% names(testData)
NIJ_keeps <- trainingData[,keeps]
NIJ_keeps$Recidivism_Arrest_Year1 <- trainingData$Recidivism_Arrest_Year1
# this example code is for round 1,
# but changes for other rounds `Recidivism_Arrest_Year2`
write.csv(NIJ_keeps,'NIJ_keeps_phase1.csv' ,row.names = F)
write.csv(testData, 'NIJ_test_phase1.csv', row.names = F)
```

4.4 Data Assumptions

Data is assumed to be correct for the task description. Data is assumed to be acceptable for privacy concerns and representative of the population under study. It is not possible to know data specifics, practitioner assumptions during munging or implications of data collection since it is provided as part of the contest.

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

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:

- One-Hot Encoding
- Missing Values Imputed
- Smooth Ridit Transform
- Nystroem Kernel SVM Classifier

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

5.3.1 One Hot encoding task

This transformer will do a binary one-hot (aka one-of-K) coding. One boolean-valued feature is constructed for each of the possible string values that the feature can take. For inputs with only 2 unique values, only one boolean-valued feature will be constructed

This encoding is needed for feeding categorical data to many estimators, notably linear models and SVMs.

Type	Name	Description	Best Searched
int	card_max	An integer that specifies the maximum number of unique values values: [1, 99999]	50000
int	card_min	An integer that specifies the minimum number of unique values values: [1, 99999]	1
bool	drop_cols	drop_cols, If True, drop last level of each feature values: [False, True]	False
select	flag	flag, If all, add highcat-cols to metadata values: ['None', 'all']	None
int	max_features	If the total number of categories created across all features exceeds this value, the top max_features most frequent categories will persist. All others will be either thrown out or	None

		grouped. A value of None disables the limit. values: [1, 999999]	
int	min_support	The minimum number of records for a category to be represented in one hot encoding. If a category has fewer counts it will be grouped with other small cardinality values values: [1, 99999]	10

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]	False
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 Ridit transformer

For a numeric feature, transform it to a ridit score based on percentile rank. The percentile score will be further adjusted to an interval between -1 and 1. The transformer can be configured to skip binary feature and date/time derived features. If the sparsity is higher than the sparsity_threshold, data will be centered to the median and the output will be a sparse matrix.

The ridit transform is an extension of Bross' (1958) RIDIT scoring method, which suggests the use of Ridit analysis for data that are ordered but not on an interval scale, such as injury categories. Bross' (1958) RIDIT's procedure is as follows: from a reference population with the same categories (of injury, for example), determine a "ridit" or score for each category. This category score is the percentile rank of an item in the reference population and is equal to the number of items in all lower categories plus one-half the number of items in the subject category, all divided

by the population size. By definition, the mean of Bross ridit calculated for the reference population will always be 0.5.

The ridit transform extends the Bross ridit method by applying the method to numerical values and normalizing the score such that the mean calculated for the reference population will always be 0 and the score will be in the interval [-1,1].

Intuitively, the ridit transform can be interpreted to be an adjusted percentile score.

Ridit transform is not smooth and variable mapping is not continuous at the bin boundaries. DataRobot developed a "smooth" version of Ridit mapping, where the mapping inside of each bin is set to linearly increase to reach the other bin's starting value. The middle point in the bin corresponds to the value of the original Ridit algorithm, and the mean of the distribution is still equal to 0.5. By using such approach, the mapping is continuous and predictions are consistent for the data values close to the boundary values.

Type	Name	Description	Best Searched
bool	skip_bins	If True, ridit transform will skip binary columns. values: [False,True]	False
bool	skip_date_features	If True, ridit transform will skip extracted features from the date column. values: [False,True]	False
float	sparsity_threshold	If sparsity level is higher than the parameter, the matrix is converted to a sparse format. values: [0, 1]	0.25

5.3.4 Approximate kernel support vector classifier.

Support vector machines are a class of "maximum margin" classifiers. They seek to maximize the separation they find between classes, and can optionally include a penalty function that allows them to mis-classify some observations for the sake of wider margins between the classes for the rest of the observations. This makes support vector machine a very robust class of machine learning models.

SVMs are very efficient in high-dimensional spaces (such as text data), including cases where the number of dimensions exceeds the number of observations, but unfortunately do not tend to scale well to a large number of examples.

SVMs can also make use of a “kernel” function, which allows for a non- linear transformation of the data before fitting the SVM. These kernel functions can be a very useful way to transform a non-linear problem into a linear domain.

This implementation of SVM approximates the feature mappings. The advantage of using approximate explicit feature maps compared to the kernel trick, which makes use of feature maps implicitly, is that explicit mappings can significantly reduce the cost of learning with large datasets. Kernel map approximations allow SVMs to run much faster, and scale up to bigger datasets, but there is a small tradeoff in accuracy.

Usually, higher accuracy is obtained with higher n_components, as it sets the the number of samples used to construct the basis features and number of features to construct. However, large value for n_components leads to high memory usage.

This implementation combines kernel map approximations with a LogisticRegression that uses a l2 penalty and a liblinear solver.

Type	Name	Description	Best Searched
multi	C	Penalty parameter C of the error term. values: {'floatgrid': [1e-10, 1e10],'select': ['auto']}	59.9484250319
select	approx	The kernel approximation method to use. values: ['nystroem', 'fourier', 'balanced_nystroem']	balanced_nystroem
multi	gamma	The parameter of the RBF-kernel. ‘heuristic’ (For RBF only) from Caputo et al.[1], i.e. 1 / (the weighted median squared distance between two samples). ‘auto’ creates a grid around a base gamma for tuning with grid-search values: {'floatgrid': [1e-10, 1e10],'select': ['auto', 'heuristic']}	0.001075780054
multi	max_sample	The maximum sample size that can be used for training. values: {'int':	None

		[100, 100000],'select': ['None']}	
intgrid	n_components	Is the target dimensionality of the feature transform. values: [1, 10000]	500
int	random_state	The seed of the pseudo random number generator to use when shuffling the data. values: [1, int(1e9)]	1234
select	smart_sampling	if balanced negative downsampling should be done when max_sample is not None. values: [True, False]	True
floatgrid	subsample	The fraction of samples to be used for fitting. values: [0.01,1]	1.0
float	tol	Tolerance for stopping criteria. values: [1e-10, 1e10]	0.0001

5.4 Literature Review and References

- Suits, Daniel B. "Use of dummy variables in regression equations." Journal of the American Statistical Association 52.280 (1957): 548-551. http://www.jstor.org/stable/2281705?seq=1
- [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
- Bross, I. D. J. [1958]. "How to Use Ridit Analysis." Biornetrics- 14, pg. 18-38. https://doi.org/10.2307/2527727
- Brockett, Patrick L., and Arnold Levine. "On a characterization of ridits." The Annals of Statistics (1977): 1245-1248. https://projecteuclid.org/download/pdf_1/euclid.aos/1176344010
- Caputo, B., Sim, K., Furesjo, F., & Smola, A. "Appearance-based Object Recognition using SVMs: Which Kernel Should I Use?". In Proc of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision (2002).

- Suykens, Johan AK, and Joos Vandewalle. "Least squares support vector machine classifiers." Neural processing letters 9.3 (1999): 293-300. http://bioinformatics.oxfordjournals.org/content/16/10/906.full.pdf
- Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: A library for support vector machines." ACM Transactions on Intelligent Systems and Technology (TIST) 2.3 (2011): 27. http://ntucsu.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf
- Williams, Christopher, and Matthias Seeger. "Using the Nystrom method to speed up kernel machines." Proceedings of the 14th Annual Conference on Neural Information Processing Systems. No. EPFL-CONF-161322. 2001. http://infoscience.epfl.ch/record/161322/files/nystroem.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 LogLoss. The model types considered during the model selection process included the following models, which are sorted by the Validation score.

Model Name	Validation Score	Cross Validation Score	Sample Percentage
Light Gradient Boosting on ElasticNet Predictions	0.5549	0.5571	100.0
Vowpal Wabbit Classifier	0.5542	0.5579	63.9949
Vowpal Wabbit Stagewise Polynomial Classifier	0.5542	0.5578	63.9949
Vowpal Wabbit Low Rank Quadratic Classifier	0.5542	0.5578	63.9949
Regularized Logistic Regression (L2)	0.5544	0.557	63.9949

Stochastic Gradient Descent Classifier	0.5544	0.5573	63.9949
Light Gradient Boosted Trees Classifier with Early Stopping	0.5545	0.5578	63.9949
ENET Blender	0.5547	0.5568	63.9949
AVG Blender	0.5547	0.5568	63.9949
Advanced AVG Blender	0.5547	0.5568	63.9949
eXtreme Gradient Boosted Trees Classifier with Early Stopping - Forest (10x)	0.5548	0.5578	63.9949
eXtreme Gradient Boosted Trees Classifier with Early Stopping (learning rate =0.02)	0.5548	0.5586	63.9949
Elastic-Net Classifier (mixing alpha=0.5 / Binomial Deviance)	0.5548	0.5572	63.9949
Keras Slim Residual Neural Network Classifier using Training Schedule (1 Layer: 64 Units)	0.5549	N/A	63.9949
Elastic-Net Classifier (L1 / Binomial Deviance)	0.5549	N/A	63.9949
Elastic-Net Classifier (L2 / Binomial Deviance)	0.5549	0.557	63.9949
eXtreme Gradient Boosted Trees Classifier with Early Stopping	0.5549	N/A	63.9949
Keras Residual AutoInt Classifier using Training Schedule (3 Attention Layers with 2 Heads, 2 Layers: 100, 100 Units)	0.5552	N/A	63.9949
Logistic Regression	0.5554	N/A	63.9949

Keras Residual Neural Factorization Machine Classifier using Training Schedule (2 Layers: 100, 100 Units)	0.5555	N/A	63.9949
Elastic-Net Classifier (mixing alpha=0.5 / Binomial Deviance) with Unsupervised Learning Features	0.5555	0.5572	63.9949
Generalized Additive2 Model	0.5556	N/A	63.9949
Generalized Additive Model	0.556	N/A	63.9949
Keras Deep Self-Normalizing Residual Neural Network Classifier using Training Schedule (3 Layers: 256, 128, 64 Units)	0.5561	N/A	63.9949
Elastic-Net Classifier (L2 / Binomial Deviance) with Binned numeric features	0.5562	N/A	63.9949
eXtreme Gradient Boosted Trees Classifier with Early Stopping and Unsupervised Learning Features	0.5564	N/A	63.9949
Gradient Boosted Trees Classifier with Early Stopping	0.5569	N/A	63.9949
Gradient Boosted Trees Classifier	0.5572	N/A	63.9949
eXtreme Gradient Boosted Trees Classifier with Early Stopping (learning rate =0.01)	0.5579	N/A	63.9949
RuleFit Classifier	0.5581	N/A	63.9949

TensorFlow Deep Learning Classifier	0.5592	N/A	63.9949
Keras Slim Residual Neural Network Classifier using Adaptive Training Schedule (1 Layer: 64 Units)	0.5592	N/A	63.9949
Keras Residual Cross Network Classifier using Training Schedule (3 Cross Layers, 4 Layers: 100, 100, 100, 100 Units)	0.5595	N/A	63.9949
Dropout Additive Regression Trees Classifier (15 leaves)	0.5607	N/A	63.9949
RandomForest Classifier (Gini)	0.5611	N/A	63.9949
RandomForest Classifier (Entropy)	0.5613	N/A	63.9949
Eureqa Generalized Additive Model Classifier (10000 Generations)	0.562	N/A	63.9949
LightGBM Random Forest Classifier	0.5623	N/A	63.9949
RandomForest Classifier (Entropy) (Shallow)	0.5627	N/A	63.9949
ExtraTrees Classifier (Gini)	0.5647	N/A	63.9949
Gradient Boosted Greedy Trees Classifier with Early Stopping	0.5653	N/A	63.9949
Keras Deep Residual Neural Network Classifier using Training Schedule (3 Layers: 512, 64, 64 Units)	0.5658	N/A	63.9949

Eureqa Generalized Additive Model Classifier (1178 Generations)	0.5673	N/A	63.9949
Breiman and Cutler Random Forest Classifier	0.57	N/A	63.9949
Decision Tree Classifier (Gini)	0.5773	N/A	63.9949
Auto-tuned K-Nearest Neighbors Classifier (Euclidean Distance)	0.5786	N/A	63.9949
Support Vector Classifier (Linear Kernel)	0.5806	N/A	63.9949
Eureqa Classifier (Default Search: 3000 Generations)	0.5833	N/A	63.9949
Eureqa Classifier (Quick Search: 250 Generations)	0.5857	N/A	63.9949
Eureqa Generalized Additive Model Classifier (40 Generations)	0.5869	N/A	63.9949
Eureqa Classifier (Instant Search: 40 Generations)	0.5968	N/A	63.9949
Keras Deep Residual Neural Network Classifier using Training Schedule (2 Layers: 512, 512 Units)	0.6979	N/A	63.9949
Keras Wide Residual Neural Network Classifier using Training Schedule (1 Layer: 1536 Units)	0.7134	N/A	63.9949

Naive Bayes combiner classifier	0.5759	N/A	15.9973
Majority Class Classifier	0.6095	N/A	15.9973

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., LogLoss) or the default assigned by DataRobot.

5.6.2 Expert Judgement and Variable Selection

NA, no manual variable selection was performed, other than exploration of protected features inclusion or exclusion in the model outcomes. This was done for exploration rather than expertise.

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 Nystroem Kernel SVM Classifier 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
Gender	Categorical	2	0	N/A	N/A	N/A	N/A	N/A	Low
Race	Categorical	2	0	N/A	N/A	N/A	N/A	N/A	Low
Age_at_Release	Categorical	7	0	N/A	N/A	N/A	N/A	N/A	Low
Residence_PUMA	Numeric	25	0	12.31	7.13	12.0	1.0	25.0	Low
Gang_Affiliated	Categorical	2	1724	N/A	N/A	N/A	N/A	N/A	Low
Supervision_Risk_Score_First	Numeric	10	270	6.064	2.39	6.0	1.0	10.0	Low
Supervision_Level_First	Categorical	3	974	N/A	N/A	N/A	N/A	N/A	Low
Education_Level	Categorical	3	0	N/A	N/A	N/A	N/A	N/A	Low
Dependents	Numeric	3	4296	0.81	0.82	1.0	0.0	2.0	Low
Prison_Offense	Categorical	5	1840	N/A	N/A	N/A	N/A	N/A	Low
Prison_Years	Categorical	4	0	N/A	N/A	N/A	N/A	N/A	Low
Prior_Arrest_Episodes_Felony	Numeric	10	3464	4.41	2.405	4.0	0.0	9.0	Low
Prior_Arrest_Episodes_Misd	Numeric	6	4605	2.085	1.68	2.0	0.0	5.0	Low
Prior_Arrest_Episodes_Violent	Numeric	3	2174	0.66	0.74	0.0	0.0	2.0	Low
Prior_Arrest_Episodes_Property	Numeric	5	3286	1.39	1.305	1.0	0.0	4.0	Low

Prior_Arrest_Episodes_Drug	Numeric	5	1725	1.37	1.31	1.0	0.0	4.0	Low
Prior_Arrest_Episodes_PPViolationCharges	Numeric	5	3600	1.45	1.39	1.0	0.0	4.0	Low
Prior_Arrest_Episodes_DVCharges	Boolean	2	0	0.17	0.37	0.0	0.0	1.0	Low
Prior_Arrest_Episodes_GunCharges	Boolean	2	0	0.26	0.44	0.0	0.0	1.0	Low
Prior_Conviction_Episodes_Felony	Numeric	3	3923	0.83	0.78	1.0	0.0	2.0	Low
Prior_Conviction_Episodes_Misd	Numeric	4	3391	1.0701	1.067	1.0	0.0	3.0	Low
Prior_Conviction_Episodes_Viol	Boolean	2	0	0.32	0.47	0.0	0.0	1.0	Low
Prior_Conviction_Episodes_Prop	Numeric	3	3033	0.61	0.75	0.0	0.0	2.0	Low
Prior_Conviction_Episodes_Drug	Numeric	2	3804	0.34	0.47	0.0	0.0	1.0	Low
Prior_Conviction_Episodes_PPViolationCharges	Boolean	2	0	0.33	0.47	0.0	0.0	1.0	Low
Prior_Conviction_Episodes_DomesticViolenceCharges	Boolean	2	0	0.079	0.27	0.0	0.0	1.0	Low
Prior_Conviction_Episodes_GunCharges	Boolean	2	0	0.13	0.34	0.0	0.0	1.0	Low
Prior_Revocations_Parole	Boolean	2	0	0.094	0.29	0.0	0.0	1.0	Low
Prior_Revocations_Probation	Boolean	2	0	0.15	0.35	0.0	0.0	1.0	Low
Condition_MH_SA	Boolean	2	0	0.66	0.47	1.0	0.0	1.0	Low
Condition_Cog_Ed	Boolean	2	0	0.44	0.5	0.0	0.0	1.0	Low
Condition_Other	Boolean	2	0	0.32	0.47	0.0	0.0	1.0	Low
Recidivism_Arrest_Year1	Boolean	2	0	0.3	0.46	0.0	0.0	1.0	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
Prior_Arrest_Episodes_Misd	Numeric	5733	32	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 2
Dependents	Numeric	5437	30	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 1
Prior_Conviction_Episodes_Felony	Numeric	4887	27	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 1
Prior_Conviction_Episodes_Drug	Numeric	4688	26	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 0
Prior_Arrest_Episodes_PPViolationCharges	Numeric	4465	25	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 1

Prior_Arrest_Episodes_Felony	Numeric	4307	24	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 4
Prior_Conviction_Episodes_Misd	Numeric	4219	23	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 1
Prior_Arrest_Episodes_Property	Numeric	4088	23	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 1
Prior_Conviction_Episodes_Prop	Numeric	3799	21	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 0
Prior_Arrest_Episodes_Violent	Numeric	2737	15	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 0
Prison_Offense	Categorical	2321	13	One-Hot Encoding	Missing indicator treated as feature
Gang_Affiliated	Categorical	2217	12	One-Hot Encoding	Missing indicator treated as feature
Prior_Arrest_Episodes_Drug	Numeric	2110	12	Missing Values Imputed	Missing indicator treated as feature,

					Imputed value: 1
Supervision_Level_First	Categorical	1212	7	One-Hot Encoding	Missing indicator treated as feature
Supervision_Risk_Score_First	Numeric	330	2	Missing Values Imputed	Missing indicator treated as feature, Imputed value: 6
Gender	Categorical	0	0	One-Hot Encoding	Missing values ignored
Prior_Arrest_Episodes_DVCharges	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Race	Categorical	0	0	One-Hot Encoding	Missing values ignored
Education_Level	Categorical	0	0	One-Hot Encoding	Missing values ignored
Condition_Other	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prior_Conviction_Episodes_DomesticViolenceCharges	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Residence_PUMA	Numeric	0	0	Missing Values Imputed	Imputed value: 12
Prior_Revocations_Parole	Numeric	0	0	Missing Values Imputed	Imputed value: 0

Prior_Conviction_Episodes_PPViolationCharges	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prior_Arrest_Episodes_GunCharges	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prior_Conviction_Episodes_Viol	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prior_Conviction_Episodes_GunCharges	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Condition_Cog_Ed	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prior_Revocations_Probation	Numeric	0	0	Missing Values Imputed	Imputed value: 0
Prison_Years	Categorical	0	0	One-Hot Encoding	Missing values ignored
Condition_MH_SA	Numeric	0	0	Missing Values Imputed	Imputed value: 1
Age_at_Release	Categorical	0	0	One-Hot Encoding	Missing values ignored

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

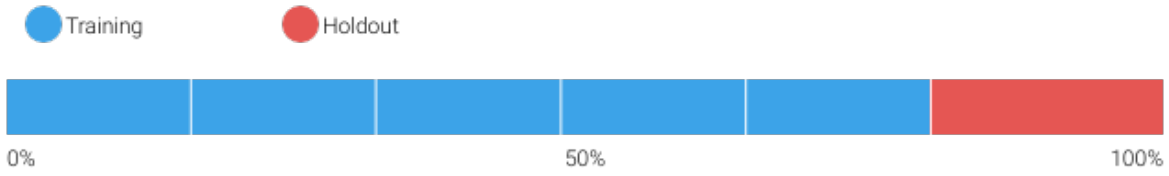
noise in the training data, rather than the underlying trends that the model was intended to capture. To avoid overfitting, the best practice is to evaluate model performance on out-of-sample data. If the model performs very well on in-sample data, (the training data) but poorly on out-of-sample data, 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 20.00222% 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 79.99778% 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 LogLoss.

6.1.1 Cross Validation Scores

Fold	Cross Validation Score (LogLoss)
Fold 1	0.55214

Fold 2	0.56329
Fold 3	0.54673
Fold 4	0.56017
Fold 5	0.55674

6.1.2 Data Partitioning Methodology

Because the distribution of the target in a binary classification project may be imbalanced, the modeling partitions were randomly selected using a stratified sample to preserve the distribution of the target for each partition.

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.

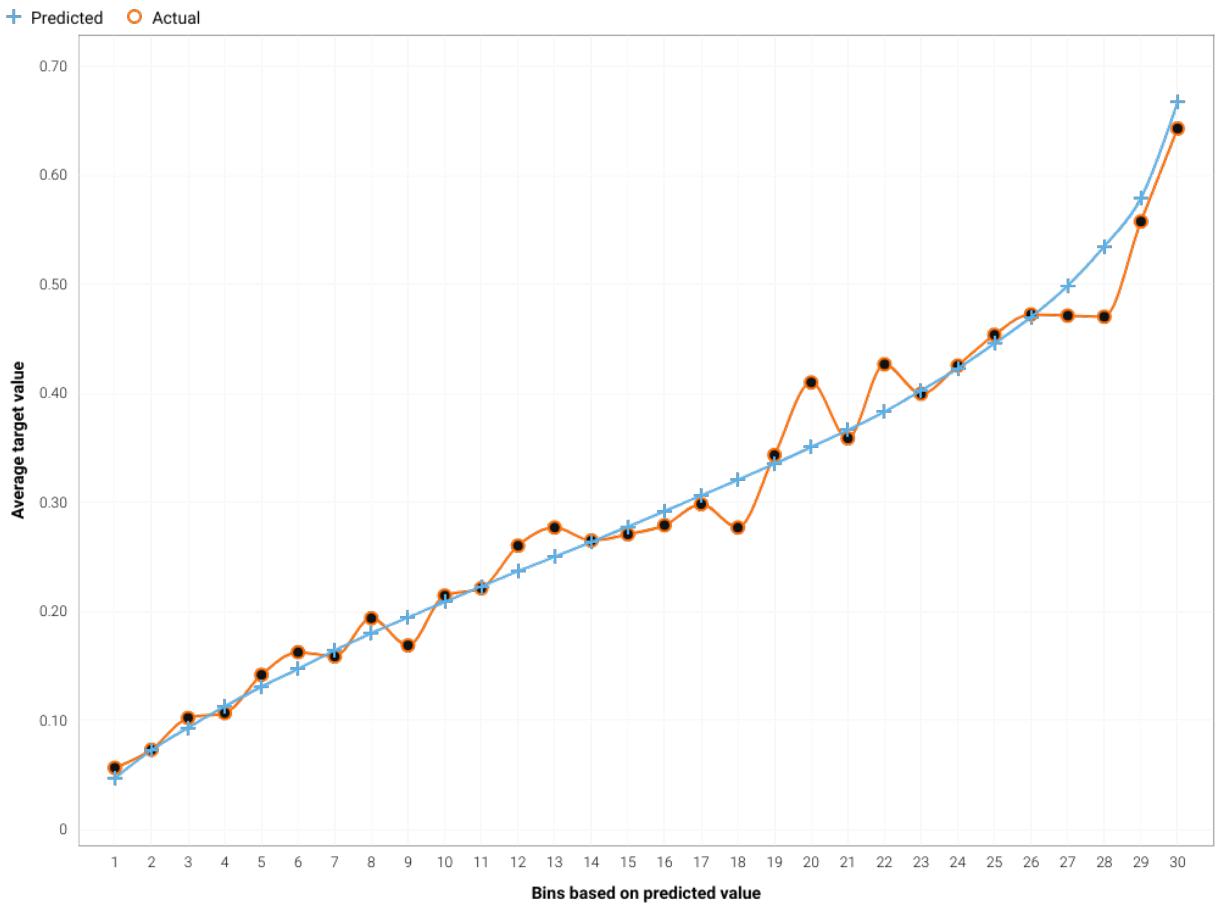
The performance metric used for this project was LogLoss. The model performance results are presented below for in-sample testing:

Scoring Type	Score (LogLoss)
cross_validation	0.5558*
holdout	0.5444*
validation	0.5521*

6.3 Sensitivity Testing and Analysis

6.3.1 Lift Chart

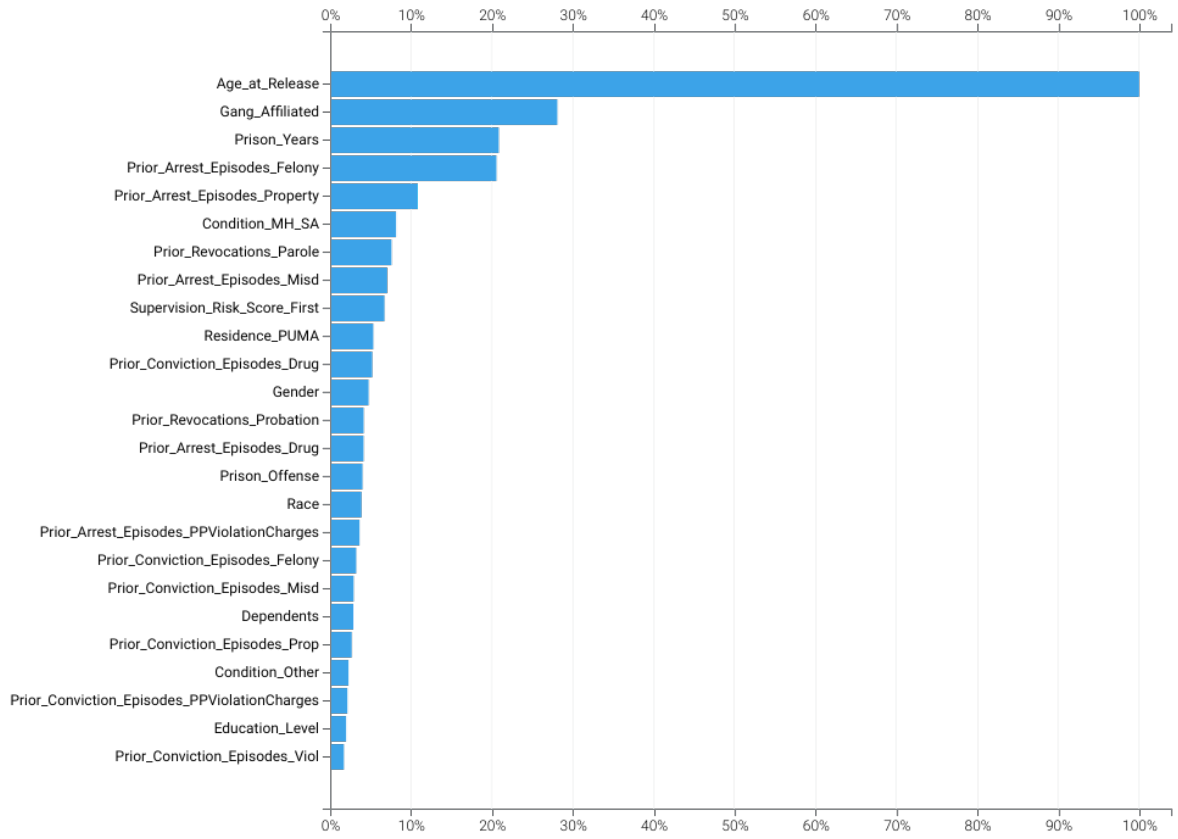
The Lift Chart sorts and groups numeric feature values into equal sized bins, depicting how well a model segments the target population and how capable it is of predicting the target, This helps the user to visualize model accuracy for each bin. The chart is sorted by predicted values -- lowest to highest 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.



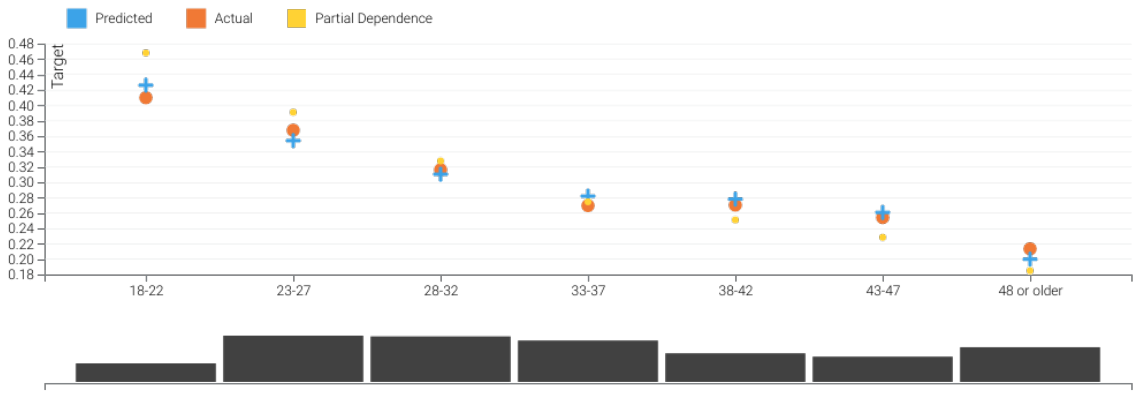
Feature Name	Impact Normalized	Impact Unnormalized
Age_at_Release	1.0	0.0349
Gang_Affiliated	0.2808	0.0098
Prison_Years	0.2085	0.0073
Prior_Arrest_Episodes_Felony	0.2056	0.0072
Prior_Arrest_Episodes_Property	0.1085	0.0038
Condition_MH_SA	0.0815	0.0028
Prior_Revocations_Parole	0.0758	0.0026
Prior_Arrest_Episodes_Misd	0.0707	0.0025
Supervision_Risk_Score_First	0.0672	0.0023
Residence_PUMA	0.0533	0.0019
Prior_Conviction_Episodes_Drug	0.0521	0.0018
Gender	0.0476	0.0017
Prior_Revocations_Probation	0.0415	0.0014
Prior_Arrest_Episodes_Drug	0.0413	0.0014

Prison_Offense	0.04	0.0014
Race	0.0388	0.0014
Prior_Arrest_Episodes_PPViolationCharges	0.0363	0.0013
Prior_Conviction_Episodes_Felony	0.0322	0.0011
Prior_Conviction_Episodes_Misd	0.0291	0.001
Dependents	0.0286	0.001
Prior_Conviction_Episodes_Prop	0.0267	0.0009
Condition_Other	0.0227	0.0008
Prior_Conviction_Episodes_PPViolationCharges	0.0213	0.0007
Education_Level	0.0197	0.0007
Prior_Conviction_Episodes_Viol	0.017	0.0006
Prior_Arrest_Episodes_DVCharges	0.0113	0.0004
Prior_Conviction_Episodes_GunCharges	0.0113	0.0004
Prior_Arrest_Episodes_GunCharges	0.0106	0.0004
Prior_Arrest_Episodes_Violent	0.007	0.0002
Prior_Conviction_Episodes_DomesticViolenceCharges	0.0049	0.0002
Supervision_Level_First	0.0015	0.0001
Condition_Cog_Ed	0.0015	0.0001

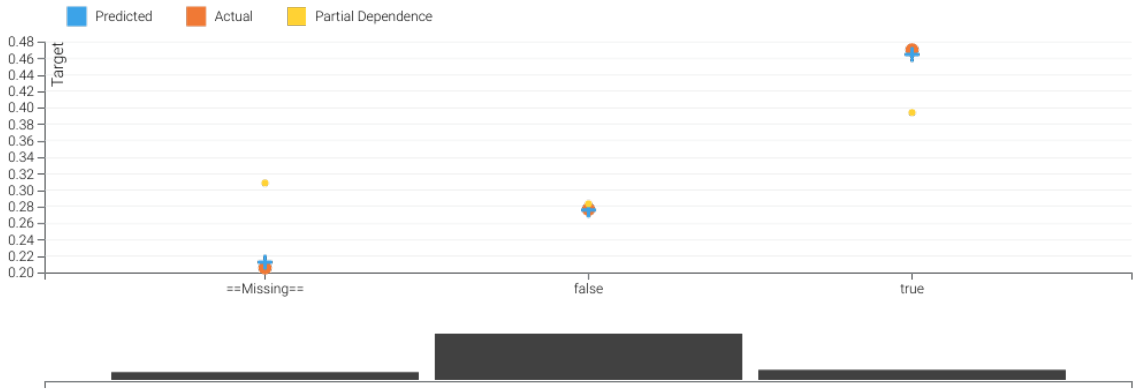
6.3.3 Sensitivity Analysis (Partial Dependence)

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.

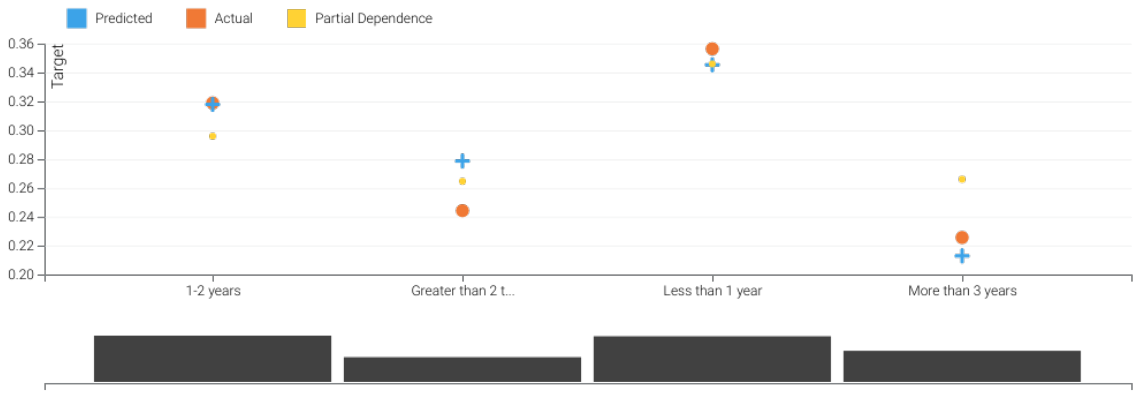
Age_at_Release



Gang_Affiliated



Prison_Years



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.

6.3.4 Accuracy (Receiver Operating Characteristic)

A confusion matrix is a table that reports true versus predicted values. The name "confusion matrix" refers to the fact that the matrix makes it easy to see if the model is confusing two classes (consistently mislabeling one class as another class). The table below presents key sensitivities that support the creation of a confusion matrix.

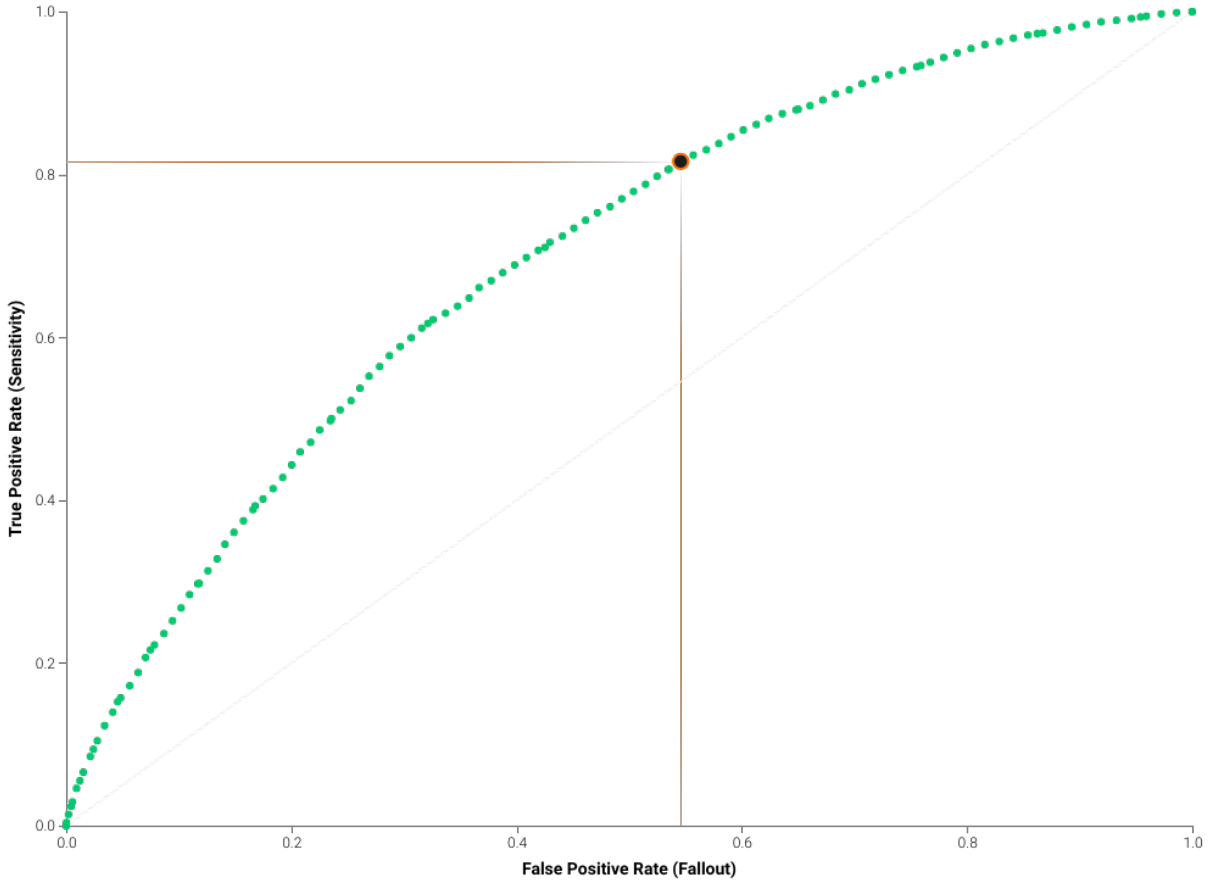
F1 Score	True Positive Rate	False Positive Rate	True Negative Rate	Positive Predictive Value	Negative Predictive Value	Accuracy	Matthews Correlation Coefficient
0.5265	0.8161	0.5457	0.4543	0.3886	0.8532	0.5622	0.2557

Where,

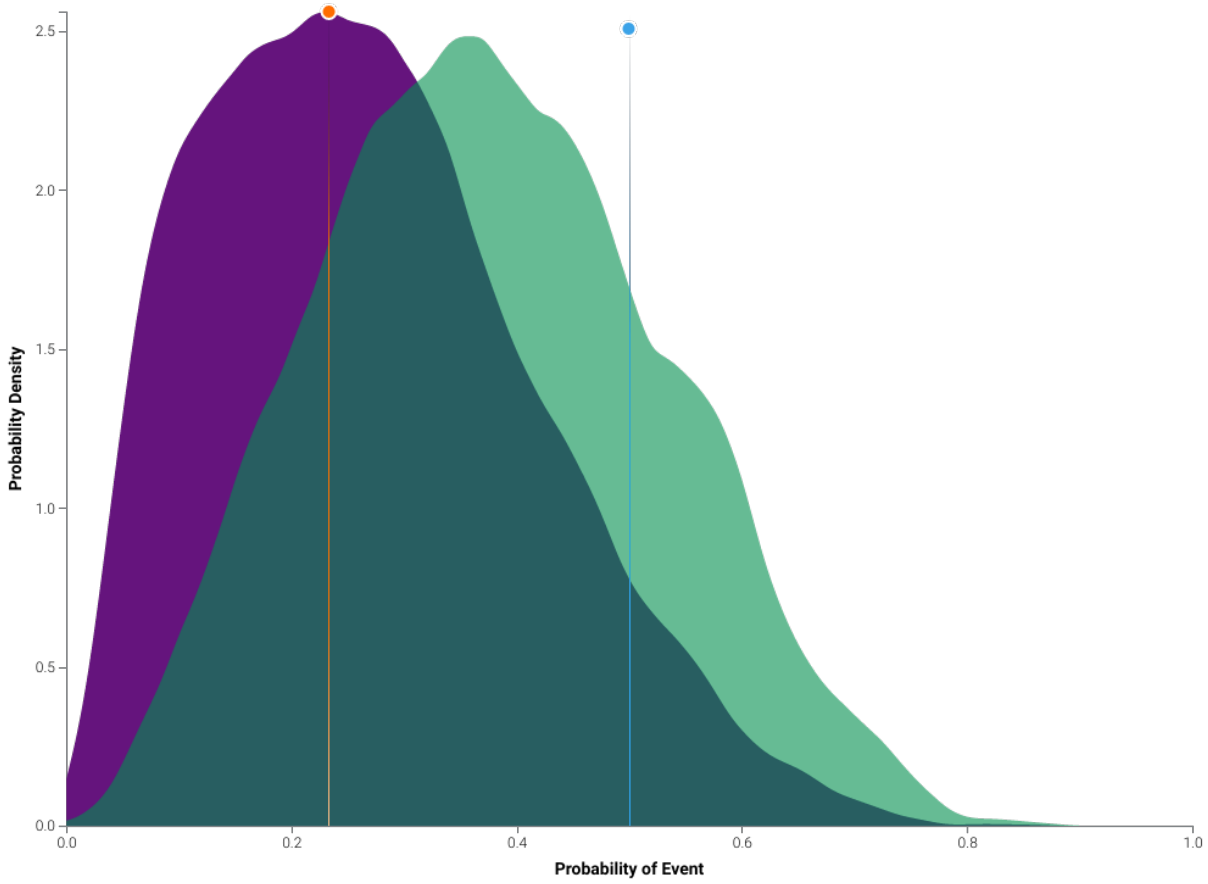
- F1 Score: A measure of the model's accuracy, computed based on precision and recall.
- True Positive Rate: Sensitivity or recall. The ratio of true positives (correctly predicted as positive) to all actual positives.
- False Positive Rate: Fallout. The ratio of false positives to all actual negatives.
- True Negative Rate: Specificity. The ratio of true negatives (correctly predicted as negative) to all actual negatives.
- Positive Predictive Value: Precision. For all the positive predictions, the percentage of cases in which the model was correct.
- Negative Predictive Value: For all the negative predictions, the percentage of cases in which the model was correct.
- Accuracy: The percentage of correctly classified instances.
- Matthews Correlation Coefficient: Measure of model quality when the classes are of very different sizes (unbalanced).

The Receiver Operating Characteristic (ROC) Curve allows the user to explore classification, performance, and statistics related to a selected model at any point on the probability scale. Because choosing the best model can be based on a number of parameters, it is important to understand whether the classification performance of a particular model meets predetermined specifications. The ROC Curve plots the true positive rate against the false positive rate for a given data source. The two important characteristics of the curve to consider are the area under the curve (AUC) and the shape of the curve. The AUC is a metric for binary classification that considers all possible thresholds and summarizes performance in a single value.

Below is the ROC curve for this model based on crossValidation.



The Prediction Distribution graph shown below illustrates the distribution of actual distribution density in relation to the threshold (a dividing line for interpretation of the graph). Every prediction to the left of the dividing line is classified as false and every prediction to the right of the dividing line is classified as true. Therefore, this graph illustrates how well the model discriminates between prediction classes.



6.3.5 Bias and Fairness

DataRobot's Bias and Fairness testing identifies whether the model exhibits biased behavior towards any classes in the dataset's protected features, based on the selected definition of fairness. Protected features and the fairness metric are chosen before Autopilot is started. DataRobot also provides a workflow that guides you towards an appropriate definition of fairness for the specific use case.

DataRobot's Bias and Fairness feature includes two model-level insights:

- Per-Class Bias, which shows whether the model is treating certain protected groups differently as measured by the selected fairness metric. This identifies if there is biased behavior, and if so, how that bias manifests, but not why.
- Cross-Class Data Disparity, which shows how different protected classes differ in their data distribution. This offers deeper insight into why the model is treating groups differently.

Together, these insights can help identify potential mitigation strategies for bias in the dataset and model, such as improving data collection or data sampling for specific groups.

Bias and Fairness testing was used in this project. The selected protected features were Race, Gender. The favorable target outcome was 0. The selected fairness metric was proportionalParity. The fairness threshold was set at 0.95.

Proportional Parity measures fairness by Equal Representation, and it is best suited for cases when you want representation to be normalized based on the population sizes of your protected classes. This metric measures whether your model's predictions are equivalent across each of the protected classes, in terms of the relative percentage of each population that receives favorable and unfavorable predictions. Note that metrics that measure Equal Representation can encourage your model to depart from the target distributions learned during training, which could lead to tradeoffs against measured accuracy metrics. This can be useful, however, if the model should make decisions based on bias and fairness rather than decisions that would be made from uninfluenced training data.

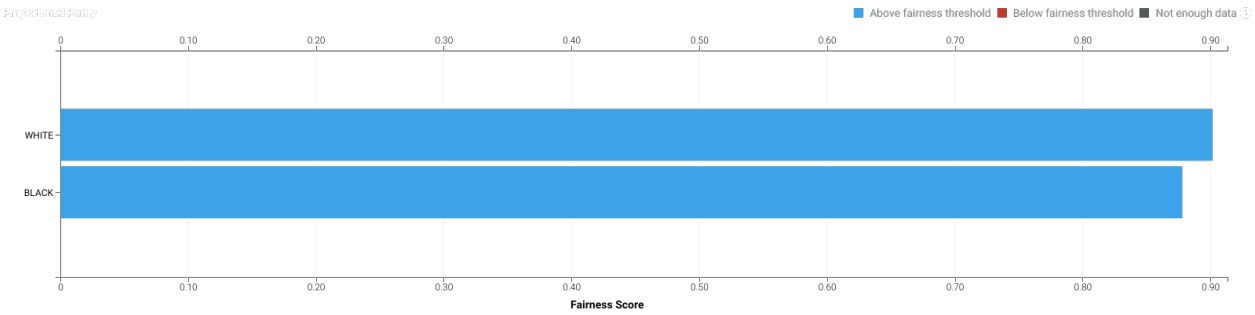
The Per-Class Bias graph shows whether the model exhibits biased behavior across protected features. The top fairness score across each protected class is scaled to 1.0, and the fairness scores for every other class are scaled relative to that value. If the fairness score for a class crosses the selected fairness threshold, the bar for that class is shown in red. If DataRobot used in-sample predictions to derive the model's performance scores (see **Overview of Model Results**), the fairness scores were calculated using in-sample validation data.

If there is not enough data for a class, its score is still calculated, but the bar for that class is shown in gray. The heuristic for whether a class does not have enough data is the following:

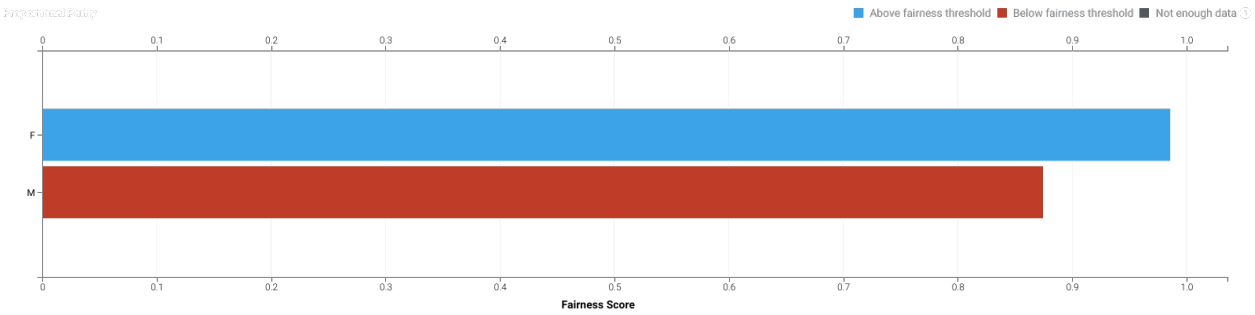
- If the class has <100 rows in the validation data, then it does not have enough data.
- If the class has between 100 and 1,000 rows in the validation data, but has fewer than <10% of the rows of the majority class, then it does not have enough data.
- If the class has >1,000 rows, then it has enough data.

The following figure is the Per-Class Bias graph for each protected feature in this project:

Race



Gender



The Cross-Class Data Disparity graph shows how different protected classes differ in their data distribution, in order to understand why the model treats each class differently based on the dataset. The X-axis depicts the feature importance of each feature in the dataset, while the Y-axis shows the Population Stability Index (PSI) for that feature compared across the two selected classes of the protected feature. The higher the feature importance, the more important that feature is to the model. The higher the PSI, the more differences there are for that feature across each of the two classes.

The following figure is the Cross-Class Data Disparity graph for each protected feature, comparing the class with the highest fairness score against the class with the lowest fairness score within each respective protected feature. If no chart is present, the feature did not have any classes with sufficient data.

7 Model Implementation and Output Reporting

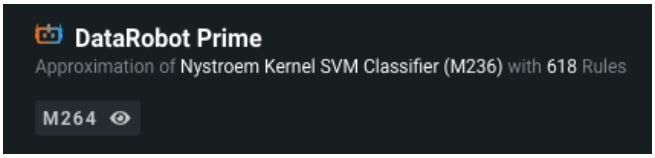
7.1 Version Control

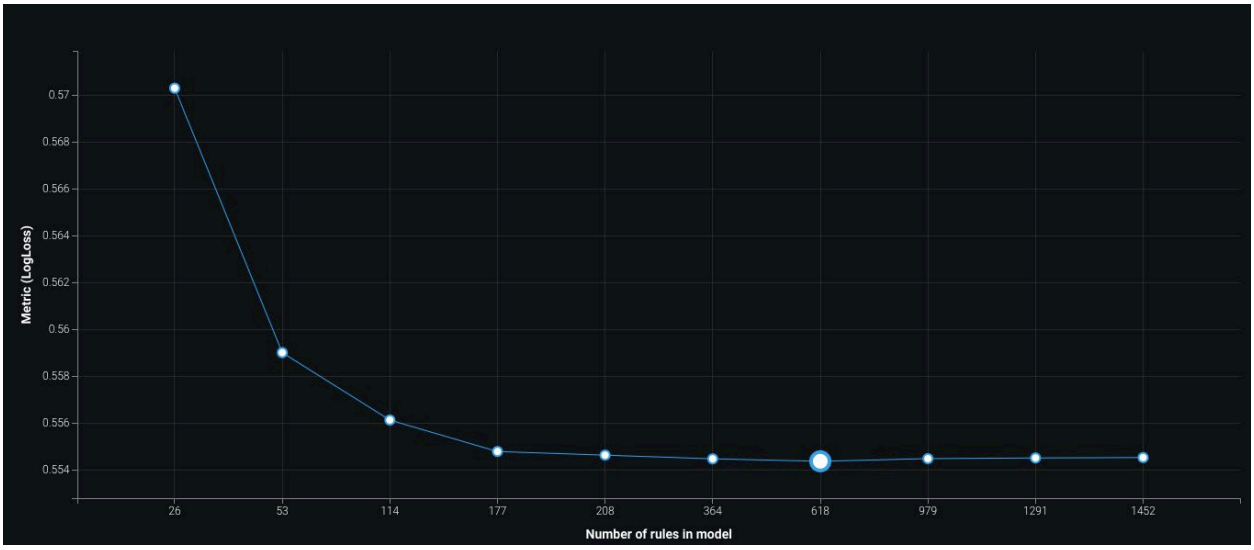
DataRobot handles model and project version control automatically by tagging each model on the Leaderboard with a unique Model ID. The Model ID represents a single instance of a model type, feature list, sample size, and set of tuning parameter values. DataRobot also maintains unique Project IDs for 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 which is an approximation. The DataRobot Prime scoring code is based on a Nystroem Kernel SVM trained on ~64% of the data. Scoring code is easy to deploy, test, and maintain on a variety of platform.

7.2 Scoring Code

The Prime Model is shown here:





Number of Rules	Log Loss Validation Values
26	.570280
53	.558987
114	.556109
177	.554765
208	.554610
364	.554453
618	.554349

An example 26 rule set scoring code file is shown below.

```
#
# -*- coding: UTF-8 -*-
# Copyright @2021. DataRobot, Inc. All Rights Reserved. Permission to use, copy, modify,
# and distribute this software and its documentation is hereby granted, provided that the
# above copyright notice, this paragraph and the following two paragraphs appear in all copies,
# modifications, and distributions of this software or its documentation. Contact DataRobot,
# 225 Franklin Street, Boston, MA, United States 02110, support@datarobot.com
# for more details.
```

```
#
# IN NO EVENT SHALL DATAROBOT BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL,
# OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST DATA, ARISING OUT OF THE USE OF THIS
# SOFTWARE AND ITS DOCUMENTATION BASED ON ANY THEORY OF LIABILITY, EVEN IF DATAROBOT HAS BEEN
# ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
# THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS".
# DATAROBOT SPECIFICALLY DISCLAIMS ANY AND ALL WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  DATAROBOT HAS NO
# OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
#

# -*- coding: utf-8 -*-
#
# Copyright 2021 DataRobot, Inc. and its affiliates.
#
# All rights reserved.
#
# DataRobot, Inc. Confidential.
#
# This is unpublished proprietary source code of DataRobot, Inc.
# and its affiliates.
#
# The copyright notice above does not evidence any actual or intended
# publication of such source code.

import calendar
from datetime import datetime
from collections import namedtuple
import re
import sys
import time
import os

import numpy as np
import pandas as pd

PY3 = sys.version_info[0] == 3
if PY3:
    string_types = str,
    text_type = str
    long_type = int
else:
    string_types = basestring,
    text_type = unicode
    long_type = long

def predict(row):
    Age_at_Release = row[u'Age_at_Release']
    Gang_Affiliated = row[u'Gang_Affiliated']
    Gender = row[u'Gender']
    Prison_Offense = row[u'Prison_Offense']
    Prison_Years = row[u'Prison_Years']
    round_Condition_Cog_Ed = np.float32(row[u'Condition_Cog_Ed'])
    round_Condition_MH_SA = np.float32(row[u'Condition_MH_SA'])
    round_Prior_Arrest_Episodes_DVCharges = np.float32(row[u'Prior_Arrest_Episodes_DVCharges'])
    round_Prior_Arrest_Episodes_Felony = np.float32(row[u'Prior_Arrest_Episodes_Felony'])
    round_Prior_Arrest_Episodes_Felony_mi = np.float32(row[u'Prior_Arrest_Episodes_Felony-mi'])
    round_Prior_Arrest_Episodes_Misd_mi = np.float32(row[u'Prior_Arrest_Episodes_Misd-mi'])
    round_Prior_Arrest_Episodes_PPViolationCharges_mi =
np.float32(row[u'Prior_Arrest_Episodes_PPViolationCharges-mi'])
    round_Prior_Arrest_Episodes_Property = np.float32(row[u'Prior_Arrest_Episodes_Property'])
    round_Prior_Arrest_Episodes_Property_mi = np.float32(row[u'Prior_Arrest_Episodes_Property-mi'])
    round_Prior_Conviction_Episodes_Misd = np.float32(row[u'Prior_Conviction_Episodes_Misd'])
    round_Prior_Conviction_Episodes_Misd_mi = np.float32(row[u'Prior_Conviction_Episodes_Misd-mi'])
    round_Prior_Revocations_Parole = np.float32(row[u'Prior_Revocations_Parole'])
    round_Supervision_Risk_Score_First = np.float32(row[u'Supervision_Risk_Score_First'])
    return sum([
        -0.7836640,
```

```
          0.37256204338043716060 * (Gang_Affiliated == u'true'),
          0.15772129275608168242 * (not Age_at_Release == u'48 or older' and
                                    round_Supervision_Risk_Score_First > 5.5 and
                                    round_Prior_Arrest_Episodes_Felony > 3.5 and
                                    round_Prior_Arrest_Episodes_Property > 0.5),
         -0.06005118877025594076 * (not Age_at_Release == u'18-22' and
                                    not Gang_Affiliated == u'true' and
                                    not Prison_Offense == u'Violent/Sex' and
                                    round_Prior_Arrest_Episodes_Misd_mi <= 0.5),
          -0.1449875815845122761 * (not Age_at_Release == u'18-22' and
                                    not Age_at_Release == u'23-27' and
                                    not Age_at_Release == u'28-32' and
                                    round_Prior_Arrest_Episodes_Felony_mi <= 0.5),
          0.07204359180493788428 * (not Gang_Affiliated == u'nan' and
                                    not Prison_Offense == u'Drug' and
                                    not Prison_Offense == u'Violent/Sex' and
                                    round_Prior_Conviction_Episodes_Misd > 0.5),
           -0.1249952311698017543 * (not Age_at_Release == u'18-22' and
                                    not Age_at_Release == u'23-27' and
                                    round_Prior_Arrest_Episodes_Felony_mi <= 0.5 and
                                    round_Prior_Revocations_Parole <= 0.5),
          0.03336763316322051420 * (not Gang_Affiliated == u'nan' and
                                    not Prison_Offense == u'Drug' and
                                    not Prison_Offense == u'Violent/Sex' and
                                    round_Condition_MH_SA > 0.5),
         -0.02359206403351475633 * (not Gang_Affiliated == u'true' and
                                    not Prison_Years == u'1-2 years' and
                                    not Prison_Years == u'Less than 1 year' and
                                    round_Supervision_Risk_Score_First <= 7.5),
          0.02420925697605558896 * (not Gender == u'F' and
                                    not Prison_Years == u'Greater than 2 to 3 years' and
                                    not Prison_Years == u'More than 3 years' and
                                    round_Prior_Revocations_Parole <= 0.5),
         -0.04777533691488510929 * (Prison_Years == u'More than 3 years'),
         -0.08785639495852445800 * (not Age_at_Release == u'18-22' and
                                    not Age_at_Release == u'23-27' and
                                    not Age_at_Release == u'28-32' and
                                    round_Prior_Conviction_Episodes_Misd_mi <= 0.5),
          0.04802465690144573640 * (round_Prior_Arrest_Episodes_PPViolationCharges_mi),
          -0.1796146619272445122 * (not Age_at_Release == u'18-22' and
                                    round_Prior_Arrest_Episodes_Property <= 2.5 and
                                    round_Prior_Arrest_Episodes_Property_mi <= 0.5 and
                                    round_Prior_Conviction_Episodes_Misd_mi <= 0.5),
           0.12317871712183220578 * (not Gang_Affiliated == u'nan' and
                                    round_Supervision_Risk_Score_First > 4.5 and
                                    round_Prior_Arrest_Episodes_Property > 0.5 and
                                    round_Condition_MH_SA > 0.5),
        -0.003195677081825905058 * (not Age_at_Release == u'18-22' and
                                    not Age_at_Release == u'23-27' and
                                    round_Prior_Arrest_Episodes_DVCharges <= 0.5 and
                                    round_Prior_Revocations_Parole <= 0.5),
          -0.0390698578107669947 * (Age_at_Release == u'48 or older'),
         -0.04776075033488830918 * (not Age_at_Release == u'23-27' and
                                    not Prison_Years == u'Less than 1 year' and
                                    round_Supervision_Risk_Score_First <= 7.5 and
                                    round_Prior_Arrest_Episodes_Felony_mi <= 0.5),
        -0.005020919648639385682 * (not Age_at_Release == u'43-47' and
                                    round_Prior_Arrest_Episodes_Felony <= 5.5 and
                                    round_Prior_Arrest_Episodes_Felony_mi <= 0.5 and
                                    round_Prior_Arrest_Episodes_Property <= 1.5),
         -0.07867367138039946250 * (not Age_at_Release == u'18-22' and
                                    not Age_at_Release == u'23-27' and
                                    round_Prior_Arrest_Episodes_Misd_mi <= 0.5 and
                                    round_Prior_Revocations_Parole <= 0.5),
          0.00178570960353369341 * (not Prison_Years == u'More than 3 years' and
                                    round_Supervision_Risk_Score_First > 6.5 and
                                    round_Prior_Conviction_Episodes_Misd > 0.5),
          0.00756014019947308553 * (round_Prior_Arrest_Episodes_Felony),
         -0.00891147671766597329 * (not Prison_Years == u'1-2 years' and
```

```
                                    not Prison_Years == u'Less than 1 year' and
                                    round_Supervision_Risk_Score_First <= 7.5 and
                                    round_Prior_Revocations_Parole <= 0.5),
          0.057556400616113528868 * (not Age_at_Release == u'48 or older' and
                                    round_Prior_Arrest_Episodes_PPViolationCharges_mi > 0.5),
         -0.052407605139358409696 * (not Gang_Affiliated == u'true' and
                                    round_Prior_Arrest_Episodes_Property <= 1.5 and
                                    round_Prior_Conviction_Episodes_Misd <= 1.5 and
                                    round_Prior_Conviction_Episodes_Misd_mi <= 0.5),
          -0.01367717953231029418 * (not Age_at_Release == u'18-22' and
                                    not Prison_Offense == u'nan' and
                                    round_Prior_Arrest_Episodes_Property <= 2.5 and
                                    round_Prior_Arrest_Episodes_Property_mi <= 0.5),
        -0.0046171086683386404828 * (not Gang_Affiliated == u'true' and
                                    not Prison_Years == u'Less than 1 year' and
                                    round_Condition_Cog_Ed <= 0.5)    ])

def get_type_conversion():
    return {
        u'Prior_Arrest_Episodes_PPViolationCharges': {'convert_func': parse_nonstandard_na,
'convert_args': None},
        u'Prior_Arrest_Episodes_Property': {'convert_func': parse_nonstandard_na, 'convert_args': None},
        u'Prior_Conviction_Episodes_Misd': {'convert_func': parse_nonstandard_na, 'convert_args': None},
        u'Prior_Arrest_Episodes_Felony': {'convert_func': parse_nonstandard_na, 'convert_args': None},
        u'Prior_Arrest_Episodes_Misd': {'convert_func': parse_nonstandard_na, 'convert_args': None},}
INDICATOR_COLS = [u'Prior_Arrest_Episodes_Felony', u'Prior_Arrest_Episodes_Misd',
u'Prior_Arrest_Episodes_PPViolationCharges', u'Prior_Arrest_Episodes_Property',
u'Prior_Conviction_Episodes_Misd']

IMPUTE_VALUES = {
    u'Condition_Cog_Ed': 0.000000,
    u'Condition_MH_SA': 1.000000,
    u'Prior_Arrest_Episodes_DVCharges': 0.000000,
    u'Prior_Arrest_Episodes_Felony': 4.000000,
    u'Prior_Arrest_Episodes_Misd': 2.000000,
    u'Prior_Arrest_Episodes_PPViolationCharges': 1.000000,
    u'Prior_Arrest_Episodes_Property': 1.000000,
    u'Prior_Conviction_Episodes_Misd': 1.000000,
    u'Prior_Revocations_Parole': 0.000000,
    u'Supervision_Risk_Score_First': 6.000000,}


def bag_of_words(text):
    """ set of whole words  in a block of text """
    if type(text) == float:
        return set()

    return set(word.lower() for word in
                re.findall(r'\w+', text, re.UNICODE | re.IGNORECASE))


def parse_date(x, date_format):
    """ convert date strings to numeric values. """
    try:
        # float values no longer pass isinstance(x, np.float64)
        if isinstance(x, (np.float64, float)):
            x = long_type(x)
        if '%f' in date_format and date_format.startswith('v2'):
            temp = str(x)
            if re.search('[\+-][0-9]+$', temp):
                temp = re.sub('[\+-][0-9]+$', '', temp)

            date_format = date_format[2:]
            dt = datetime.strptime(temp, date_format)
            sec = calendar.timegm(dt.timetuple())
            return sec * 1000 + dt.microsecond // 1000
        elif '%M' in date_format:
            temp = str(x)
            if re.search('[\+-][0-9]+$', temp):
```

```
                temp = re.sub('[\+-][0-9]+$', '', temp)

            return calendar.timegm(datetime.strptime(temp, date_format).timetuple())
        else:
            return datetime.strptime(str(x), date_format).toordinal()
    except:
        return float('nan')


def parse_percentage(s):
    """ remove percent sign so percentage variables can be converted to numeric """
    if isinstance(s, float):
        return s
    if isinstance(s, int):
        return float(s)
    try:
        return float(s.replace('%', ''))
    except:
        return float('nan')

def parse_nonstandard_na(s):
    """ if a column contains numbers and a unique non-numeric,
        then the non-numeric is considered to be N/A
    """
    try:
        ret = float(s)
        if np.isinf(ret):
            return float('nan')
        return ret
    except:
        return float('nan')

def parse_length(s):
    """ convert feet and inches as string to inches as numeric """
    try:
        if '"' in s and "'" in s:
            sp = s.split("'")
            return float(sp[0]) * 12 + float(sp[1].replace('"', ''))
        else:
            if "'" in s:
                return float(s.replace("'", '')) * 12
            else:
                return float(s.replace('"', ''))
    except:
        return float('nan')

def parse_currency(s):
    """ strip currency characters and commas from currency columns """
    if not isinstance(s, text_type):
        return float('nan')
    s = re.sub(u'[\$\u20AC\u00A3\uFFE1\u00A5\uFFE5]|(EUR)', '', s)
    s = s.replace(',', '')
    try:
        return float(s)
    except:
        return float('nan')


def parse_currency_replace_cents_period(val, currency_symbol):
    try:
        if np.isnan(val):
            return val
    except TypeError:
        pass
    if not isinstance(val, string_types):
        raise ValueError('Found wrong value for currency: {}'.format(val))
    try:
        val = val.replace(currency_symbol, "", 1)
        val = val.replace(" ", "")
```

```
        val = val.replace(",", "")
        val = float(val)
    except ValueError:
        val = float('nan')
    return val


def parse_currency_replace_cents_comma(val, currency_symbol):
    try:
        if np.isnan(val):
            return val
    except TypeError:
        pass
    if not isinstance(val, string_types):
        raise ValueError('Found wrong value for currency: {}'.format(val))
    try:
        val = val.replace(currency_symbol, "", 1)
        val = val.replace(" ", "")
        val = val.replace(".", "")
        val = val.replace(",", ".")
        val = float(val)
    except ValueError:
        val = float('nan')
    return val


def parse_currency_replace_no_cents(val, currency_symbol):
    try:
        if np.isnan(val):
            return val
    except TypeError:
        pass
    if not isinstance(val, string_types):
        raise ValueError('Found wrong value for currency: {}'.format(val))
    try:
        val = val.replace(currency_symbol, "", 1)
        val = val.replace(" ", "")
        val = val.replace(",", "")
        val = val.replace(".", "")
        val = float(val)
    except ValueError:
        val = float('nan')
    return val

def parse_numeric_types(ds):
    """ convert strings with numeric types (date, currency, etc.)
        to actual numeric values """
    TYPE_CONVERSION = get_type_conversion()
    for col in ds.columns:
        if col in TYPE_CONVERSION:
            convert_func = TYPE_CONVERSION[col]['convert_func']
            convert_args = TYPE_CONVERSION[col]['convert_args']
            ds[col] = ds[col].apply(convert_func, args=convert_args)
    return ds

def sanitize_name(name):
    safe = name.strip().replace("-", "_").replace("$", "_").replace(".", "_")
    safe = safe.replace("{", "_").replace("}", "_")
    safe = safe.replace('"', '_')
    safe = safe.replace("\n", "_")
    safe = safe.replace("\r", "_")
    return safe

def rename_columns(ds):
    new_names = {}
    existing_names = set()
    blank_index = 0
    for old_col in ds.columns:
        col = sanitize_name(old_col)
```

```
        if col == '':
            col = 'Unnamed: %d' % blank_index
            blank_index += 1
        if col in existing_names:
            raise ValueError('Duplication detected. Column with name=['
                        + old_col + '] was preprocessed to['
                        + col + '] that already exists')
        existing_names.add(col)
        new_names[old_col] = col
    ds.rename(columns=new_names, inplace=True)
    return ds

def add_missing_indicators(ds):
    for col in INDICATOR_COLS:
        ds[col + '-mi'] = ds[col].isnull().astype(int)
    return ds

def impute_values(ds):
    for col in ds:
        if col in IMPUTE_VALUES:
            ds.loc[ds[col].isnull(), col] = IMPUTE_VALUES[col]
    return ds

BIG_LEVELS = {
    u'Gang_Affiliated': [
        u'false',
        u'true',
    ],
    u'Gender': [
        u'F',
        u'M',
    ],
    u'Prison_Years': [
        u'1-2 years',
        u'Greater than 2 to 3 years',
        u'Less than 1 year',
        u'More than 3 years',
    ],
    u'Prison_Offense': [
        u'Drug',
        u'Other',
        u'Property',
        u'Violent/Non-Sex',
        u'Violent/Sex',
    ],
    u'Age_at_Release': [
        u'18-22',
        u'23-27',
        u'28-32',
        u'33-37',
        u'38-42',
        u'43-47',
        u'48 or older',
    ],
}


SMALL_NULLS = {
    u'Gender': 1,
    u'Age_at_Release': 1,
    u'Prison_Years': 1,
}


VAR_TYPES = {
    u'Age_at_Release': 'C',
    u'Condition_Cog_Ed': 'N',
    u'Condition_MH_SA': 'N',
    u'Gang_Affiliated': 'C',
```

```
    u'Gender': 'C',
    u'Prior_Arrest_Episodes_DVCharges': 'N',
    u'Prior_Arrest_Episodes_Felony': 'N',
    u'Prior_Arrest_Episodes_Misd': 'N',
    u'Prior_Arrest_Episodes_PPViolationCharges': 'N',
    u'Prior_Arrest_Episodes_Property': 'N',
    u'Prior_Conviction_Episodes_Misd': 'N',
    u'Prior_Revocations_Parole': 'N',
    u'Prison_Offense': 'C',
    u'Prison_Years': 'C',
    u'Supervision_Risk_Score_First': 'N',
}


def combine_small_levels(ds):
    for col in ds:
        if BIG_LEVELS.get(col, None) is not None:
            mask = np.logical_and(~ds[col].isin(BIG_LEVELS[col]), ds[col].notnull())
            if np.any(mask):
                ds.loc[mask, col] = 'small_count'
        if SMALL_NULLS.get(col):
            mask = ds[col].isnull()
            if np.any(mask):
                ds.loc[mask, col] = 'small_count'
        if VAR_TYPES.get(col) == 'C' or VAR_TYPES.get(col) == 'T':
            mask = ds[col].isnull()
            if np.any(mask):
                if ds[col].dtype == float:
                    ds[col] = ds[col].astype(object)
                ds.loc[mask, col] = 'nan'
    return ds

# N/A strings in addition to the ones used by Pandas read_csv()
NA_VALUES = ['null', 'na', 'n/a', '#N/A', 'N/A', '?', '.', '', 'Inf', 'INF', 'inf', '-inf', '-Inf', '-
INF', ' ', 'None', 'NaN', '-nan', 'NULL', 'NA', '-1.#IND', '1.#IND', '-1.#QNAN', '1.#QNAN', '#NA', '#N/A
N/A', '-NaN', 'nan']

# True/False strings in addition to the ones used by Pandas read_csv()
TRUE_VALUES = ['TRUE', 'True', 'true']
FALSE_VALUES = ['FALSE', 'False', 'false']

DEFAULT_ENCODING = 'utf8'

REQUIRED_COLUMNS =
[u"Age_at_Release",u"Condition_Cog_Ed",u"Condition_MH_SA",u"Gang_Affiliated",u"Gender",u"Prior_Arrest_Epis
odes_DVCharges",u"Prior_Arrest_Episodes_Felony",u"Prior_Arrest_Episodes_Misd",u"Prior_Arrest_Episodes_PPVi
olationCharges",u"Prior_Arrest_Episodes_Property",u"Prior_Conviction_Episodes_Misd",u"Prior_Revocations_Pa
role",u"Prison_Offense",u"Prison_Years",u"Supervision_Risk_Score_First"]


def validate_columns(column_list):
    if set(REQUIRED_COLUMNS) <= set(column_list):
        return True
    else :
        raise ValueError("Required columns missing: %s" %
                         (set(REQUIRED_COLUMNS) - set(column_list)))

def convert_bool(ds):
    TYPE_CONVERSION = get_type_conversion()
    for col in ds.columns:
        if VAR_TYPES.get(col) == 'C' and ds[col].dtype in (int, float):
            mask = ds[col].notnull()
            ds[col] = ds[col].astype(object)
            ds.loc[mask, col] = ds.loc[mask, col].astype(text_type)
        elif VAR_TYPES.get(col) == 'N' and ds[col].dtype == bool:
            ds[col] = ds[col].astype(float)
        elif ds[col].dtype == bool:
            ds[col] = ds[col].astype(text_type)
        elif ds[col].dtype == object:
```

```
            if VAR_TYPES.get(col) == 'N' and col not in TYPE_CONVERSION:
                mask = ds[col].apply(lambda x: x in TRUE_VALUES)
                if np.any(mask):
                    ds.loc[mask, col] = 1
                mask = ds[col].apply(lambda x: x in FALSE_VALUES)
                if np.any(mask):
                    ds.loc[mask, col] = 0
                ds[col] = ds[col].astype(float)
            elif TYPE_CONVERSION.get(col) is None:
                mask = ds[col].notnull()
                ds.loc[mask, col] = ds.loc[mask, col].astype(text_type)
    return ds

def get_dtypes():
    return {a: object for a, b in VAR_TYPES.items() if b == 'C'}

def predict_dataframe(ds):
    return ds.apply(predict, axis=1)

def run_dataframe(ds):
    ds = rename_columns(ds)
    ds = convert_bool(ds)
    validate_columns(ds.columns)
    ds = parse_numeric_types(ds)
    ds = add_missing_indicators(ds)
    ds = impute_values(ds)
    ds = combine_small_levels(ds)
    prediction = 1/(1 + np.exp(-predict_dataframe(ds)))
    return prediction


def run(dataset_path, output_path, encoding=None):
    if encoding is None:
        encoding = DEFAULT_ENCODING

    ds = pd.read_csv(dataset_path, na_values=NA_VALUES, low_memory=False,
                     dtype=get_dtypes(), encoding=encoding)

    prediction = run_dataframe(ds)
    prediction_file = output_path
    prediction.name = 'Prediction'
    prediction.to_csv(prediction_file, header=True, index_label='Index')


def _construct_parser():
    import argparse

    parser = argparse.ArgumentParser(description='Make offline predictions with DataRobot Prime')

    parser.add_argument(
        '--encoding',
        type=str,
        help=('the encoding of the dataset you are going to make predictions with. '
              'DataRobot Prime defaults to UTF-8 if not otherwise specified. See the '
              '"Codecs" column of the Python-supported standards chart '
              '(https://docs.python.org/2/library/codecs.html#standard-encodings) '
              'for possible alternative entries.'),
        metavar='<encoding>'
    )
    parser.add_argument(
        'input_path',
        type=str,
        help=('a .csv file (your dataset); columns must correspond to the '
              'feature set used to generate the DataRobot Prime model.'),
        metavar='<data_file>'
    )
    parser.add_argument(
        'output_path',
        type=str,
```

```
        help='the filename where DataRobot writes the results.',
        metavar='<output_file>'
    )

    return parser


def _parse_command(args):
    parser = _construct_parser()
    parsed_args = parser.parse_args(args[1:])

    if parsed_args.encoding is None:
        sys.stderr.write('Warning: For input data encodings other than UTF-8, '
                         'search "Prime examples" in the DataRobot Users Guide at
https://app.datarobot.com/docs/users-guide/index.html')
        parsed_args.encoding = DEFAULT_ENCODING

    return parsed_args


if __name__ == '__main__':
    args = _parse_command(sys.argv)
    run(args.input_path, args.output_path, encoding=args.encoding)
```

7.3 Rulefit Classification Insights

As part of the extensive DataRobot model search, various project insights can be obtained. Here are the top 10 rulefit explanations for cohorts in the data.

Rule	Lift	Mean Rel. Target	Mean Target	Observations [%]
Age_at_Release == "18-22" & Age_at_Release != "33-37" & Gang_Affiliated == "true"	0.80714873	1.80714873	0.539325843	3.08492201
Age_at_Release == "18-22" & Gang_Affiliated == "true"	0.80714873	1.80714873	0.539325843	3.08492201
Gang_Affiliated == "true" & Supervision_Level_First == "Specialized"	0.695562739	1.695562739	0.506024096	5.75389948
Gang_Affiliated == "true" & Prior_Arrest_Episodes_PPViolationCharges <= 1.5 & Condition_MH_SA > 0.5	0.68379645	1.68379645	0.502512563	6.897746967
Gang_Affiliated == "true" & Prior_Arrest_Episodes_Felony > 1.5	0.631393811	1.631393811	0.486873508	14.52339688
Education_Level != "At least some college" & Gang_Affiliated == "true"	0.616523459	1.616523459	0.482435597	14.80069324

Age_at_Release == "18-22" & Prior_Arrest_Episodes_Felony > 2.5	0.612393353	1.612393353	0.481203008	4.610051993
Gang_Affiliated == "true"	0.575520533	1.575520533	0.470198675	15.70190641
Age_at_Release == "18-22" & Residence_PUMA > 10.5 & Prior_Arrest_Episodes_Felony > 1.5	0.539536052	1.539536052	0.459459459	3.847487002
Gang_Affiliated == "true" & Dependents <= 1.5	0.535762679	1.535762679	0.458333333	13.3102253