[redacted]
Prativa Basnet
[redacted]

Introduction

The Finance industry uses a number of descriptive and predictive analytical techniques to address business problems and maximize profits. Often this work involves binary classification tasks, where the goal is to classify and input data into two mutually exclusive categories. This form of classification categorizes observations in a dataset into one of two classes, typically represented in coding languages as 0 and 1. Binary classification is used in FinTech to address issues such as financial fraud claims, credit and investment risks, and bankruptcy risks. Predictive analytics using binary classification often involves modeling using logistic regression, decision trees, XG Boost, and many other statistical and machine learning techniques. Equifax data sourced to Atlanticus for this project includes trade line performance on accounts booked outside of Atlanticus. The dataset provided contains 541 variables and 988,267 observations from June 2021. Atlanticus supplied Kennesaw State University with the dataset, with the intention of having students develop a predictive model for determining "good" credit risks from "bad" risks. The measure of risk is based on likelihood of response and default, with both representing potential cash flow disruption, and with default, a potential collection cost increase for lenders. The Atlanticus dataset utilized for this project includes two binary classifications under the "goodbad" dependent variable. Our two classifications are response versus nonresponse to credit mailers, and default versus no default once credit is issued. The goal of our project is to predict whether individuals will respond to a credit offer mailer and whether, when issued credit, they will default. Once the credit risk models were developed and hyperparameters tuned, they were scored on an out-of-time dataset from July 2021 with similar data to our test set.

Methods

Data Processing

To determine which variables are most responsible for variation within the dataset, and thus integral to our classification model, we performed several data cleaning and preprocessing steps. The first step in this cleaning process involved the handling of coded missing data. The different "ams" variables in the dataset all have different max values based on the scale and size of digits, as such no data is "missing." Any truly missing data is coded to 99xx, depending on the number of digits in the max value for each variable. Before any imputation could be done on the data the coded missing data had to be standardized. Our team elected to use SAS 9.4 functions provided and altered from Atlanticus, to recode all missing data. In addition, before imputation could be done, variables containing only one level including zero and missing values were deleted. These features were removed because constant columns with no variability do not provide substantial useful information for analysis and removing them can improve the quality of the analysis by better allocating computational resources, simplifying the high dimensional dataset to make it more manageable, and focusing on more meaningful features. Following the recoding of missing values, variables were dropped on the basis of the presence of missing amounts above 30% (Appendix, Table 1). This was a choice supported by the literature that allowed for more manageability of the dataset as well as computational resource constraints (Markov et al, 2022).

The next step in the project workflow included a couple of different pipelines for imputation of missing values. Imputation is a crucial precursor for clustering of variables and feature selection. Median imputation was performed first, chosen due to its traditional usage in credit risk modeling and relative ease of implementation. Multiple imputation methods however, such as MICE, have been shown to be preferable in predictive analytics to minimize bias and improve model performance (Mera-Gaona et al, 2021). Our team attempted several other imputation measures, including mean, KNN, and MICE, to maximize data preservation for the model, however given the size of the dataset and available computing resources, these

methods were not feasible with the raw dataset. Once feature selection had been completed using the median imputation pipeline, we took the original raw dataset and dropped all variables except those selected through clustering/binning (Figure 1) and performed MICE imputation in Python to compare the variations in model performance of the two different methods of dealing with missing values.

Clustering: Variable Selection

Cluster analysis of variables using principal components was performed on the median-imputed data. The comparison of output from this methodology allowed for variable reduction and selection. By definition many of the variables in the dataset are highly correlated (repeated time measures such as 6-month, 9-month, and 12-months for the same feature), and as such modeling without accounting for this in variable selection would be impossible. Variables were further reduced that had Variable Inflation Factors above 10, a value representative of the relative multicollinearity, yielding a final 175 variables (Appendix, Table 3). Figure 1 shows the Clusters and Variance Explained Plot used for variable selection with clustering. Ten clusters were chosen based on the variance explained proportion of 0.6. From each cluster the most representative variables for each cluster that also demonstrated the least correlation to adjacent clusters were chosen. 68 variables were selected through the cluster analysis, all with VIF values below 10 (Appendix, Table 4).

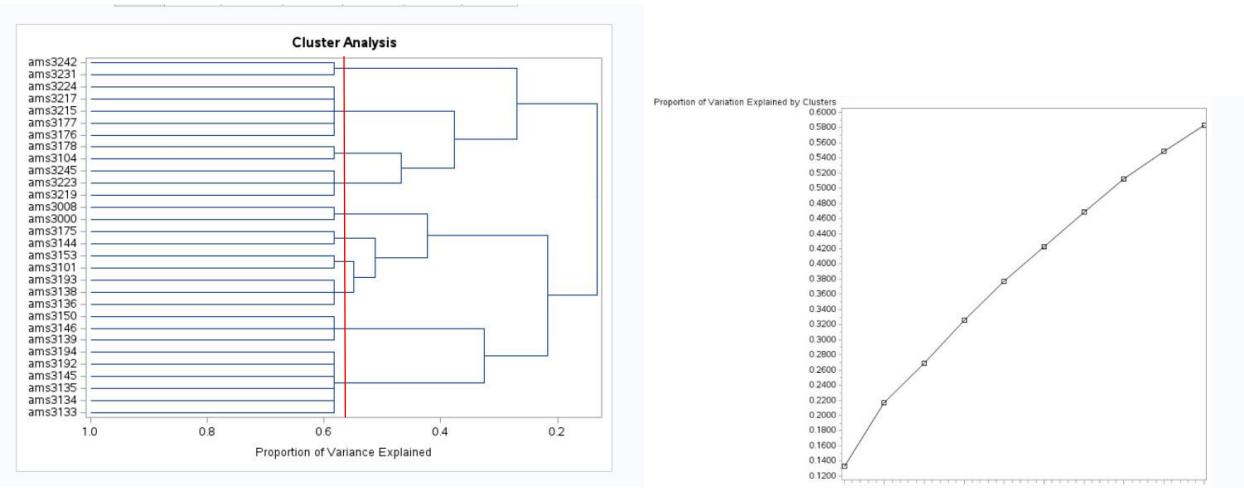


Figure 1: Cluster Analysis and Variance Explained

Discretization

To address issues of scale within our selected variables binning was performed in SAS 9.4. Binning continuous variables changes them into bins or groups, which allows one to calculate information values and other metrics that can be used to measure the predictive power an independent variable has on the dependent variable. In practice, typically variables are discretized into 10-20 bins (Zeng, 2013). When considering different numbers of bins and different binning strategies (bucket, quantile, winsorizing) a change in information values was not observed. We used 10 bins based on the output from our analysis. We utilized quantile binning since each bin was filled with observations with this method, unlike bucket and winsorized binning where some bins were empty. The interpretation for information value in Siddiqi (2006) can be seen in Appendix, Table 5. Our information values from the binning process were weak on nearly all of the variables except 18, and only 4 had strong predictive power.

Brandi Jones
Prativa Basnet
Nina Grundlingh

Random Forest Variable Selection Exploration

Low predictive power from the median imputation workflow including clustering and binning led our team to explore random forest as a means of variable selection as well. Random forest was run on the variables selected after variable clustering. The OOB Gini against each variable is plotted in Figure . The Gini index is used to measure the probability for an instance of an observation being misclassified when chosen at random. The Gini index ranges from 0-0.5 where a value of 0 indicates no misclassification is probable and 0.5 high misclassification probable (this is the same as a random assignment). According to parameters described by Nyongesa (2020), the top 19 variables are important for the model (ams3951 down to ams3316). These variables were identified as more likely to result in a parsimonious model prediction.

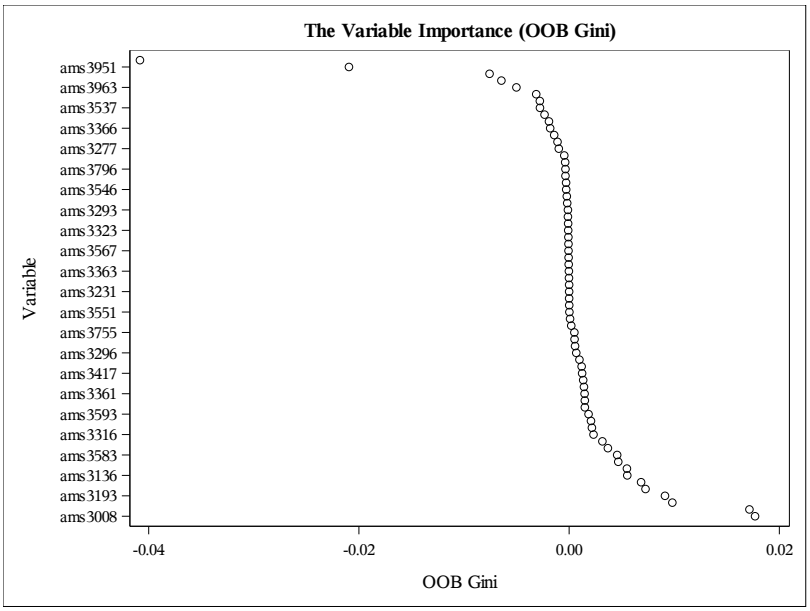


Figure 2: Variable Importance (OOB Gini)

After training the random forest selected variables, we utilize binning for each according to the values the random forest used to split the variables. Each decision tree in the random forest has split points for the continuous variables at different levels of the tree. We limited the tree depth to 20 since in practice, bins of 10-20 are generally used (Zeng, 2013). Once all the unique split points for each variable are collected, we sort these in ascending order and use them to bin or discretize our variables (Siddiqi, 2006).

Following training and binning of the random forest selected variables, a lack of improvement in predictive power led to alteration of our final workflow to include recoding of missing values, dropping of features with <30% missing, cluster analysis for variable selection, median and MICE imputation (for performance effect comparison), and discretization using binning following the feature selection and elimination. After comparison of imputation methods, MICE imputation was used for final models due to superior model results. These methods led to the creation of the cleaned datasets used for modeling, described in Table 1.

Brandi Jones
Prativa Basnet
Nina Grundlingh

Table 1: Datasets used for modeling.

Model	Number of Observations	Number of Variables
Response	761,516	57
Default	187,842	48

Modeling and Scoring

The team decided to try three different model pipelines for both our response and default predictions. Response and Default were modeled separately for all model methods. The models used were logistic regression, random forest, and XGBoost. Performance and profitability of models was examined following completion on the test datasets. Train/Test/Validation split methods were used for all model types. Random Forest/Logistic Models used 80%/20% train/test split, while XGBoost Model used a 60%:20%:20% train/validate/test split.

Logistic Regression

Separate logistic regression models for default and response were run using SAS, and specifically utilized the high-performance logistic model function rather than the traditional logistic function in SAS. Stepwise variable selection was used for both response and default models. Different cut point values were considered for the logistic output for response as well as default. The final cut point value selected was chosen to ensure that the true positive predictions and true negative predictions were maximized, and false negatives minimized. Model variables and cut points for logistic regression listed in Table 2.

Model	Variables	Cut Point Values
Response	ams3224 ams3296	0.245
Default	ams3257, ams3609, ams3633, ams3796, ams3224, ams3288, ams3581, ams3757	0.46

Table 2: Logistic Regression model scoring metrics.

Random Forest: Weighted and Unweighted

Four separate random forest models for default and response were run using R Studio. In an effort to counteract the imbalance in the response dataset and improve predictions, a class weight was added to one random forest response and one random forest default model. An unweighted random forest response and an unweighted random forest default model were run as well for comparison. The class weight calculation used is seen in Table 3. Cut points for models followed similar methodology to that of logistic models seen above.

	Proportion	Weight Calculation
Response	0.246	1/proportion = 4.065
Nonresponse	0.754	1/proportion = 1.325

Table 3: Random Forest model weight class calculation.

Brandi Jones
Prativa Basnet
Nina Grundlingh

XGBoost Modeling

XGBoost models used an altered workflow from the preliminary approach seen in logistic and random forest modelling. Following variable selection for Response and Default models, MICE imputation only was used for this pipeline. Columns with 30% missing values were dropped based on prior literature approach, and to better allocate computational resources in Python for modeling. Bayesian Optimization was used for hyperparameter tuning with 30 iterations to determine the "best parameters." Response and Default models were run separately as with logistic and random forest. Scoring methodology was similar to that of logistic described above.

Uncertainty and Calibration Methods

To better understand each model's confidence in predictions, we set out to quantify the uncertainty of the predictions. We define a prediction to be uncertain if the probability falls between 0.4-0.6. Otherwise, we say the prediction is 'quite certain'. The uncertainty in predictions in relation to the correctness of the predictions can be seen in Table to Table . The predictions for our Responders model for the logistic regression and Unweighted Random Forest had 100% certainty, or almost 100% predictions being quite certain (Table and Table). The weighted random forest for the default model almost had complete uncertainty with almost all predicted probabilities being near 0.5 (Table).

Table 4: Logistic regression prediction certainty for the responders model
Quite certain=<0.6
Uncertain=0.4-0.6

	Quite Certain	**Uncertain**
correct	75.20%	0
wrong	24.80%	0

Table 5: Logistic regression predictions for the default model

	Quite Certain	**Uncertain**
correct	29.26%	29.59%
wrong	18.42%	22.74%

Table 6: Random forest (unweighted) predictions for the responders model

	Quite Certain	**Uncertain**
correct	73.08%	1.92%
wrong	23.95%	1.05 %

Table 7: Random forest (unweighted) predictions for the default model

	Quite Certain	**Uncertain**
correct	30.18%	29.49%
wrong	18.00%	22.34%

Brandi Jones
Prativa Basnet
Nina Grundlingh

Table 8: Random forest (weighted) predictions for the responders model

	Quite Certain	Uncertain
correct	29.91%	31.10%
wrong	12.19%	26.80 %

Table 9: Random forest (weighted) predictions for the default model

	Quite Certain	Uncertain
correct	0.60%	54.81%
wrong	0.30%	44.29%

Table 10: XGBoost predictions for the response model

	Quite Certain	Uncertain
correct	78.47%	0.98%
wrong	19.15%	1.40%

Table 11: XGBoost predictions for the default model

	Quite Certain	Uncertain
correct	72.19%	4.89%
wrong	18.67%	4.25%

Ensemble classifiers, such as random forest and boosted trees, seldom return predicted probabilities close to 0 or 1 because they average responses from multiple models. Niculescu-Mizil (2005) found that boosted trees do not return well-calibrated probabilities and can thus benefit from calibration. Calibration is a technique used to adjust the predictions of a model so that they are probabilistically meaningful.

Formally, a model is said to be perfectly calibrated if for any p, a prediction class with confidence p is correct $100 * p$ percent of the time. For example, if a model predicts 1% of customers to default and in reality, 1% actually do default then the model is said to be well-calibrated. Calibration is then a measure of discrepancy between what we see empirically and what we expect in theory.

Model calibration is measured using scoring functions such as brier score or negative log loss as well as plots knowns as calibration plots or curves or reliability diagrams. The plots are created by binning the predictions and obtaining the mean predicted probability for each bin. Only bins with 10 or more predictions were considered to remove noise.

The brier score is defined as follows:

$$BS = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2$$

Where p is the probability of the event occurring and y_i is 1 if the event occurs and 0 otherwise. The brier score takes values between 0 and 1. A lower brier score suggests that the model is more calibrated which is preferred.

A model can be calibrated by fitting a calibration function. The calibration function should be monotonic, minimize the scoring rule and trained on an independent data set. Doing calibration on an independent dataset avoids overfitting. Platt Scaling is a common method used to

calibrate a model. Platt Scaling fits a logistic regression model to the output probabilities generated by the binary classifier.
Let the output of a binary classifier be $f(x)$. To get calibrated probabilities, pass the output through a sigmoid:

$$P(y = 1|f) = \frac{1}{1 + \exp(Af + B)}$$

where parameters A & B are fitted using maximum likelihood estimation from a fitting training set (f_i, y_i).
Isotonic regression is also used to calibrate models. This fits a non-parametric, piecewise, non-decreasing line to a sequence of points in such a way as to make the line as close to the original points as possible. Unlike Platt Scaling, isotonic regression does not require the curve to be S-shaped.
Given the predictions f_i and the true targets y_i, the basic assumption in isotonic regression is that:

$$y_i = m(f_i) + \epsilon_i$$

Where m is an isotonic (monotonically increasing) function.
Then, given a training set (f_i, y_i), the isotonic regression problem is finding the isotonic function $\hat{m}$ such that

$$\hat{m} = argmin_z \sum (y_i - z(f_i))^2$$

The pair-adjacent violators (PAV) algorithm is commonly used.

A drawback to isotonic regression is that it is sensitive to outliers. This method is best used when one has a large calibration set (over 1000 examples). Calibration was performed on the unweighted random forest for both the responder and default models. This was done on a separate holdout data set which was 10% of the data, leaving 10% for testing. After calibrating the models, the plots were done on the test data. Brier score and calibration plots were used to assess the calibration before and after applying isotonic regression.
The calibration plots before model calibration is applied for the responder and default models are given in Figure and Figure , respectively.

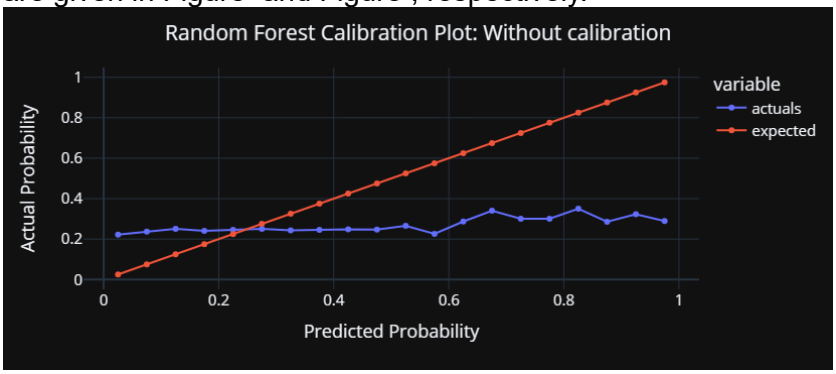


Figure 3: Calibration plot for the unweighted random forest for the responder model without calibration

Brandi Jones
Prativa Basnet
Nina Grundlingh

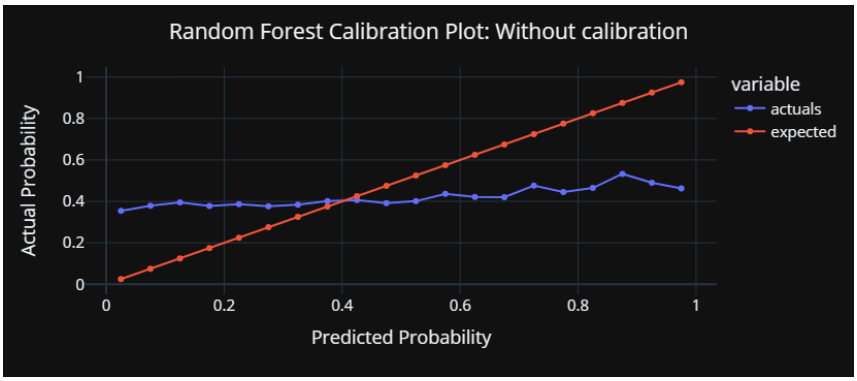


Figure 4: Calibration plot for the unweighted random forest for the default model without calibration

The calibration plots after applying isotonic regression to calibrate the responder and default models are given in Figure and Figure , respectively. In both cases the range of predicted probabilities shrink, the responder model more so than the default model.

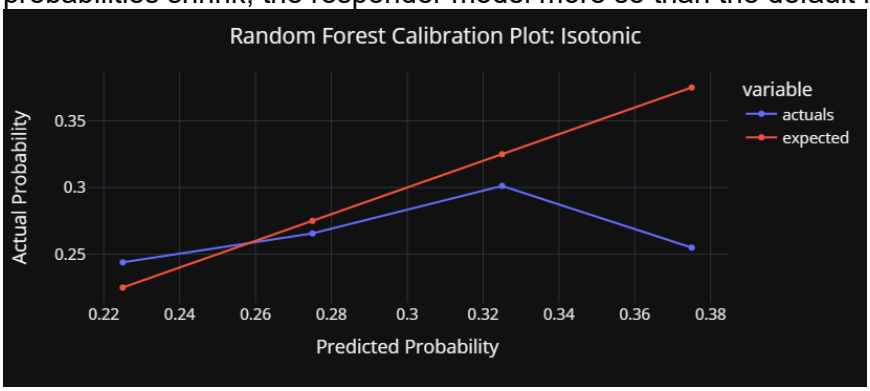


Figure 5: Calibration plot for the unweighted random forest for the responder model after applying isotonic regression

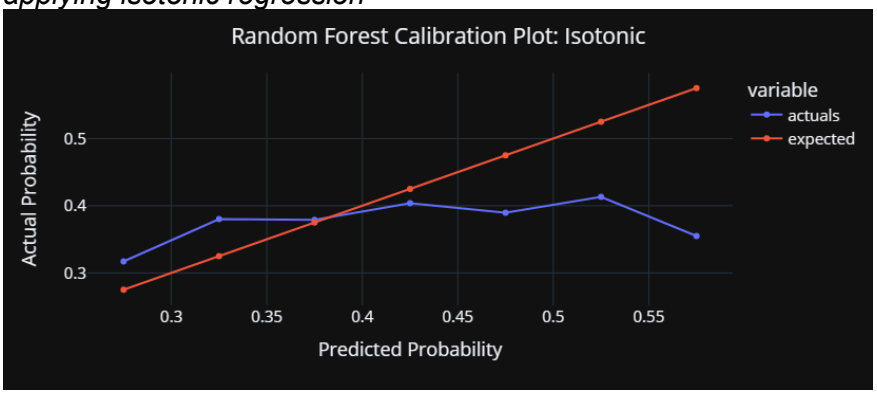


Figure 6: Calibration plot for the unweighted random forest for the default model after applying isotonic regression

To assess effectiveness of calibration, tabulated brier score, accuracy, sensitivity, specificity is examined before and after calibration for the models.

Results

Model Performance

Brandi Jones
Prativa Basnet
Nina Grundlingh

Table 12 compares the performance of each final Response model. Table 13 compares the performance of each Default model. Table 14 shows the change to model performance before and after calibration.

MODEL	F1	ACCURACY	SPECIFICITY	SENSITIVITY	PRECISION
RESPONDER RANDOM FOREST (Wt.)	0.284	0.610	0.707	0.3140	0.260
RESPONDER RANDOM FOREST (No Wt.)	0.038	0.750	0.989	0. 020	0.386
RESPONDER LOGISTIC REGRESSION	0.369	0.507	0.281	0.727	0.247
RESPONDER XGBOOST	0.206	0.664	0.9429	0.7513	0.2498

Table 12: Model comparisons for Response Models

MODEL	F1	ACCURACY	SPECIFICITY	SENSITIVITY	PRECISION
DEFAULT RANDOM FOREST (Wt.)	0.637	0.554	0.399	0.660	0.617
DEFAULT RANDOM FOREST (No Wt.)	0.041	0.597	0.988	0.021	0.597
DEFAULT LOGISTIC REGRESSION	0.137	0.538	0.936	0.080	0.473
DEFAULT XGBOOST	0.0497	0.880	0.9049	0. 9078	0.0919

Table 13: Model comparisons for Default Models

Table 14: Model evaluation before and after applying calibration

Model	**Before/after calibration**	**Brier Score**	**Accuracy**	**Sensitivity**
Responder	Before	0.2026	0.7333	0.0572
	After	0.1857	0.7531	0.0002
Default	Before	0.2610	0.5663	0.2503
	After	0.2404	0.5974	0.0054

Model	**Before/after calibration**	**Brier Score**	**Accuracy**	**Sensitivity**
Responder	Before	0.2026	0.7500	0.02028
	After	0.1857	0.7531	0.0002
Default	Before	0.2610	0.5966	0.0213
	After	0.2404	0.5974	0.0054

Brandi Jones
Prativa Basnet
Nina Grundlingh

Scoring and Selecting "Good Risk" Individuals

To optimize profit, we need to combine the outputs from the responder and default models. The objective is to send mail to those most likely to respond and least likely to default. There are multiple approaches to go about selecting individuals to mail to e.g. selecting off a certain percentage of individuals having the highest probabilities of responding and the percentage of individuals having the lowest probabilities of defaulting. We were able to achieve a profit by using the predicted probabilities as cut points for the unweighted, uncalibrated random forest models (Table 15) and the XGBOOST models (Table 16).

Table 15: Profit/loss for different combinations of cut points from the Unweighted, Uncalibrated RF models

Response predicted probs > y

Default predicted probs < x	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
0.1	-12120	-6550	-2300	-510	-90
0.2	-12120	-6550	-2300	-510	-90
0.3	-11870	-6300	-2050	-260	**+160**
0.4	-25320	-19750	-15500	-13710	-13290
0.5	-138670	-13310	-128850	-127060	-1226640

From Table 15, selecting individuals who have a predicted probability to default less than 0.3 and those who have a predicted probability to respond greater than 0.9 yields a profit of $160. This gives 15 individuals selected to mail to out of 152304 (test dataset). Table 16 shows the profit/loss values based on predicted probabilities in the XGBoost models. A max profit from these models of $75050 involves mailing to 967 individuals.

Table 26: Profit/loss for different combinations of cut points from the XGBoost models

Response predicted probs > y

Default predicted probs < x	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
0.1	+16620	+23140	+26570	+24330	+24000
0.2	+59070	+65440	+68670	+67980	**+75050**
0.3	+23020	+32140	+37470	+38830	+60770
0.4	-152130	-132560	-125930	-119120	-77450
0.5	-405580	-367560	-349830	-337170	-283700

Discussion

Based on our model performance and scoring of profitability the XGBoost models using MICE imputation are the best overall. MICE imputation improves the performance of all models, but does not increase the profitability of the random forest models enough to outpace the XGBoost modelling method. All of the pipelines explored for this project involved numerous decision points that could likely impact performance and profitability as the change in imputation

did, making the credit risk classification a unique problem with many more pipelines to explore before a true “best” model is found. In addition, weighting of XGBoost may further improve the profitability as the model is vulnerable to the imbalance in response data in much the same way as the random forest models. Examining this weight class modelling with XGBoost is a likely next step in improving profitability of the credit risk modelling pipelines discussed in this study.

Brandi Jones
Prativa Basnet
Nina Grundlingh

References

Anton Markov, Zinaida Seleznyova, Victor Lapshin (2022) Credit scoring methods: Latest trends and points to consider, The Journal of Finance and Data Science, Volume 8, Pages 180-201, https://doi.org/10.1016/j.jfds.2022.07.002.

Mera-Gaona, M., Neumann, U., Vargas-Canas, R., & López, D. M. (2021). Correction: Evaluating the impact of multivariate imputation by MICE in feature selection. PloS one, 16(12), e0261739. https://doi.org/10.1371/journal.pone.0261739

Niculescu-Mizil, A., & Caruana, R. (2005, August). Predicting good probabilities with supervised learning. In Proceedings of the 22nd international conference on Machine learning (pp. 625-632).

Nyongesa, D. (2020). Variable selection using Random Forests in SAS. In *SAS Global Forum* (Vol. 4826).

Guoping Zeng, (2013) "Metric Divergence Measures and Information Value in Credit Scoring", Journal of Mathematics, vol. 2013, Article ID 848271, 10 pages. https://doi.org/10.1155/2013/848271

Siddiqi, N. (2006). Credit risk scorecards–developing and implementing intelligent credit scoring. Wiley

Woo, MinJae (2024) XGBoost with Bayesian Optimization Code and Tuning Usage. DS 7140.

XGBoost Classifier Weight Option. (2024) https://stackoverflow.com/questions/42192227/xgboost-python-classifier-class-weight-option

Brandi Jones
Prativa Basnet
Nina Grundlingh

Appendix

Table 1: Variables Selected After 50% Missing Removal									
ams3000	ams3001	ams3008	ams3009	ams3010	ams3013	ams3014	ams3015	ams3016	ams3017
ams3018	ams3023	ams3024	ams3026	ams3027	ams3101	ams3104	ams3133	ams3134	ams3135
ams3136	ams3138	ams3139	ams3141	ams3142	ams3144	ams3145	ams3146	ams3150	ams3153
ams3175	ams3176	ams3177	ams3178	ams3192	ams3193	ams3194	ams3215	ams3217	ams3219
ams3223	ams3224	ams3228	ams3231	ams3235	ams3236	ams3237	ams3239	ams3242	ams3245
ams3246	ams3257	ams3266	ams3268	ams3270	ams3272	ams3273	ams3276	ams3277	ams3285
ams3286	ams3288	ams3290	ams3292	ams3293	ams3296	ams3297	ams3307	ams3309	ams3311
ams3312	ams3315	ams3316	ams3318	ams3320	ams3322	ams3323	ams3326	ams3327	ams3330
ams3332	ams3333	ams3335	ams3336	ams3340	ams3347	ams3348	ams3349	ams3350	ams3351
ams3352	ams3353	ams3354	ams3355	ams3356	ams3357	ams3358	ams3359	ams3360	ams3361
ams3362	ams3363	ams3364	ams3365	ams3366	ams3367	ams3369	ams3370	ams3371	ams3372
ams3373	ams3374	ams3377	ams3379	ams3380	ams3382	ams3383	ams3385	ams3386	ams3388
ams3389	ams3391	ams3392	ams3394	ams3395	ams3397	ams3398	ams3400	ams3401	ams3403
ams3404	ams3406	ams3407	ams3409	ams3410	ams3412	ams3413	ams3415	ams3417	ams3419
ams3420	ams3422	ams3423	ams3424	ams3426	ams3428	ams3430	ams3431	ams3433	ams3434
ams3435	ams3437	ams3439	ams3441	ams3442	ams3444	ams3445	ams3446	ams3448	ams3450
ams3452	ams3453	ams3455	ams3456	ams3457	ams3473	ams3474	ams3475	ams3476	ams3477
ams3478	ams3479	ams3480	ams3535	ams3536	ams3537	ams3539	ams3540	ams3544	ams3545
ams3546	ams3547	ams3548	ams3550	ams3551	ams3555	ams3556	ams3557	ams3558	ams3559
ams3561	ams3562	ams3566	ams3567	ams3568	ams3569	ams3570	ams3572	ams3573	ams3577
ams3578	ams3579	ams3581	ams3583	ams3585	ams3586	ams3589	ams3590	ams3592	ams3593
ams3594	ams3596	ams3597	ams3600	ams3601	ams3603	ams3605	ams3607	ams3608	ams3609
ams3611	ams3612	ams3614	ams3616	ams3618	ams3619	ams3622	ams3623	ams3625	ams3627
ams3629	ams3630	ams3633	ams3634	ams3724	ams3726	ams3728	ams3744	ams3745	ams3748

Table 2: Additional Variables Dropped Before Median Imputation									
ams3013	ams3014	ams3015	ams3016	ams3017	ams3018	ams3023	ams3814	ams3815	ams3838
ams3839	ams3894	ams3895	ams3896	ams3897	ams3898	ams3899	ams3914	ams3915	ams3916
ams3941	ams3942	ams3943	ams3944	ams3945	ams3946	ams3947	ams3948	ams3949	ams3973
ams3974	ams3975	ams3976	ams3977	ams3981	ams3982	ams3983	ams3984	ams3985	ams3986
ams3987	ams3988	ams3989	ams3990	ams3991	ams3992	ams3009	ams3027		

Brandi Jones
Prativa Basnet
Nina Grundlingh

Table 3: Variables With a VIF <=10									
ams3000	ams3008	ams3101	ams3104	ams3133	ams3134	ams3135	ams3136	ams3138	ams3139
ams3144	ams3145	ams3146	ams3150	ams3153	ams3175	ams3176	ams3177	ams3178	ams3192
ams3193	ams3194	ams3215	ams3217	ams3219	ams3223	ams3224	ams3231	ams3242	ams3245
ams3257	ams3266	ams3268	ams3270	ams3272	ams3273	ams3276	ams3277	ams3285	ams3286
ams3288	ams3290	ams3292	ams3293	ams3296	ams3297	ams3307	ams3309	ams3311	ams3312
ams3315	ams3316	ams3318	ams3320	ams3322	ams3323	ams3326	ams3327	ams3349	ams3350
ams3351	ams3352	ams3353	ams3354	ams3355	ams3356	ams3357	ams3358	ams3359	ams3360
ams3361	ams3362	ams3363	ams3364	ams3365	ams3366	ams3369	ams3370	ams3371	ams3372
ams3373	ams3377	ams3380	ams3382	ams3383	ams3385	ams3417	ams3419	ams3420	ams3422
ams3423	ams3433	ams3444	ams3455	ams3477	ams3478	ams3535	ams3536	ams3537	ams3539
ams3540	ams3544	ams3545	ams3546	ams3547	ams3548	ams3550	ams3551	ams3555	ams3556
ams3557	ams3558	ams3559	ams3561	ams3562	ams3567	ams3568	ams3569	ams3570	ams3572
ams3573	ams3577	ams3578	ams3579	ams3581	ams3583	ams3585	ams3586	ams3589	ams3590
ams3593	ams3603	ams3605	ams3608	ams3609	ams3611	ams3612	ams3619	ams3625	ams3627
ams3629	ams3630	ams3633	ams3634	ams3724	ams3726	ams3728	ams3749	ams3750	ams3755
ams3756	ams3757	ams3758	ams3796	ams3806	ams3860	ams3862	ams3863	ams3902	ams3903
ams3904	ams3905	ams3909	ams3950	ams3951	ams3952	ams3953	ams3956	ams3959	ams3960
ams3961	ams3962	ams3963	ams3964	ams3966					

Table 4: 68 Variables Selected from Cluster Analysis									
ams3133	ams3224	ams3193	ams3231	ams3219	ams3136	ams3008	ams3178	ams3175	ams3101
ams3297	ams3296	ams3277	ams3257	ams3327	ams3326	ams3323	ams3316	ams3293	ams3276
ams3363	ams3361	ams3366	ams3382	ams3358	ams3420	ams3417	ams3351	ams3356	ams3364
ams3377	ams3546	ams3555	ams3544	ams3545	ams3557	ams3567	ams3477	ams3551	ams3540
ams3558	ams3550	ams3537	ams3559	ams3634	ams3583	ams3630	ams3633	ams3612	ams3578
ams3724	ams3726	ams3750	ams3585	ams3755	ams3593	ams3963	ams3960	ams3796	ams3909
ams3756		ams3758	ams3904	ams3862	ams3902	ams3951	ams3966	ams3757	

Table 5: Information Value Interpretation

Information Value	Interpretation of Variable
< 0.02	Not relevant for modelling
0.02 – 0.1	Weak predictive power
0.1 – 0.3	Medium predictive power
0.3 – 0.5	Strong predictive power
> 0.5	Suspicious predictive power

R Code

```
---
title: "Atlanticus"
author: "Team PNB"
date: "May 1, 2024"
output:
  word_document: default
  html_document: default
  pdf_document: default
editor_options:
  chunk_output_type: console
---

```{r setup, include=FALSE}
# Install packages tinytex to knit the code with result

options(repos = c(CRAN = "https://cran.rstudio.com/"))
library(tinytex)
```

```{r}
#remotes::install_github("imbs-hl/ranger")
install.packages("ranger", repos = "https://cran.rstudio.com/")

```



```{r}
#install.packages("tidyverse")
#install.packages("caret")
#install.packages("ROCR")
#install.packages("vip")
#install.packages("ranger")
#install.packages("randomForest")
#install.packages("mlbench")
#install.packages("rpart.plot")
#install.packages("MLmetrics")


```

```{r}
# Read the packages
library(tidyverse)
library(ROCR)
library(vip)
library(caret)
library(ranger)
library(xgboost)
library(rpart)
```
```

```
library(ggplot2)
library(ROSE)
library(MLmetrics)
```

```{r}
# Read the dataset
model_set_responder <-
read_csv("/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_fold
er/model_set_responder.csv")
glimpse(model_set_responder)

model_set_default <-
read_csv("/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_fold
er/model_set_default.csv")
glimpse(model_set_default)
```

```{r}
# Check missing values in each variable
colSums(is.na(model_set_responder))
colSums(is.na(model_set_default))
```

```{r}
# Rename dataset
model_set_responder_final<-model_set_responder
glimpse(model_set_responder_final)

model_set_default_final<-model_set_default
glimpse(model_set_default_final)
```

Responder
```{r}
# change the target variable to factor
model_set_responder_final["goodbad_responder"] <-
lapply(model_set_responder_final["goodbad_responder"], as.factor)
glimpse(model_set_responder_final)
```

```{r}
#Check which factor level is associated with 'responder is 1' and 'nonresponder is
0'
levels(model_set_responder_final$goodbad_responder)
```

```{r}
# Check how balanced the target variable is
target_balance_responder <- table(model_set_responder_final$goodbad_responder)
print(target_balance_responder)
percentage_balance_responder <- prop.table(target_balance_responder) * 100
cat("Target variable balance (in percentage):\n") ## Target variable balance (in
percentage):
print(percentage_balance_responder) # 0=574034 is 75.4% and 1=187482 is 24.6%
```

```{r}
# Change the order of factor levels for the "goodbad_responder" variable
model_set_responder_final$goodbad_responder<-
relevel(model_set_responder_final$goodbad_responder, ref = "1") # Recheck which
factor level is associated with '1' and '0'
levels(model_set_responder_final$goodbad_responder)
```

```{r}
# Check number of levels for each variable in data frame
sapply(model_set_responder_final, function(x) length(unique(x)))
```

```{r}
# Check and modify class levels of the outcome variable
levels(model_set_responder_final$goodbad_responder) <-
make.names(levels(model_set_responder_final$goodbad_responder))
```

```{r}
# Count of target variable
table(model_set_responder_final$goodbad_responder)#  x1(1)= 187482 x0(0) =574034
prop.table(table(model_set_responder_final$goodbad_responder)) # x1 is 24.6% and x0
is 75.4%
```

```{r}
## Split the Data into a 80/20 Training/Testing Set ##
set.seed(123)
p_split_RF_responder <- rsample::initial_split(model_set_responder_final,prop=0.8)
train_data_responder <- rsample::training(p_split_RF_responder)
test_data_responder <- rsample::testing(p_split_RF_responder)
```

```{r}
# Count of target variable
table(train_data_responder$goodbad_responder)#  x1(1)= 149,858 x0(0) =459,354
prop.table(table(train_data_responder$goodbad_responder)) # x1 is 24.6% and x0 is
75.4%
```
```{r}
# Count of target variable
table(test_data_responder$goodbad_responder)#  x1(1)= 149,858 x0(0) =459,354
prop.table(table(test_data_responder$goodbad_responder)) # x1 is 24.6% and x0 is
75.4%
```

##ClassWEIGHT
MODEL##
#Class weights calculation for responder model
```{r}
# Calculate the proportion of the minority class (responders) to the majority class
(non-responders)
proportion_responders <- sum(train_data_responder$goodbad_responder == "X1") /
nrow(train_data_responder)
proportion_responders
proportion_non_responders <- 1 - proportion_responders
proportion_non_responders

# Assign class weights based on the proportions
weight_responders <- 1 / proportion_responders
weight_non_responders <- 1 / proportion_non_responders

# Create a vector of class weights
class_weights_responder <- ifelse(train_data_responder$goodbad_responder == "X1",
weight_responders, weight_non_responders)
class_weights_responder

```

```{r}
# List all variable names in the dataset
all_variables_responder <- colnames(train_data_responder)


# Remove the variable you don't want to use in the model
variables_to_include_responder <- all_variables_responder[!all_variables_responder
%in% c("goodbad_responder","bureau_seq_nbr")]
```

```
# Create a formula excluding the variable to ignore
model_formula_responder <- as.formula(paste("goodbad_responder ~",
paste(variables_to_include_responder, collapse = " + ")))
model_formula_responder

```


Random Forest Model for responder model with class weights
```{r}
#?ranger()
## Identify hyper-parameters
# 1.mtry (Randomly selected predictor)
# 2.splitrule
# 3.min.node size
# Specify values for hyper-parameters
# mtry --> generally start with around square-root of variables (56 parameters)
sqrt(56) # 7.4

my.mtry = c(2,3,5,7,8,9,10)
#splitrule
my.rule = "gini"
# min.node.size --> default is 1 for classification
my.nodes = c(1,3)
# create tuning grid

my.grid = expand.grid(mtry = my.mtry,
                      splitrule = my.rule,
                      min.node.size = my.nodes)
my.grid


#Select an appropriate evaluation metric (after oversampling dataset is imblance so
use "accuracy")
my.metric = "Kappa"

# Then, use the train function with the specified class weights
rf.tune_responder <- caret::train(model_formula_responder, data =
train_data_responder,
                        method = "ranger",
                        metric = my.metric,
                        importance = "impurity",
                        trControl = trainControl(classProbs = TRUE,
                                                 method = "cv", number = 5),
                        tuneGrid = my.grid,
                        weights = class_weights_responder)
rf.tune_responder

rf.tune_responder$results %>% arrange(desc(Kappa))
```

```
## Select best hyperparameters based on evaluation metric
rf.tune_responder$bestTune
```

```{r}
## Use tuned model to predict test sample
prob.tune_rf_responder = predict(rf.tune_responder, test_data_responder,type =
"prob")
head(prob.tune_rf_responder)
```

```{r}
# Predicted classes
predicted_classes_resp <- predict(rf.tune_responder, test_data_responder)
head(predicted_classes_resp)

# Combine predicted probabilities and classes into a single table
predictions_table_resp <- cbind(test_data_responder, Predicted_Class =
predicted_classes_resp, Predicted_Probabilities = prob.tune_rf_responder)

# View the combined table
print(predictions_table_resp)
```

```{r}
responder_prob<-predictions_table_resp %>%
  select(bureau_seq_nbr,goodbad_responder,Predicted_Class,
Predicted_Probabilities.X1,Predicted_Probabilities.X0)
head(responder_prob)
```

```{r}
# Save data as CSV file
write.csv(responder_prob,
"/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/respon
der_weight_prob.csv", row.names = FALSE)
```

```{r}
## Confusion Matrix for best performing random forest model using "test sample"
pred_responder = predict(rf.tune_responder,test_data_responder, type = "raw")
#ranger version of 'class'
conf_responder = caret::confusionMatrix(data = pred_responder, reference =
test_data_responder$goodbad_responder)
conf_responder
# TP FN
```

```
# FP TN
t(conf_responder$table)
```

```{r}
# Random forest model variable importance scores were based on improvement in gini
impurity
# Extract scaled variable importance scores
rf_importance_responder <- varImp(rf.tune_responder, scale = TRUE)
print(rf_importance_responder)
```

```{r}
# Create a data frame for Random Forest variable importance
rf_importance_responder_df <- data.frame( Variable =
rownames(rf_importance_responder$importance), Importance =
rf_importance_responder$importance$Overall)
```

```{r}
# Sort the data frame by Importance in descending order for a meaningful plot
rf_importance_responder_df <- rf_importance_responder_df %>%
  filter(Importance >= 10)

plot_responder<- rf_importance_responder_df %>%
  ggplot(aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "orange") +
  geom_text(aes(label = ifelse(Importance > 10, sprintf("%.1f", Importance), "")),
            size = 3, color = 'black', vjust = 2.5) +
  labs(title = "Random Forest Variable Importance of Responder Model (Gini
Impurity)",
       x = "Variable",
       y = "Importance")+
  scale_y_continuous(breaks = seq(0, 100, by = 10))+
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5,size=10),
        axis.text.x = element_text(angle = 45, hjust = 1))
plot_responder
```

```{r}
# ROCR package for ROCR package for Random forest model for test dataset
prediction_responder =
ROCR::prediction(prob.tune_rf_responder[,1],test_data_responder$goodbad_responder)
#first column of pred prob
```

```
class(prediction_responder)
prediction_responder # mainly used as an intermediate object
str(prediction_responder)

# Plot an ROC curve based on prediction on test
roc_responder = ROCR::performance(prediction_responder, "tpr", "fpr")
str(roc_responder) #measure, x.measure (y-axis is first --> measure)
plot(roc_responder)
```

```{r}
# Nicer plot for Random forest model
df.roc_responder = data.frame(fpr = as.vector(unlist(roc_responder@x.values)), tpr
= as.vector(unlist(roc_responder@y.values))) #head(df.roc_rf)

roc_plot_responder <-ggplot() +
  geom_line(data = df.roc_responder, aes(x = fpr, y = tpr, color = "Responder
Model"), linewidth = 1) +
  geom_abline(lty = "dashed", alpha = 0.5, color = "gray25", linewidth = 1.2) +
scale_color_manual(values = c("darkred", "darkblue")) +
  ggtitle("ROC: Random Forest Model for Responder Customer") +
  guides(color = guide_legend(title = element_blank())) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size=10), legend.text =
element_text(size = 8))
roc_plot_responder
```

```{r}
#AUC of ROC curve for Random Forest Model of test dataset
auc_responder = ROCR::performance(prediction_responder, measure = "auc")
str(auc_responder)
auc_responder@y.values %>% unlist()
```

```{r}
# Coordinates for the annotations
responder_annotation <- data.frame(x = 0.7, y = 0.5, label = "AUC: 0.523 ")
# Add annotations to the plot
roc_plot_responder +
  geom_text(data = responder_annotation, aes(x = x, y = y, label = label), color =
"darkred",fontface = "bold", size = 3)
```

Scoring dataset
```{r}
# read scoring dataset
responder_score_median_binned <-
read_csv("/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_fold
```

er/responder_score_median_binned.csv")
glimpse(responder_score_median_binned)
```

```{r}
## Use tuned model to predict scoring data
prob.tune_rf_responder_scoring = predict(rf.tune_responder,
responder_score_median_binned,type = "prob")
head(prob.tune_rf_responder_scoring)
```

```{r}
# Predicted classes
predicted_classes_resp_score <- predict(rf.tune_responder,
responder_score_median_binned)
head(predicted_classes_resp_score)

# Combine predicted probabilities and classes into a single table
predictions_table_resp_score <- cbind(responder_score_median_binned,
Predicted_Class = predicted_classes_resp_score, Predicted_Probabilities =
prob.tune_rf_responder_scoring)

# View the combined table
print(predictions_table_resp_score)
```

```{r}
responder_prob_score<-predictions_table_resp_score %>%
  select(seq,Predicted_Class,
Predicted_Probabilities.X1,Predicted_Probabilities.X0)
head(responder_prob_score)
```

```{r}
# Save data as CSV file
write.csv(responder_prob_score,
"/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/respon
der_weight_prob_score.csv", row.names = FALSE)
```

#Default Model

```{r}

```
change the target variable to factor
model_set_default_final["goodbad"] <- lapply(model_set_default_final["goodbad"],
as.factor)
glimpse(model_set_default_final)
```

```{r}
#Check which factor level is associated with 'default(delinquency) is 1' and
'nondefault(nondelinquency) is 0'
levels(model_set_default_final$goodbad)
```

```{r}
# Check how balanced the target variable is
target_balance_default <- table(model_set_default_final$goodbad)
print(target_balance_default)
percentage_balance_default <- prop.table(target_balance_default) * 100
cat("Target variable balance (in percentage):\n") ## Target variable balance (in
percentage):
print(percentage_balance_default) # 0=111188 is 59.3% and 1=76294 is 40.7%
```

we are looking for customer who will default so 1 should be positive.
```{r}
# Change the order of factor levels for the "goodbad_responder" variable
model_set_default_final$goodbad<- relevel(model_set_default_final$goodbad, ref =
"1") # Recheck which factor level is associated with '1' and '0'
levels(model_set_default_final$goodbad)
```

```{r}
# Check number of levels for each variable in data frame
sapply(model_set_default_final, function(x) length(unique(x)))
```

```{r}
# Check and modify class levels of the outcome variable
levels(model_set_default_final$goodbad) <-
make.names(levels(model_set_default_final$goodbad))
```

```{r}
# Count of target variable
table(model_set_default_final$goodbad)#  x1(1)= 76294 x0(0) =111188
prop.table(table(model_set_default_final$goodbad)) # x1 is 40.7% and x0 is 59.3%
```

```{r}
## Split the Data into a 80/20 Training/Testing Set ##
set.seed(123)
```

```
p_split_RF_default <- rsample::initial_split(model_set_default_final,prop=0.8)
train_data_default <- rsample::training(p_split_RF_default)
test_data_default <- rsample::testing(p_split_RF_default)
```

```{r}
# Count of target variable
table(train_data_default$goodbad)#   x0(0) =88870 x1(1)= 61115
prop.table(table(train_data_default$goodbad)) #  x0 is 59.3% and x1 is 40.7%
```

```{r}
# List all variable names in the dataset
all_variables_default <- colnames(train_data_default)


# Remove the variable you don't want to use in the model
variables_to_include_default <- all_variables_default[!all_variables_default %in%
c("goodbad","bureau_seq_nbr")]


# Create a formula excluding the variable to ignore
model_formula_default <- as.formula(paste("goodbad ~",
paste(variables_to_include_default, collapse = " + ")))
model_formula_default

```

Random Forest Model for default model without class weight
```{r}
#?ranger()
## Identify hyper-parameters
# 1.mtry (Randomly selected predictor)
# 2.splitrule
# 3.min.node size
# Specify values for hyper-parameters
# mtry --> generally start with around square-root of variables (47 parameters)
sqrt(47) # 6.8

my.mtry = c(2,3,5,7)
#splitrule
my.rule = "gini"
# min.node.size --> default is 1 for classification
my.nodes = c(1,3)
# create tuning grid

my.grid = expand.grid(mtry = my.mtry,
                      splitrule = my.rule,
```

```
                      min.node.size = my.nodes)
my.grid


#Select an appropriate evaluation metric
my.metric = "Accuracy"

#Train model over hyperparameters
set.seed(2)
rf.tune_default_noweight = caret::train(model_formula_default, data
=train_data_default,
                        method = "ranger",
                        metric = my.metric ,
                        importance = "impurity",
                        trControl = trainControl(classProbs =  T,method = "cv",
number = 5) ,
                        tuneGrid = my.grid) # grid of hyperparameters
rf.tune_default_noweight

rf.tune_default_noweight$results %>% arrange(desc(Accuracy))

## Select best hyperparameters based on evaluation metric
rf.tune_default_noweight$bestTune
```


#Prob for nonweighted
```{r}
## Use tuned model to predict test sample
prob.tune_rf_default_noweight = predict(rf.tune_default_noweight,
test_data_default,type = "prob")
head(prob.tune_rf_default_noweight)
```


```{r}
# Predicted classes
predicted_classes_default_noweight <- predict(rf.tune_default_noweight,
test_data_default)
head(predicted_classes_default_noweight)

# Combine predicted probabilities and classes into a single table
predictions_table_default_noweight <- cbind(test_data_default, Predicted_Class =
predicted_classes_default_noweight, Predicted_Probabilities =
prob.tune_rf_default_noweight)

# View the combined table
print(predictions_table_default_noweight)
```


```{r}
default_prob_noweight<-predictions_table_default_noweight %>%
```

```
  select(bureau_seq_nbr,goodbad,Predicted_Class,
Predicted_Probabilities.X0,Predicted_Probabilities.X1)
head(default_prob_noweight)
```

```{r}
# Save data as CSV file
write.csv(default_prob_noweight,
"/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/defaul
t_prob_noweight.csv", row.names = FALSE)
```

```{r}
## Confusion Matrix for best performing random forest model using "test sample"
pred_default_noweight = predict(rf.tune_default_noweight,test_data_default, type =
"raw") #ranger version of 'class'
conf_default_noweight = caret::confusionMatrix(data = pred_default_noweight,
reference = test_data_default$goodbad)
conf_default_noweight
# TP FN
# FP TN
t(conf_default_noweight$table)
```

```{r}
# Random forest model variable importance scores were based on improvement in gini
impurity
# Extract scaled variable importance scores
rf_importance_default_noweight <- varImp(rf.tune_default_noweight, scale = TRUE)
print(rf_importance_default_noweight)
```

```{r}
# Create a data frame for Random Forest variable importance
rf_importance_default_df_noweight <- data.frame( Variable =
rownames(rf_importance_default_noweight$importance), Importance =
rf_importance_default_noweight$importance$Overall)
```

```{r}
# Sort the data frame by Importance in descending order for a meaningful plot
# Filter the data frame to remove variables with Importance less than 10
rf_importance_default_df_noweight_filtered <- rf_importance_default_df_noweight %>%
```

```
  filter(Importance >= 10)

# Plot the filtered data
p_default1 <- rf_importance_default_df_noweight_filtered %>%
  ggplot(aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "orange") +
  geom_text(aes(label = sprintf("%.1f", Importance)), size = 3, color = 'black',
vjust = 2.5) +
  labs(title = "Random Forest Variable Importance of Default Model (Gini
Impurity)",
       x = "Variable",
       y = "Importance") +
  scale_y_continuous(breaks = seq(0, 100, by = 10)) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 10),
        axis.text.x = element_text(angle = 45, hjust = 1))

# Print the plot
print(p_default1)

```
```{r}
# ROCR package for ROCR package for Random forest model for test dataset
prediction_default =
ROCR::prediction(prob.tune_rf_default[,1],test_data_default$goodbad) #first column
of pred prob
class(prediction_default)
prediction_default # mainly used as an intermediate object
str(prediction_default)

# Plot an ROC curve based on prediction on test
roc_default = ROCR::performance(prediction_default, "tpr", "fpr")
str(roc_default) #measure, x.measure (y-axis is first --> measure)
plot(roc_default)
```

```{r}
# Nicer plot for Random forest model
df.roc_default = data.frame(fpr = as.vector(unlist(roc_default@x.values)), tpr =
as.vector(unlist(roc_default@y.values))) #head(df.roc_rf)

roc_plot_default <-ggplot() +
  geom_line(data = df.roc_default, aes(x = fpr, y = tpr, color = "Default Model"),
linewidth = 1) +
  geom_abline(lty = "dashed", alpha = 0.5, color = "gray25", linewidth = 1.2) +
scale_color_manual(values = c("darkred", "darkblue")) +
  ggtitle("ROC: Random Forest Model for NonDefault Customer") +
```

```
  guides(color = guide_legend(title = element_blank())) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size=10), legend.text =
element_text(size = 8))
roc_plot_default
```

```{r}
#AUC of ROC curve for Random Forest Model of test dataset
auc_default = ROCR::performance(prediction_default, measure = "auc")
str(auc_default)
auc_default@y.values %>% unlist()
```

```{r}
# Coordinates for the annotations
default_annotation <- data.frame(x = 0.7, y = 0.5, label = "AUC: 0.539 ")
# Add annotations to the plot
roc_plot_default +
  geom_text(data =default_annotation, aes(x = x, y = y, label = label), color =
"darkred",fontface = "bold", size = 3)
```

Scoring dataset
```{r}
# read scoring dataset
default_score_median_binned <-
read_csv("/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_fold
er/default_score_median_binned.csv")
glimpse(default_score_median_binned)
```

```{r}
## Use tuned model to predict scoring data
prob.tune_rf_default_scoring = predict(rf.tune_default_noweight,
default_score_median_binned,type = "prob")
head(prob.tune_rf_default_scoring)
```

```{r}
# Predicted classes
predicted_classes_default_score <- predict(rf.tune_default_noweight,
default_score_median_binned)
head(predicted_classes_default_score)
```

```
# Combine predicted probabilities and classes into a single table
predictions_table_default_score <- cbind(default_score_median_binned,
Predicted_Class = predicted_classes_default_score, Predicted_Probabilities =
prob.tune_rf_default_scoring)

# View the combined table
print(predictions_table_default_score)
```

```{r}
default_prob_score<-predictions_table_default_score %>%
  select(seq,Predicted_Class,
Predicted_Probabilities.X1,Predicted_Probabilities.X0)
head(default_prob_score)
```

```{r}
# Save data as CSV file
write.csv(default_prob_score,
"/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/defaul
t_prob_score.csv", row.names = FALSE)
```

SAS Code

```
/* Create permanent library*/
/* libname pnb
'/gpfs/user_home/os_home_dirs/bjone407/stat8330_spring24_team2/Shared_folder'; */

/* Prativa - Create permanent library */
/* libname pnb
'/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/STAT8330_Prativa';
*/

/* Nina - Create permanent library */
libname pnb
'/gpfs/user_home/os_home_dirs/ngrundli/stat8330_spring24_team2/Shared_folder';

/*KSU_DEV_DATA.csvdataset*/
FILENAME REFFILE
'/gpfs/user_home/os_home_dirs/ngrundli/ds7900_spring24_atlanticus/KSU_DE82_OOT_DATA
.csv';
PROC IMPORT DATAFILE=REFFILE
        DBMS=CSV
        OUT=pnb.KSU_DEV_OOT_DATA;
        GETNAMES=YES;
RUN;


proc contents data=pnb.ksu_dev_oot_data;
run;

/* set macro var to be able to call this on */
%let d1=pnb.ksu_dev_oot_data;

/* Brandi-atlanticus macro to recode missing vars */
/* %include
'/gpfs/user_home/os_home_dirs/bjone407/stat8330_spring24_team2/Shared_folder/atlant
icus_recode_macro.sas'; */

/* Prativa- atlanticus macro to recode missing vars */
/* %include
'/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/atlant
icus_recode_macro.sas';  */

/* Nina- atlanticus macro to recode missing vars */
%include
'/gpfs/user_home/os_home_dirs/ngrundli/stat8330_spring24_team2/Shared_folder/atlant
icus_recode_macro.sas';



proc contents data=&d1 noprint out=varlist(keep=name type);
run;
```

```
proc sql noprint;
  select name, type into :names separated by ' ',
              :lengths separated by ' '
  from varlist
  where name not in ('bureau_seq_nbr' 'grid24' 'goodbad' 'goodbad_responder'
'seq');
quit;


/* recode missing observations */
data pnb.recoded_all;
  set &d1;
  %recode_all;
run;

data pnb.recoded_all2;
set pnb.recoded_all;
run;

/* remove rows with missing values. First create nmiss variable */
data pnb.recoded_all2;
set pnb.recoded_all2;
if nmiss(of ams3000--ams3995)<0.3 then delete;
nmiss = nmiss(of ams3000--ams3995);
run;

/* remove rows with more than 50% missing values */
data pnb.recoded_all3;
set pnb.recoded_all2;
if nmiss > 538*0.5 then delete;
run;

proc contents data=pnb.recoded_all3;
run;

/* Brandi - drop vars with certain percentage missing */
/* %include
"/gpfs/user_home/os_home_dirs/bjone407/stat8330_spring24_team2/Shared_folder/drop_v
ars.sas"; */

/*Prativa - Drops vars with certain percentage missing*/
/* %include
"/gpfs/user_home/os_home_dirs/pbasnet2/stat8330_spring24_team2/Shared_folder/drop_v
ars.sas";  */

/*Nina - Drops vars with certain percentage missing*/
%include
"/gpfs/user_home/os_home_dirs/ngrundli/stat8330_spring24_team2/Shared_folder/drop_v
ars.sas";
```

```
/*Use get_missing_vars */
%get_missing_vars(lib=pnb, dsn=recoded_all3, threshold=0.3, exclude_vars=seq
bureau_seq_nbr grid24 goodbad goodbad_responder);

%put &drop_vars;

%let var_list = &drop_vars.;
/* %put &var_list; */

/* need to add back goodbad_responder and grid24 that were removed in drop_vars
macro */
%macro remove_vars(var_list, vars_to_remove);
    %local i j var_list_new remove_var;
    %let var_list_new = &var_list;
    %do j = 1 %to %sysfunc(countw(&vars_to_remove));
        %let remove_var = %scan(&vars_to_remove, &j);
        %let i = 1;
        %do %while (%scan(&var_list_new, &i) ne %str() and %scan(&var_list_new, &i)
ne &remove_var);
            %let i = %eval(&i + 1);
        %end;
        %if %scan(&var_list_new, &i) eq &remove_var %then %do;
            %let var_list_new = %sysfunc(tranwrd(&var_list_new, %str( )&remove_var,
%str()));
                %let i = 1;
        %end;
        %else %do;
            %let i = %eval(&i + 1);
        %end;
    %end;
    &var_list_new
%mend;

/* removing grid24 from var_list - this DOES NOT work */
%let updated_var_list = %remove_vars(&var_list, grid24);
%put &updated_var_list;

/* removing goodbad_responder - this DOES work */
%let updated_var_list2 = %remove_vars(&updated_var_list, goodbad_responder);
%put &updated_var_list2;

data pnb.dropped_vars;
set pnb.recoded_all3;
/* drop &drop_vars; */
drop &updated_var_list2;
run;


data pnb.dropped_vars2;
set pnb.dropped_vars;
```

```
run;

proc contents data=pnb.dropped_vars2 out=varlist(keep=name);
run;


proc sql noprint;
  select name
  into :varlist separated by ' '
  from varlist
  where name not in ('goodbad');
quit;


/* check the number of missing per variable after dropping */
/*  proc freq data=pnb.dropped_vars  ;  */
/*  tables _all_; */
/*  run;  */

/* now do median imputation on remaining vars */
proc stdize data=pnb.dropped_vars2 out=pnb.imputed
/*       oprefix=Orig_          /* prefix for original variables */
      reponly               /* only replace; do not standardize */
      method=MEDIAN;          /* or MEDIAN, MINIMUM, MIDRANGE, etc. */
   var &varlist.;              /* you can list multiple variables to impute */
run;

proc means data = pnb.imputed nmiss n ;
run;

/*
################################################################################################
############## */
/*Create dataset for responder and nonresponder model*/
data pnb.responder (drop=nmiss goodbad);
set pnb.imputed;
run;

/* selecting the variables from the variable clustering results */
/*Variable list after doing hierarchical clustering (56 independent vars) */
data pnb.responder_final_score_model (keep= seq ams3133 ams3277 ams3266 ams3224
ams3246 ams3193 ams3136 ams3008 ams3101 ams3273
 ams3177 ams3297 ams3316 ams3352 ams3358 ams3296 ams3327 ams3360 ams3350 ams3364
ams3359 ams3546 ams3555 ams3544
 ams3537 ams3557 ams3562 ams3420 ams3365 ams3423 ams3477 ams3547 ams3380 ams3634
ams3583 ams3630 ams3577 ams3633
 ams3756 ams3612 ams3569 ams3593 ams3757 ams3579 ams3755 ams3963 ams3960 ams3796
ams3909 ams3904 ams3806 ams3860
 ams3902 ams3905 ams3953 ams3862);
set pnb.responder;
```

```
run;
proc print data=pnb.responder_final_score_model(obs=10);run;


/*Create dataset for default and nondefault model "default"*/
data pnb.default (drop=nmiss goodbad_responder);
set pnb.imputed;
/* if goodbad = . then delete; */
run;


/*Variable list after doing hierarchical clustering (47independent vars) */
data pnb.default_final_score_model (keep=seq ams3277 ams3133 ams3224 ams3296
ams3193 ams3136 ams3288 ams3290 ams3008 ams3257
ams3101 ams3293 ams3316 ams3358 ams3359 ams3356 ams3423 ams3385 ams3383 ams3380
ams3327 ams3364 ams3350 ams3546 ams3555
ams3581 ams3590 ams3577 ams3609 ams3535 ams3544 ams3547 ams3536 ams3558 ams3573
ams3611 ams3964 ams3960 ams3758 ams3796
ams3909 ams3860 ams3904 ams3902 ams3627 ams3633 ams3757);
set pnb.default;
run;
proc print data=pnb.default_final_score_model (obs=10);run;


/* HARD CODE BINNING */
data pnb.responder_score_binned (drop= ams3008 ams3101 ams3133 ams3136 ams3177
ams3193 ams3224 ams3246 ams3257 ams3266 ams3273
ams3277 ams3288 ams3290 ams3293 ams3296 ams3297 ams3316 ams3327 ams3350 ams3352
ams3356 ams3358 ams3359 ams3360 ams3364
ams3365 ams3380 ams3383 ams3385 ams3420 ams3423 ams3477 ams3535 ams3536 ams3537
ams3544 ams3546 ams3547 ams3555 ams3557
ams3558 ams3562 ams3569 ams3573 ams3577 ams3579 ams3581 ams3583 ams3590 ams3593
ams3609 ams3611 ams3612 ams3627 ams3630
ams3633 ams3634 ams3755 ams3756 ams3757 ams3758 ams3796 ams3806 ams3860 ams3862
ams3902 ams3904 ams3905 ams3909 ams3953
ams3960 ams3963 ams3964);
set pnb.responder_final_score_model;
if ams3008<0.2 then BIN_ams3008=1;
else if ams3008<1.02 then BIN_ams3008=2;
else if ams3008>=1.02 then BIN_ams3008=3;
if ams3101<0.2 then BIN_ams3101=1;
else if ams3101<1.02 then BIN_ams3101=2;
else if ams3101>=1.02 then BIN_ams3101=3;
if ams3133<0.2 then BIN_ams3133=1;
else if ams3133<1.02 then BIN_ams3133=2;
else if ams3133>=1.02 then BIN_ams3133=3;
if ams3136<0.2 then BIN_ams3136=1;
else if ams3136<1.02 then BIN_ams3136=2;
else if ams3136>=1.02 then BIN_ams3136=3;
if ams3177<0.2 then BIN_ams3177=1;
```

```
else if ams3177<1.02 then BIN_ams3177=2;
else if ams3177>=1.02 then BIN_ams3177=3;
if ams3193<0.2 then BIN_ams3193=1;
else if ams3193<1.02 then BIN_ams3193=2;
else if ams3193>=1.02 then BIN_ams3193=3;
if ams3224<0.2 then BIN_ams3224=1;
else if ams3224<1.02 then BIN_ams3224=2;
else if ams3224>=1.02 then BIN_ams3224=3;
if ams3246<0.2 then BIN_ams3246=1;
else if ams3246>=0.2 then BIN_ams3246=1;
/* if ams3257<0.2 then BIN_ams3257=1; */
/* else if ams3257>=0.2 then BIN_ams3257=2; */
if ams3266<0.2 then BIN_ams3266=1;
else if ams3266>=0.2 then BIN_ams3266=2;
if ams3273<0.2 then BIN_ams3273=1;
else if ams3273>=0.2 then BIN_ams3273=2;
if ams3277<0.2 then BIN_ams3277=1;
else if ams3277<1.02 then BIN_ams3277=2;
else if ams3277>=1.02 then BIN_ams3277=3;
/* if ams3288<0.2 then BIN_ams3288=1; */
/* else if ams3288<1.02 then BIN_ams3288=2; */
/* else if ams3288>=1.02 then BIN_ams3288=3; */
/* if ams3290<0.2 then BIN_ams3290=1; */
/* else if ams3290>=0.2 then BIN_ams3290=2; */
/* if ams3293<0.2 then BIN_ams3293=1; */
/* else if ams3293>=0.2 then BIN_ams3293=2; */
if ams3296<0.2 then BIN_ams3296=1;
else if ams3296>=0.2 then BIN_ams3296=2;
if ams3297<0.2 then BIN_ams3297=1;
else if ams3297>=0.2 then BIN_ams3297=2;
if ams3316<0.2 then BIN_ams3316=1;
else if ams3316>=0.2 then BIN_ams3316=2;
if ams3327<0.2 then BIN_ams3327=1;
else if ams3327>=0.2 then BIN_ams3327=2;
if ams3350<0.2 then BIN_ams3350=1;
else if ams3350>=0.2 then BIN_ams3350=2;
if ams3352<0.2 then BIN_ams3352=1;
else if ams3352>=0.2 then BIN_ams3352=2;
/* if ams3356<0.2 then BIN_ams3356=1; */
/* else if ams3356>=0.2 then BIN_ams3356=1; */
if ams3358<0.2 then BIN_ams3358=1;
else if ams3358>0.2 then BIN_ams3358=2;
if ams3359<0.2 then BIN_ams3359=1;
else if ams3359>=0.2 then BIN_ams3359=1;
if ams3360<0.2 then BIN_ams3360=1;
else if ams3360>0.2 then BIN_ams3360=2;
if ams3364<0.2 then BIN_ams3364=1;
else if ams3364>=0.2 then BIN_ams3364=1;
if ams3365<0.2 then BIN_ams3365=1;
else if ams3365>=0.2 then BIN_ams3365=1;
```

```
if ams3380<0.2 then BIN_ams3380=1;
else if ams3380>=0.2 then BIN_ams3380=1;
/* if ams3383<0.2 then BIN_ams3383=1; */
/* else if ams3383>=0.2 then BIN_ams3383=1; */
/* if ams3385<0.2 then BIN_ams3385=1; */
/* else if ams3385>=0.2 then BIN_ams3385=1; */
if ams3420<0.2 then BIN_ams3420=1;
else if ams3420>0.2 then BIN_ams3420=2;
if ams3423<2 then BIN_ams3423=1;
else if ams3423>=0.2 then BIN_ams3423=1;
if ams3477<0.2 then BIN_ams3477=1;
else if ams3477>=0.2 then BIN_ams3477=1;
/* if ams3535<0.2 then BIN_ams3535=1; */
/* else if ams3535<1.02 then BIN_ams3535=2; */
/* else if ams3535>=1.02 then BIN_ams3535=3; */
/* if ams3536<0.2 then BIN_ams3536=1; */
/* else if ams3536>=0.2 then BIN_ams3536=2; */
if ams3537<0.2 then BIN_ams3537=1;
else if ams3537>=0.2 then BIN_ams3537=2;
if ams3544<0.2 then BIN_ams3544=1;
else if ams3544<1.02 then BIN_ams3544=2;
else if ams3544>=1.02 then BIN_ams3544=3;
if ams3546<0.2 then BIN_ams3546=1;
else if ams3546>=0.2 then BIN_ams3546=2;
if ams3547<0.2 then BIN_ams3547=1;
else if ams3547>=0.2 then BIN_ams3547=2;
if ams3555<0.2 then BIN_ams3555=1;
else if ams3555>=0.2 then BIN_ams3555=2;
if ams3557<0.2 then BIN_ams3557=1;
else if ams3557>=0.2 then BIN_ams3557=2;
/* if ams3558<2 then BIN_ams3558=1; */
/* else if ams3558>=0.2 then BIN_ams3558=1; */
if ams3562<0.2 then BIN_ams3562=1;
else if ams3562>=0.2 then BIN_ams3562=2;
if ams3569<2 then BIN_ams3569=1;
else if ams3569>=0.2 then BIN_ams3569=1;
/* if ams3573<0.2 then BIN_ams3573=1; */
/* else if ams3573>=0.2 then BIN_ams3573=2; */
if ams3577<0.2 then BIN_ams3577=1;
else if ams3577>=0.2 then BIN_ams3577=2;
if ams3579<0.2 then BIN_ams3579=1;
else if ams3579>=0.2 then BIN_ams3579=2;
/* if ams3581<0.2 then BIN_ams3581=1; */
/* else if ams3581<1.02 then BIN_ams3581=2; */
/* else if ams3581>=1.02 then BIN_ams3581=3; */
if ams3583<0.2 then BIN_ams3583=1;
else if ams3583>=0.2 then BIN_ams3583=2;
/* if ams3590<0.2 then BIN_ams3590=1; */
/* else if ams3590<1.02 then BIN_ams3590=2; */
/* else if ams3590>=1.02 then BIN_ams3590=3; */
```

```
if ams3593<0.2 then BIN_ams3593=1;
else if ams3593>=0.2 then BIN_ams3593=2;
/* if ams3609<0.2 then BIN_ams3609=1; */
/* else if ams3609>=0.2 then BIN_ams3609=2; */
/* if ams3611<0.2 then BIN_ams3611=1; */
/* else if ams3611<1.02 then BIN_ams3611=2; */
/* else if ams3611>=1.02 then BIN_ams3611=3; */
if ams3612<0.2 then BIN_ams3612=1;
else if ams3612<1.02 then BIN_ams3612=2;
else if ams3612>=1.02 then BIN_ams3612=3;
/* if ams3627<0.2 then BIN_ams3627=1; */
/* else if ams3627<1.02 then BIN_ams3627=2; */
/* else if ams3627>=1.02 then BIN_ams3627=3; */
if ams3630<0.2 then BIN_ams3630=1;
else if ams3630<1.02 then BIN_ams3630=2;
else if ams3630>=1.02 then BIN_ams3630=3;
if ams3633<0.2 then BIN_ams3633=1;
else if ams3633<1.02 then BIN_ams3633=2;
else if ams3633>=1.02 then BIN_ams3633=3;
if ams3634<0.2 then BIN_ams3634=1;
else if ams3634<1.02 then BIN_ams3634=2;
else if ams3634>=1.02 then BIN_ams3634=3;
if ams3755<0.2 then BIN_ams3755=1;
else if ams3755>=0.2 then BIN_ams3755=2;
if ams3756<0.2 then BIN_ams3756=1;
else if ams3756>=0.2 then BIN_ams3756=2;
if ams3757<0.2 then BIN_ams3757=1;
else if ams3757>=0.2 then BIN_ams3757=2;
/* if ams3758<0.2 then BIN_ams3758=1; */
/* else if ams3758>=0.2 then BIN_ams3758=2; */
if ams3796<0.2 then BIN_ams3796=1;
else if ams3796<1.02 then BIN_ams3796=2;
else if ams3796>=1.02 then BIN_ams3796=3;
if ams3806<2 then BIN_ams3806=1;
else if ams3806>=0.2 then BIN_ams3806=1;
if ams3860<2 then BIN_ams3860=1;
else if ams3860>=0.2 then BIN_ams3860=1;
if ams3862<2 then BIN_ams3862=1;
else if ams3862>=0.2 then BIN_ams3862=1;
if ams3902<0.01 then BIN_ams3902=1;
else if ams3902>=0.01 then BIN_ams3902=2;
if ams3904<2 then BIN_ams3904=1;
else if ams3904>=0.2 then BIN_ams3904=1;
if ams3905<2 then BIN_ams3905=1;
else if ams3905>=0.2 then BIN_ams3905=1;
if ams3909<0.2 then BIN_ams3909=1;
else if ams3909<1.02 then BIN_ams3909=2;
else if ams3909>=1.02 then BIN_ams3909=3;
if ams3953<0.2 then BIN_ams3953=1;
else if ams3953<1.02 then BIN_ams3953=2;
```

```
else if ams3953>=1.02 then BIN_ams3953=3;
if ams3960<0.2 then BIN_ams3960=1;
else if ams3960<1.02 then BIN_ams3960=2;
else if ams3960>=1.02 then BIN_ams3960=3;
if ams3963<0.2 then BIN_ams3963=1;
else if ams3963<1.02 then BIN_ams3963=2;
else if ams3963>=1.02 then BIN_ams3963=3;
/* if ams3964<0.2 then BIN_ams3964=1; */
/* else if ams3964<1.02 then BIN_ams3964=2; */
/* else if ams3964>=1.02 then BIN_ams3964=3; */
run;

/* save as csv file */
proc export data=pnb.responder_score_binned
outfile='/gpfs/user_home/os_home_dirs/ngrundli/stat8330_spring24_team2/Shared_folde
r/responder_score_median_binned.csv';
run;


data pnb.default_score_binned (drop= ams3008 ams3101 ams3133 ams3136 ams3177
ams3193 ams3224 ams3246 ams3257 ams3266 ams3273
ams3277 ams3288 ams3290 ams3293 ams3296 ams3297 ams3316 ams3327 ams3350 ams3352
ams3356 ams3358 ams3359 ams3360 ams3364
ams3365 ams3380 ams3383 ams3385 ams3420 ams3423 ams3477 ams3535 ams3536 ams3537
ams3544 ams3546 ams3547 ams3555 ams3557
ams3558 ams3562 ams3569 ams3573 ams3577 ams3579 ams3581 ams3583 ams3590 ams3593
ams3609 ams3611 ams3612 ams3627 ams3630
ams3633 ams3634 ams3755 ams3756 ams3757 ams3758 ams3796 ams3806 ams3860 ams3862
ams3902 ams3904 ams3905 ams3909 ams3953
ams3960 ams3963 ams3964);
set pnb.default_final_score_model;
if ams3008<0.2 then BIN_ams3008=1;
else if ams3008<1.02 then BIN_ams3008=2;
else if ams3008>=1.02 then BIN_ams3008=3;
if ams3101<0.2 then BIN_ams3101=1;
else if ams3101<1.02 then BIN_ams3101=2;
else if ams3101>=1.02 then BIN_ams3101=3;
if ams3133<0.2 then BIN_ams3133=1;
else if ams3133<1.02 then BIN_ams3133=2;
else if ams3133>=1.02 then BIN_ams3133=3;
if ams3136<0.2 then BIN_ams3136=1;
else if ams3136<1.02 then BIN_ams3136=2;
else if ams3136>=1.02 then BIN_ams3136=3;
/* if ams3177<0.2 then BIN_ams3177=1; */
/* else if ams3177<1.02 then BIN_ams3177=2; */
/* else if ams3177>=1.02 then BIN_ams3177=3; */
if ams3193<0.2 then BIN_ams3193=1;
else if ams3193<1.02 then BIN_ams3193=2;
else if ams3193>=1.02 then BIN_ams3193=3;
if ams3224<0.2 then BIN_ams3224=1;
```

```
else if ams3224<1.02 then BIN_ams3224=2;
else if ams3224>=1.02 then BIN_ams3224=3;
/* if ams3246<0.2 then BIN_ams3246=1; */
/* else if ams3246>=0.2 then BIN_ams3246=1; */
if ams3257<0.2 then BIN_ams3257=1;
else if ams3257>=0.2 then BIN_ams3257=2;
/* if ams3266<0.2 then BIN_ams3266=1; */
/* else if ams3266>=0.2 then BIN_ams3266=2; */
/* if ams3273<0.2 then BIN_ams3273=1; */
/* else if ams3273>=0.2 then BIN_ams3273=2; */
if ams3277<0.2 then BIN_ams3277=1;
else if ams3277<1.02 then BIN_ams3277=2;
else if ams3277>=1.02 then BIN_ams3277=3;
if ams3288<0.2 then BIN_ams3288=1;
else if ams3288<1.02 then BIN_ams3288=2;
else if ams3288>=1.02 then BIN_ams3288=3;
if ams3290<0.2 then BIN_ams3290=1;
else if ams3290>=0.2 then BIN_ams3290=2;
if ams3293<0.2 then BIN_ams3293=1;
else if ams3293>=0.2 then BIN_ams3293=2;
if ams3296<0.2 then BIN_ams3296=1;
else if ams3296>=0.2 then BIN_ams3296=2;
/* if ams3297<0.2 then BIN_ams3297=1; */
/* else if ams3297>=0.2 then BIN_ams3297=2; */
if ams3316<0.2 then BIN_ams3316=1;
else if ams3316>=0.2 then BIN_ams3316=2;
if ams3327<0.2 then BIN_ams3327=1;
else if ams3327>=0.2 then BIN_ams3327=2;
if ams3350<0.2 then BIN_ams3350=1;
else if ams3350>=0.2 then BIN_ams3350=2;
/* if ams3352<0.2 then BIN_ams3352=1; */
/* else if ams3352>=0.2 then BIN_ams3352=2; */
if ams3356<0.2 then BIN_ams3356=1;
else if ams3356>=0.2 then BIN_ams3356=1;
if ams3358<0.2 then BIN_ams3358=1;
else if ams3358>0.2 then BIN_ams3358=2;
if ams3359<0.2 then BIN_ams3359=1;
else if ams3359>=0.2 then BIN_ams3359=1;
/* if ams3360<0.2 then BIN_ams3360=1; */
/* else if ams3360>0.2 then BIN_ams3360=2; */
if ams3364<0.2 then BIN_ams3364=1;
else if ams3364>=0.2 then BIN_ams3364=1;
/* if ams3365<0.2 then BIN_ams3365=1; */
/* else if ams3365>=0.2 then BIN_ams3365=1; */
if ams3380<0.2 then BIN_ams3380=1;
else if ams3380>=0.2 then BIN_ams3380=1;
if ams3383<0.2 then BIN_ams3383=1;
else if ams3383>=0.2 then BIN_ams3383=1;
if ams3385<0.2 then BIN_ams3385=1;
else if ams3385>=0.2 then BIN_ams3385=1;
```

```
/* if ams3420<0.2 then BIN_ams3420=1; */
/* else if ams3420>0.2 then BIN_ams3420=2; */
if ams3423<2 then BIN_ams3423=1;
else if ams3423>=0.2 then BIN_ams3423=1;
/* if ams3477<0.2 then BIN_ams3477=1; */
/* else if ams3477>=0.2 then BIN_ams3477=1; */
if ams3535<0.2 then BIN_ams3535=1;
else if ams3535<1.02 then BIN_ams3535=2;
else if ams3535>=1.02 then BIN_ams3535=3;
if ams3536<0.2 then BIN_ams3536=1;
else if ams3536>=0.2 then BIN_ams3536=2;
/* if ams3537<0.2 then BIN_ams3537=1; */
/* else if ams3537>=0.2 then BIN_ams3537=2; */
if ams3544<0.2 then BIN_ams3544=1;
else if ams3544<1.02 then BIN_ams3544=2;
else if ams3544>=1.02 then BIN_ams3544=3;
if ams3546<0.2 then BIN_ams3546=1;
else if ams3546>=0.2 then BIN_ams3546=2;
if ams3547<0.2 then BIN_ams3547=1;
else if ams3547>=0.2 then BIN_ams3547=2;
if ams3555<0.2 then BIN_ams3555=1;
else if ams3555>=0.2 then BIN_ams3555=2;
/* if ams3557<0.2 then BIN_ams3557=1; */
/* else if ams3557>=0.2 then BIN_ams3557=2; */
if ams3558<2 then BIN_ams3558=1;
else if ams3558>=0.2 then BIN_ams3558=1;
/* if ams3562<0.2 then BIN_ams3562=1; */
/* else if ams3562>=0.2 then BIN_ams3562=2; */
/* if ams3569<2 then BIN_ams3569=1; */
/* else if ams3569>=0.2 then BIN_ams3569=1; */
if ams3573<0.2 then BIN_ams3573=1;
else if ams3573>=0.2 then BIN_ams3573=2;
if ams3577<0.2 then BIN_ams3577=1;
else if ams3577>=0.2 then BIN_ams3577=2;
/* if ams3579<0.2 then BIN_ams3579=1; */
/* else if ams3579>=0.2 then BIN_ams3579=2; */
if ams3581<0.2 then BIN_ams3581=1;
else if ams3581<1.02 then BIN_ams3581=2;
else if ams3581>=1.02 then BIN_ams3581=3;
/* if ams3583<0.2 then BIN_ams3583=1; */
/* else if ams3583>=0.2 then BIN_ams3583=2; */
if ams3590<0.2 then BIN_ams3590=1;
else if ams3590<1.02 then BIN_ams3590=2;
else if ams3590>=1.02 then BIN_ams3590=3;
/* if ams3593<0.2 then BIN_ams3593=1; */
/* else if ams3593>=0.2 then BIN_ams3593=2; */
if ams3609<0.2 then BIN_ams3609=1;
else if ams3609>=0.2 then BIN_ams3609=2;
if ams3611<0.2 then BIN_ams3611=1;
else if ams3611<1.02 then BIN_ams3611=2;
```

```
else if ams3611>=1.02 then BIN_ams3611=3;
/* if ams3612<0.2 then BIN_ams3612=1; */
/* else if ams3612<1.02 then BIN_ams3612=2; */
/* else if ams3612>=1.02 then BIN_ams3612=3; */
if ams3627<0.2 then BIN_ams3627=1;
else if ams3627<1.02 then BIN_ams3627=2;
else if ams3627>=1.02 then BIN_ams3627=3;
/* if ams3630<0.2 then BIN_ams3630=1; */
/* else if ams3630<1.02 then BIN_ams3630=2; */
/* else if ams3630>=1.02 then BIN_ams3630=3; */
if ams3633<0.2 then BIN_ams3633=1;
else if ams3633<1.02 then BIN_ams3633=2;
else if ams3633>=1.02 then BIN_ams3633=3;
/* if ams3634<0.2 then BIN_ams3634=1; */
/* else if ams3634<1.02 then BIN_ams3634=2; */
/* else if ams3634>=1.02 then BIN_ams3634=3; */
/* if ams3755<0.2 then BIN_ams3755=1; */
/* else if ams3755>=0.2 then BIN_ams3755=2; */
/* if ams3756<0.2 then BIN_ams3756=1; */
/* else if ams3756>=0.2 then BIN_ams3756=2; */
if ams3757<0.2 then BIN_ams3757=1;
else if ams3757>=0.2 then BIN_ams3757=2;
if ams3758<0.2 then BIN_ams3758=1;
else if ams3758>=0.2 then BIN_ams3758=2;
if ams3796<0.2 then BIN_ams3796=1;
else if ams3796<1.02 then BIN_ams3796=2;
else if ams3796>=1.02 then BIN_ams3796=3;
/* if ams3806<2 then BIN_ams3806=1; */
/* else if ams3806>=0.2 then BIN_ams3806=1; */
if ams3860<2 then BIN_ams3860=1;
else if ams3860>=0.2 then BIN_ams3860=1;
/* if ams3862<2 then BIN_ams3862=1; */
/* else if ams3862>=0.2 then BIN_ams3862=1; */
if ams3902<0.01 then BIN_ams3902=1;
else if ams3902>=0.01 then BIN_ams3902=2;
if ams3904<2 then BIN_ams3904=1;
else if ams3904>=0.2 then BIN_ams3904=1;
/* if ams3905<2 then BIN_ams3905=1; */
/* else if ams3905>=0.2 then BIN_ams3905=1; */
if ams3909<0.2 then BIN_ams3909=1;
else if ams3909<1.02 then BIN_ams3909=2;
else if ams3909>=1.02 then BIN_ams3909=3;
/* if ams3953<0.2 then BIN_ams3953=1; */
/* else if ams3953<1.02 then BIN_ams3953=2; */
/* else if ams3953>=1.02 then BIN_ams3953=3; */
if ams3960<0.2 then BIN_ams3960=1;
else if ams3960<1.02 then BIN_ams3960=2;
else if ams3960>=1.02 then BIN_ams3960=3;
/* if ams3963<0.2 then BIN_ams3963=1; */
/* else if ams3963<1.02 then BIN_ams3963=2; */
```

```
/* else if ams3963>=1.02 then BIN_ams3963=3; */
if ams3964<0.2 then BIN_ams3964=1;
else if ams3964<1.02 then BIN_ams3964=2;
else if ams3964>=1.02 then BIN_ams3964=3;
run;

/* save as csv file */
proc export data=pnb.default_score_binned
outfile='/gpfs/user_home/os_home_dirs/ngrundli/stat8330_spring24_team2/Shared_folde
r/default_score_median_binned.csv';
run;
```

Atlanticus Competition

STAT8330: Applied Binary Classification

Team PNB

Prativa Basnet

Nina Grundlingh

Brandi Jones

Atlanticus®

KENNESAW STATE UNIVERSITY

Our Presentation:

- Introduction
  - The Problem
  - The Data
- Data Cleaning/Processing
- Variable Selection Methodology
- Modeling
  - Approach
  - Model Comparisons
  - Final Model Evaluation Metrics
- Conclusions

Atlanticus®

KENNESAW STATE UNIVERSITY

Introduction

- Binary classification is commonly used in FinTech to address a variety of industry problems:
  - Financial fraud claims
  - Credit and investment risks
  - Bankruptcy risks
- Traditional classification methods include logistic regression, decision trees, etc.

Atlanticus®

KENNESAW STATE UNIVERSITY

The Problem

- This project focuses on data related to direct mail credit campaigns.
- The goal of our project is to predict whether individuals will respond to a credit offer mailer and whether, when issued credit, they will default.
- Want to determine who to send mail offers to so profit is maximized, and loss is minimized.

Atlanticus®

KENNESAW STATE UNIVERSITY

Data

- Atlanticus credit data:
  - Equifax data sourced including trade line performance on accounts booked outside of Atlanticus
- Time Periods: June 2021, July 2021
- 541 variables
- 988,267 observations
- One dependent variable: ‘goodbad’

goodbad	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	743198	75.20	743198	75.20
0	153729	15.56	896927	90.76
1	91340	9.24	988267	100.00

Atlanticus®

KENNESAW STATE UNIVERSITY

Data Preparation

Imputation

- Transforming coded missing values
- Remove variables with >30% missing
- Remove observations with >50% missing
  - In primary approach only
- Median imputation
  - In primary approach only
- MICE imputation
  - In secondary approach only

Atlanticus®

KENNESAW STATE UNIVERSITY

Data Preparation

Feature engineering

- Creation of new binary response variables: ‘goodbad_responder’, ‘goodbad_default’
- Created separate datasets for each response
- Remove variables with VIF>10

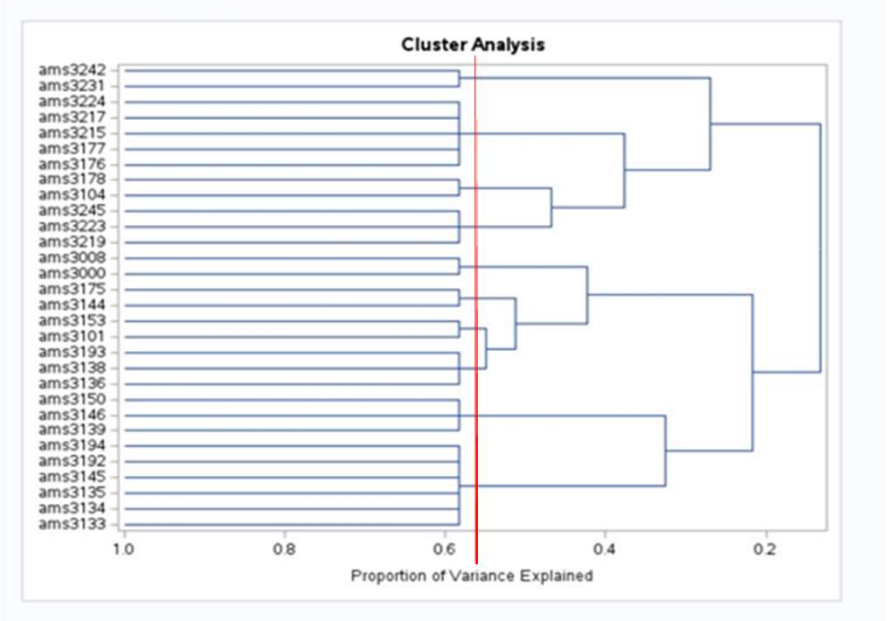
Atlanticus®

KENNESAW STATE UNIVERSITY

Data Preparation

Feature engineering

- Variable clustering
  - Utilizing principal components
  - Variable most representative of each cluster chosen
- Tree-based binning to discretize variables



Atlanticus®

KENNESAW STATE UNIVERSITY

Data Preparation

Table 1: Datasets used for modelling

Model	Number of observations	Number of variables
Response	761,516	57
Default	187,842	48

Train/Test Split, Validation

- Random Forest/Logistic Model - 80/20 split
- XGBoost Model - 60:20:20 split

Atlanticus®

KENNESAW STATE UNIVERSITY

Modeling: Approaches

- Response and default modelled separately.
- Four models considered:
  - Logistic regression
  - Unweighted random forest
  - Weighted random forest
  - XGBoost

Atlanticus®

KENNESAW STATE UNIVERSITY

Response Model Class Weight

Table 2: Weight calculation for Response Model

	Proportion	Weight Calculation
Response	0.246	1/proportion = 4.065
Nonresponse	0.754	1/proportion = 1.325

Atlanticus

KENNESAW STATE UNIVERSITY

Response Model Performance

Table 3a: Model comparison for Response Model

	F1	ACCURACY	SPECIFICITY	SENSITIVITY	PRECISION
RESPONDER RANDOM FOREST (Wt.)	0.284	0.610	0.707	0.3140	0.260
RESPONDER RANDOM FOREST (No Wt.)	0.038	0.750	0.989	0. 020	0.386
RESPONDER LOGISTIC REGRESSION	0.369	0.507	0.281	0.727	0.247

Atlanticus®

KENNESAW STATE UNIVERSITY

Default Model Performance

Table 3b: Model comparison for Default Model

	F1	ACCURACY	SPECIFICITY	SENSITIVITY	PRECISION
DEFAULT RANDOM FOREST (Wt.)	0.637	0.554	0.399	0.660	0.617
DEFAULT RANDOM FOREST (No Wt.)	0.041	0.597	0.988	0.021	0.597
DEFAULT LOGISTIC REGRESSION	0.137	0.538	0.936	0.080	0.473

Atlanticus®

KENNESAW STATE UNIVERSITY

XGBoost Model Approach

- Following variable selection for Response and Default:
  - Columns with 30% Missing Dropped
  - Variables selected from clustering/binning in primary approach
  - MICE Imputation
  - 60:20:20 train/validation/test split
  - Response and Default models run separately
  - Bayesian Optimization/Tuning

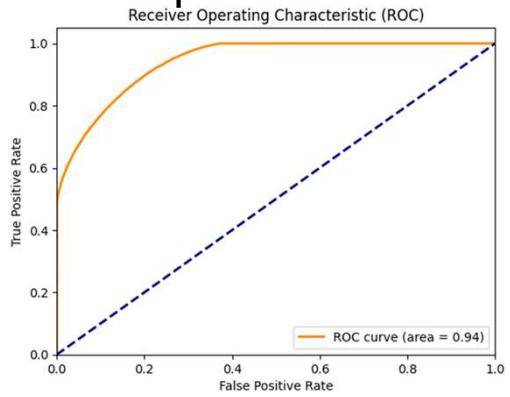
KENNESAW STATE UNIVERSITY

XGBoost Model Performance

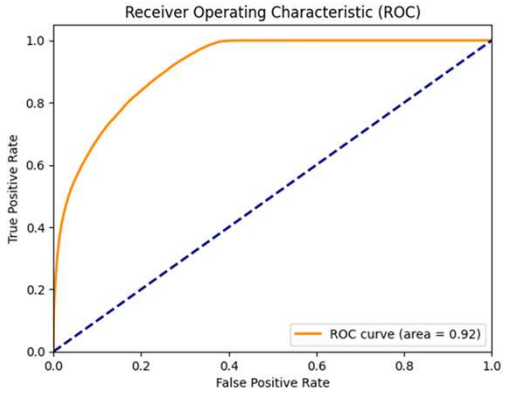
Table 4: XGBoost Response and Default Model Performance

	F1	ACCURACY	AUC	SENSITIVITY	PRECISION
RESPONDER MODEL	0.2055	0.664	0.9429	0.7513	0.2498
DEFAULT MODEL	0.0497	0.880	0.9049	0. 9078	0.0919

Responder Model



Default Model



KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

Response Models			
		Quite Certain	Uncertain
Random Forest (weighted)	correct	29.91%	31.10%
	wrong	12.19%	26.80%
Random Forest (unweighted)	correct	73.08%	1.92%
	wrong	23.95%	1.05%

Atlanticus®

KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

Default Models			
		Quite Certain	Uncertain
Random Forest (weighted)	correct	0.60%	54.81%
	wrong	0.30%	44.29%
Random Forest (unweighted)	correct	30.18%	29.49%
	wrong	18.00%	22.34%

Atlanticus®

KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

XGBoost Models			
		Quite Certain	Uncertain
Response Model	correct	8.618%	17.366%
	wrong	12.191%	82.548%
Default Model	correct	73.081%	3.418%
	wrong	2.961%	96.591%

Atlanticus®

KENNESAW STATE UNIVERSITY

A Note on Calibration

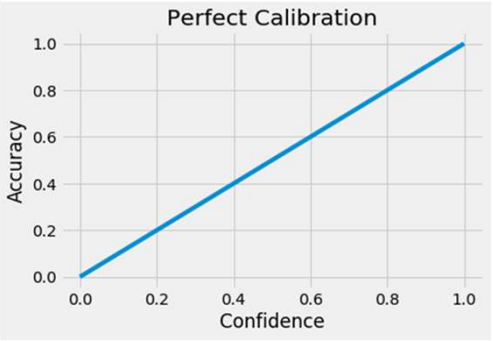
- Postprocessing technique to adjust predictions so that they are probabilistically meaningful.
- Machine and deep learning algorithms tend to be not as well-calibrated.
- A model is well-calibrated if for any p, a prediction class with confidence p is correct $100 * p$ percent of the time.

Atlanticus®

KENNESAW STATE UNIVERSITY

Calibration

- Measured using:
  - Scoring rules e.g., brier score
  - Calibration plot/reliability diagram



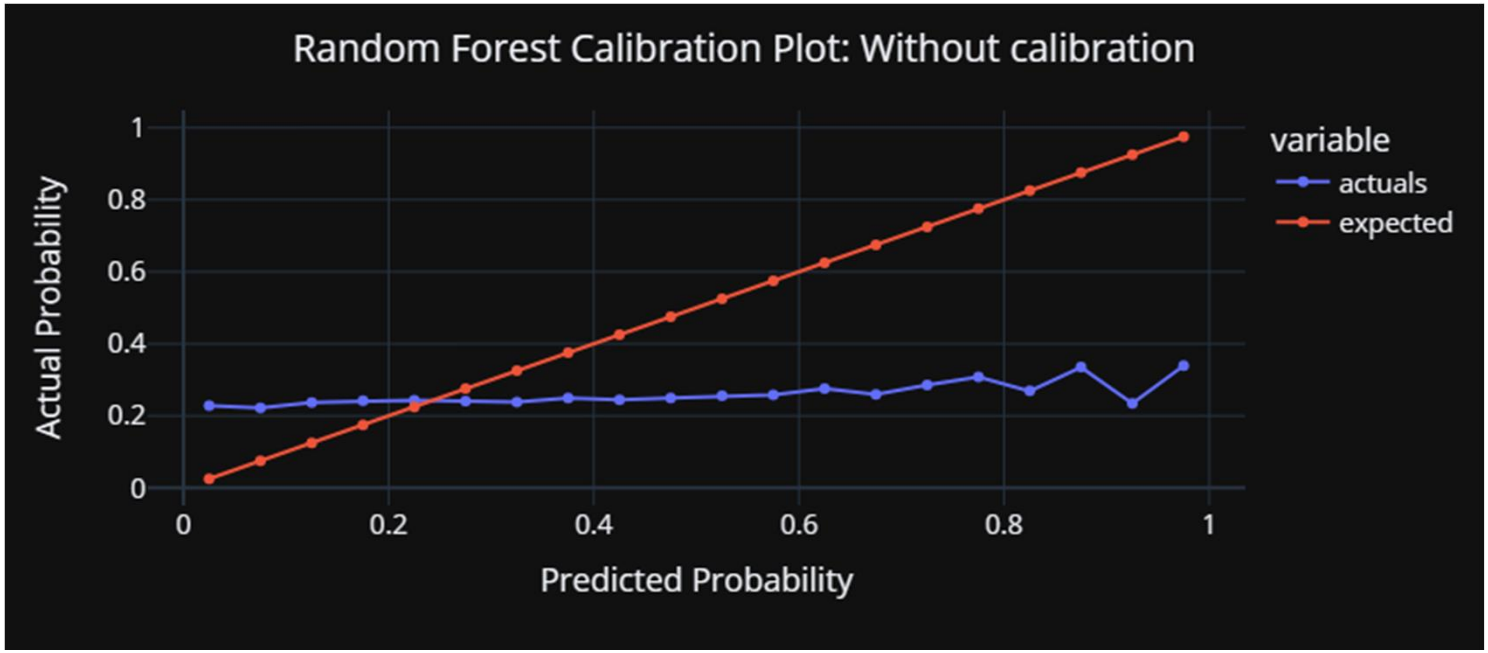
- Model is calibrated by fitting a calibration function that is:
  - Monotonic
  - Minimizes the proper scoring rule
  - Trained on independent data

- Calibration methods include Platt Scaling and isotonic regression.

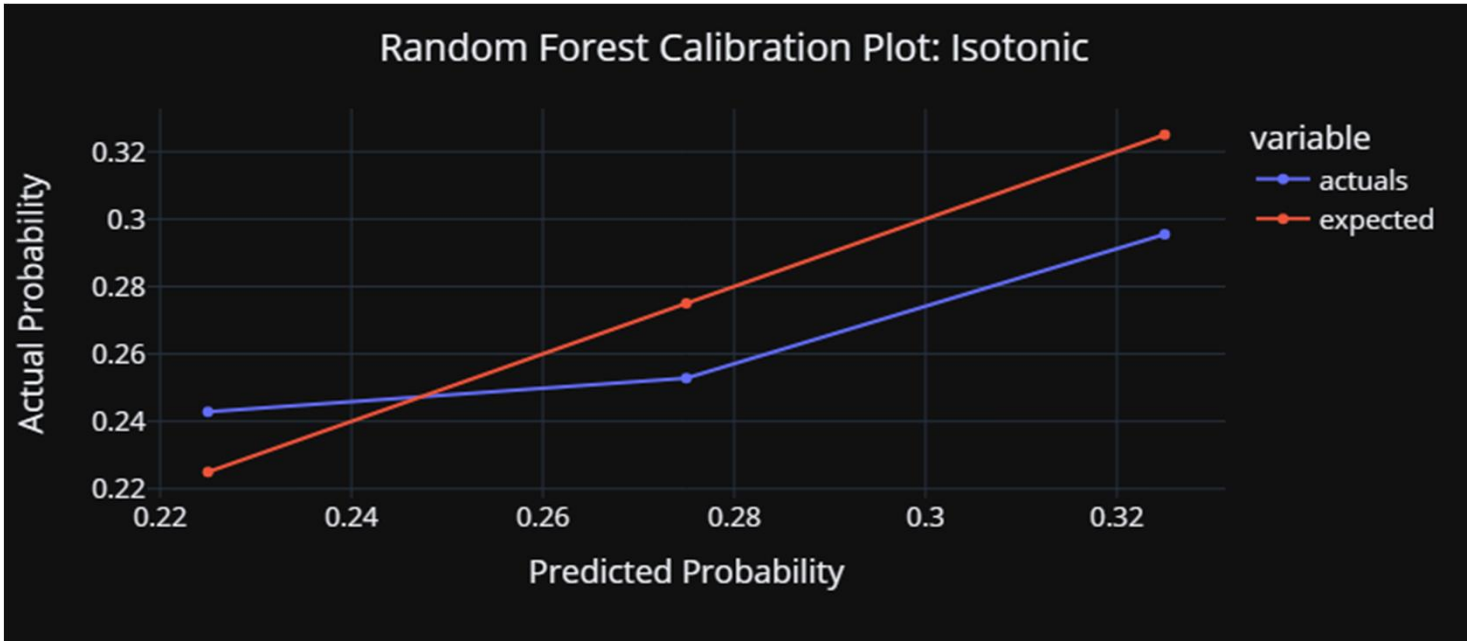
Atlanticus®

KENNESAW STATE UNIVERSITY

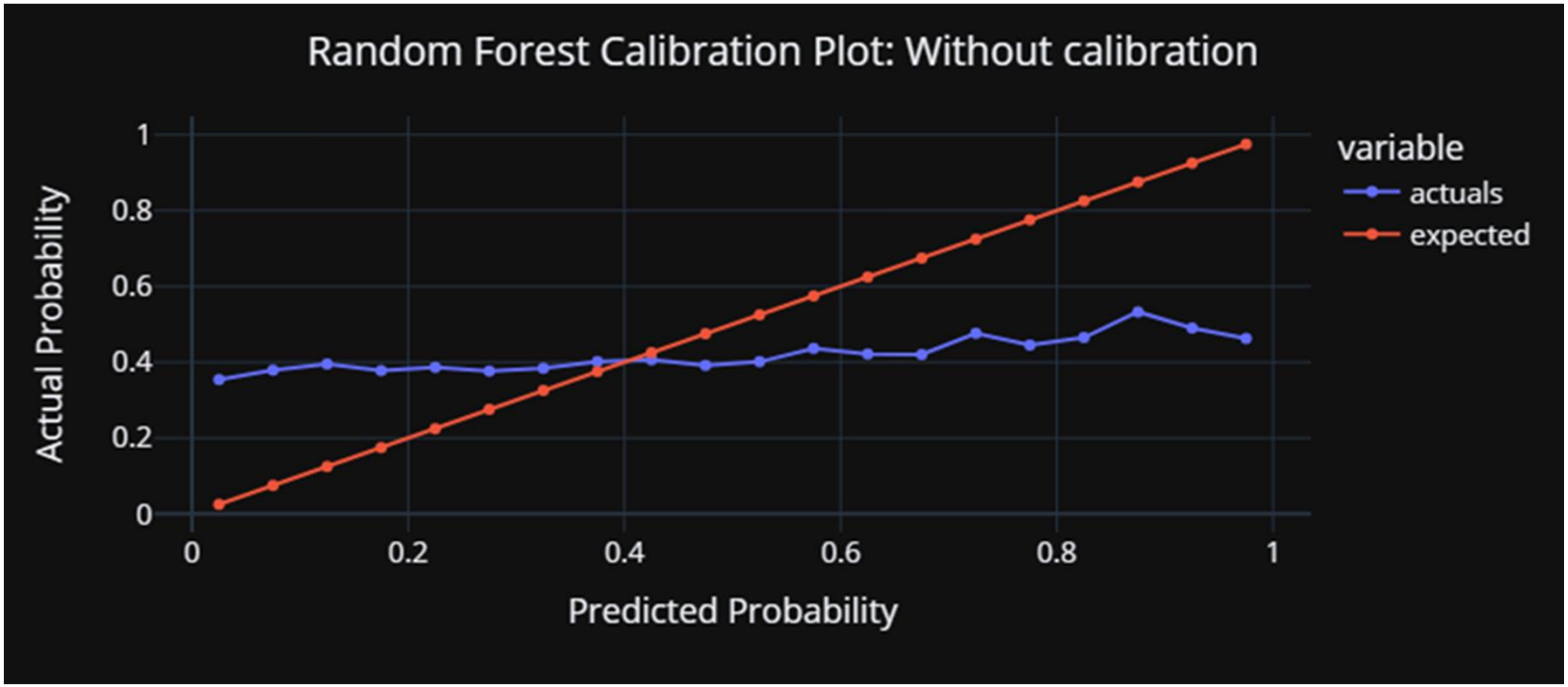
Calibrating Our Responder Model



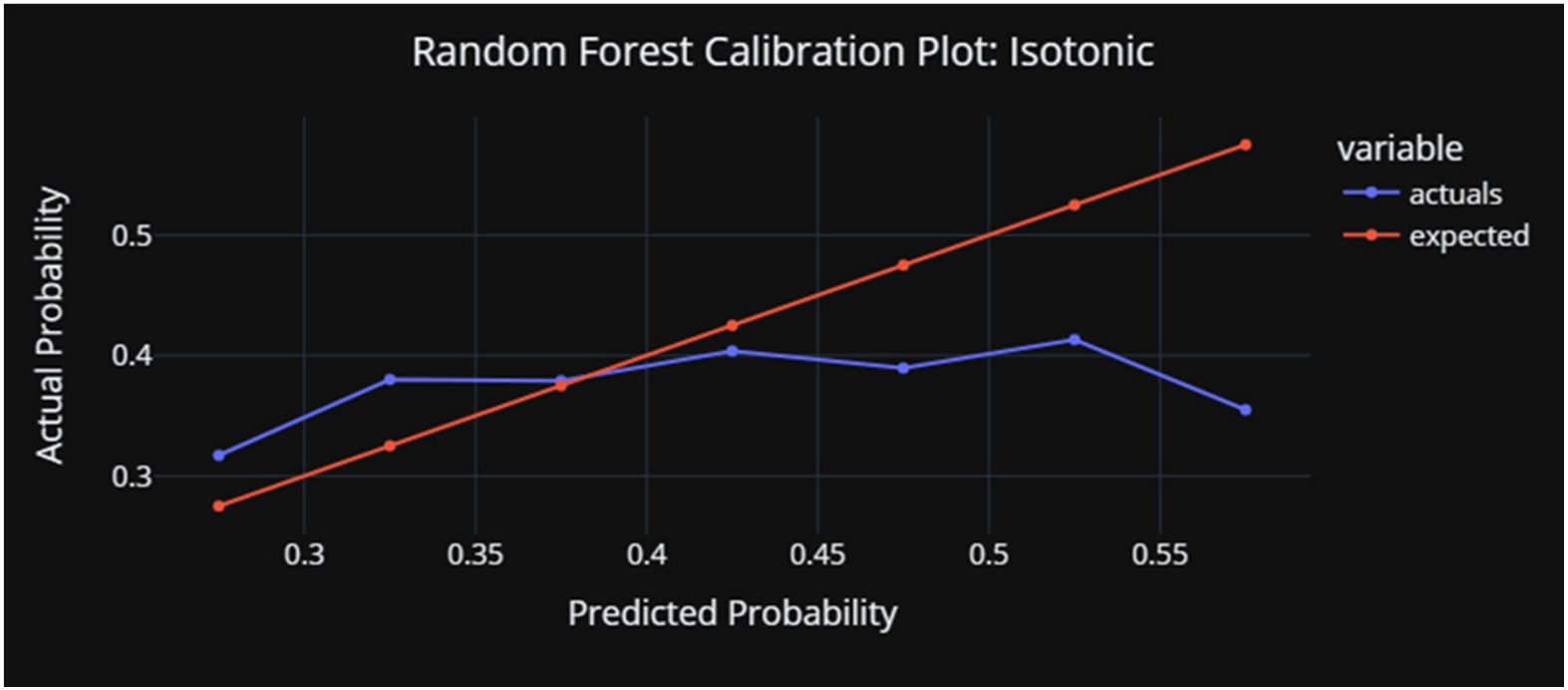
Calibrating Our Responder Model



Calibrating Our Default Model



Calibrating Our Default Model



Calibration Results

Model	Before/after calibration	Brier Score	Accuracy	Specificity	Sensitivity
Responder	Before	0.2208	0.6572	0.7988	0.2217
	After	0.1809	0.7548	1.0	0.0
Default	Before	0.2610	0.5966	0.9879	0.0213
	After	0.2404	0.5974	0.9940	0.0054

Atlanticus®

KENNESAW STATE UNIVERSITY

How to Score? Primary Approach

- Follow same process for removing missing observations and variables.
- Do median imputation on remaining variables.
- Use the selected variables from our variable clustering results.
- Bin variables using the tree-based approach.
- Run responder dataset through uncalibrated random forest (weighted).
- Run default dataset through the calibrated random forest (unweighted).

Atlanticus®

KENNESAW STATE UNIVERSITY

How to Score? Secondary Approach

- Follow same process for removing missing observations and variables.
- Do MICE imputation on remaining variables.
- Use the selected variables from our variable clustering results.
- Bin variables using the tree-based approach.
- Run responder dataset through XGBoost response model.
- Run default dataset through the XGBoost default model.

Atlanticus®

KENNESAW STATE UNIVERSITY

Who to Select?

Response predicted probs > y

Default predicted probs < x

	0.5	**0.6**	**0.7**	**0.8**	**0.9**
0.1	-12120	-6550	-2300	-510	-90
0.2	-12120	-6550	-2300	-510	-90
0.3	-11870	-6300	-2050	-260	**+160**
0.4	-25320	-19750	-15500	-13710	-13290
0.5	-138670	-13310	-128850	-127060	-1226640

Atlanticus®

KENNESAW STATE UNIVERSITY

Conclusions

- In terms of model performance:
  - Random Forest (weighted) model gave best performance for modelling response in primary approach.
  - Random Forest (unweighted) model after calibration gave best performance for modelling default in primary approach.
  - XGBoost models in secondary approach gave best overall performance.
- Best profit for primary approach was from the unweighted responder random forest.

Atlanticus®

KENNESAW STATE UNIVERSITY

Thank you!

Questions?

References

Niculescu-Mizil, A., & Caruana, R. (2005, August). Predicting good probabilities with supervised learning. In Proceedings of the 22nd international conference on Machine learning (pp. 625-632).

Nyongesa, D. (2020). Variable selection using Random Forests in SAS. In SAS Global Forum (Vol. 4826).

Guoping Zeng, (2013) "Metric Divergence Measures and Information Value in Credit Scoring", Journal of Mathematics, vol. 2013, Article ID 848271, 10 pages. https://doi.org/10.1155/2013/848271

Siddiqi, N. (2006). Credit risk scorecards–developing and implementing intelligent credit scoring. Wiley

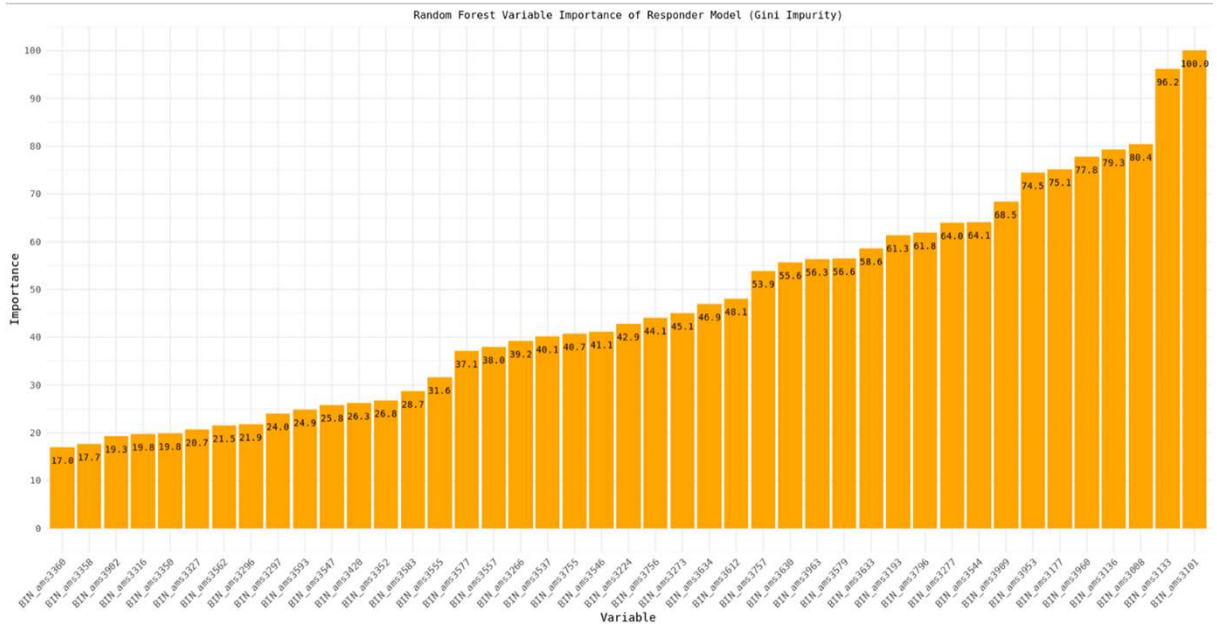
Asgharnezhad, H., Shamsi, A., Alizdesani, R., Khorsravi, A. (2022). Objective evaluation of deep uncertainty predictions for COVID-19 detection. Scientific Reports. DOI:10.1038/s41598-022-05052-x

KENNESAW STATE UNIVERSITY

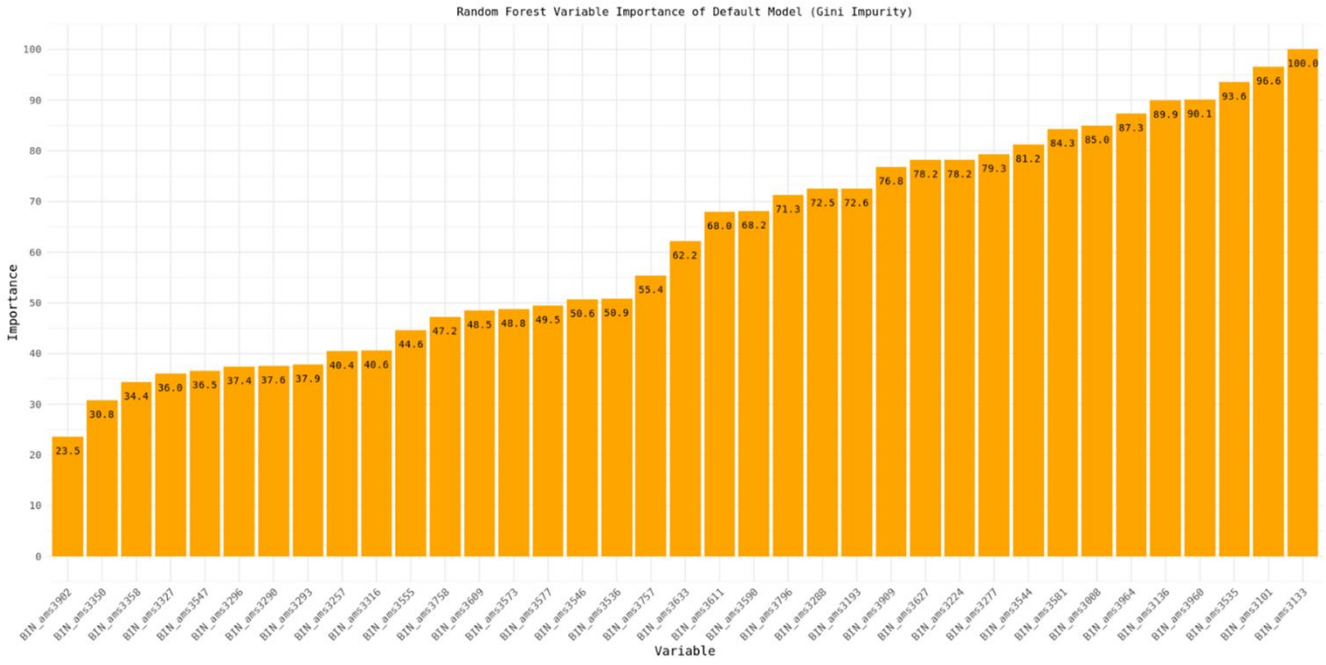
Appendix

KENNESAW STATE UNIVERSITY

Responder Random Forest Important Variables



Default Random Forest Important Variables

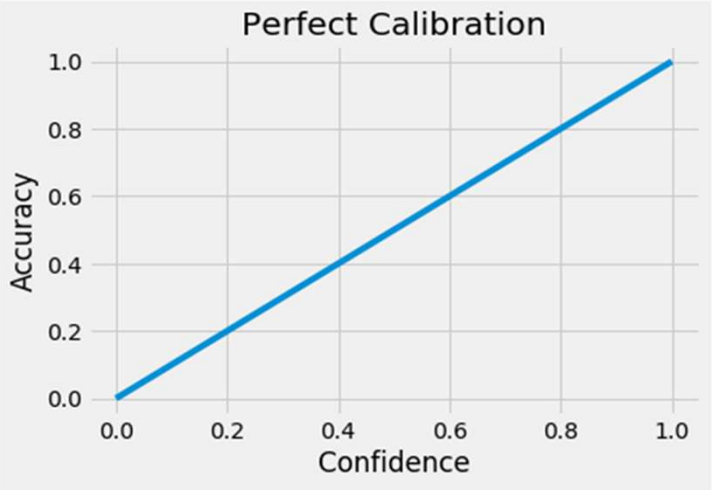


Calibration

Let Y be a binary outcome, Yˆ be the class prediction and Pˆ it's associated confidence the perfect calibration can be defined as:

$$P\left(\hat{Y} = Y \middle| \hat{P} = p\right) = p, \qquad \forall p \in [0,1]$$

A model is perfectly calibrated if, for any p, a prediction class with confidence p is correct $100 * p$ percent of the time.



KENNESAW STATE UNIVERSITY

Logistic Regression Models

Stepwise selection

Responder

- Vars selected: ams3224, ams3296
- Cutpoint set at 0.245

Default

- Vars selected: ams3257, ams3609, ams3633, ams3796, ams3224, ams3288, ams3581, ams3757
- Cutpoint set at 0.46

KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Responder Models

Logistic Regression

	Quite Certain	Uncertain
correct	75.20%	0
wrong	24.80%	0

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

Random Forest (unweighted)

	Quite Certain	Uncertain
correct	73.08%	1.92%
wrong	23.95%	1.05 %

Random Forest (weighted)

	Quite Certain	Uncertain
correct	29.91%	31.10%
wrong	12.19%	26.80%

Atlanticus

KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Responder Models

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

Random Forest (unweighted)

	Quite Certain	Uncertain
correct	73.08%	1.92%
wrong	23.95%	1.05 %

Random Forest (weighted)

	Quite Certain	Uncertain
correct	29.91%	31.10%
wrong	12.19%	26.80%

Atlanticus®

KENNESAW STATE UNIVERSITY

Uncertainty in Model Predictions

Default Models

Uncertain: 0.4 < prob < 0.6
Quite Certain: prob < 0.4 or prob > 0.6

Random Forest (unweighted)

	Quite Certain	Uncertain
correct	30.18%	29.49%
wrong	18.00%	22.34%

Random Forest (weighted)

	Quite Certain	Uncertain
correct	0.60%	54.81%
wrong	0.30%	44.29%

Atlanticus

KENNESAW STATE UNIVERSITY

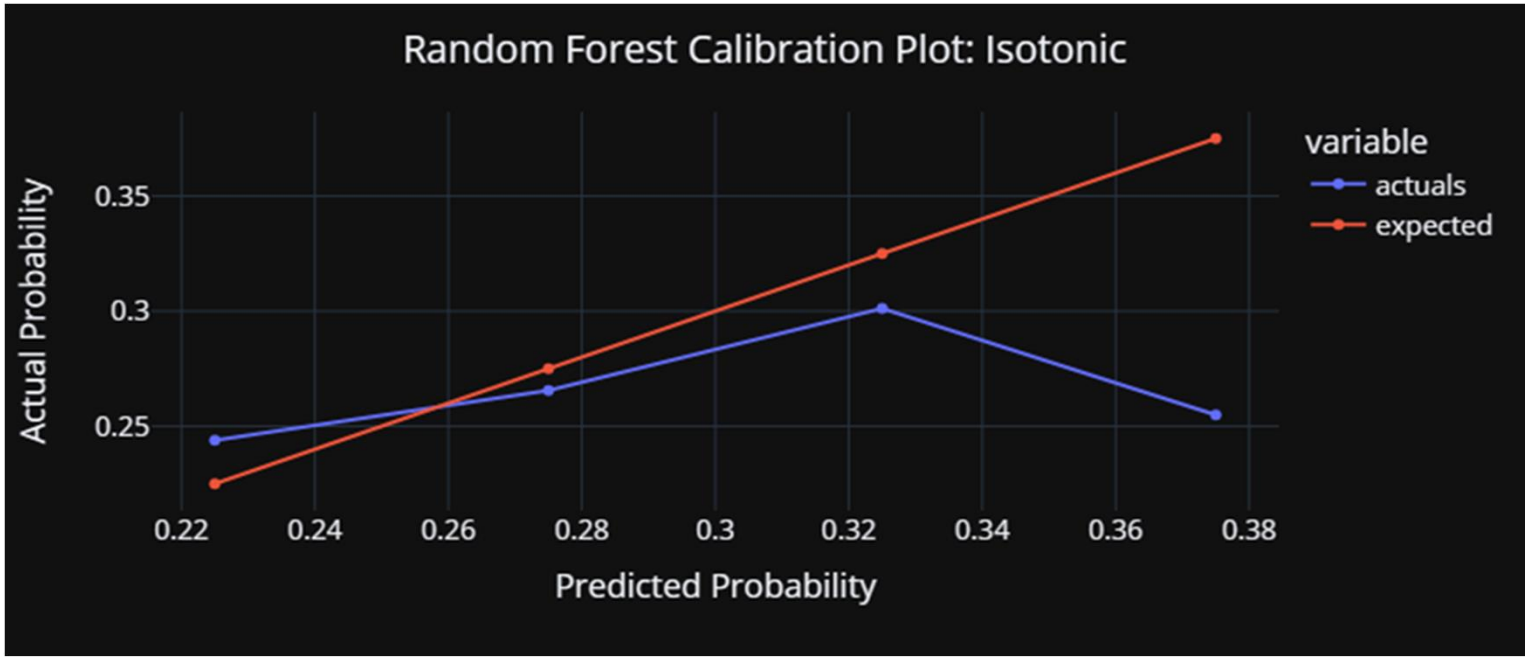
Calibration Results

Model	Before/after calibration	Brier Score	Accuracy	Specificity	Sensitivity
Responder	Before	0.2026	0.7500	0.9894	0.0203
	After	0.1857	0.7531	0.9998	0.0002
Default	Before	0.2610	0.5966	0.9879	0.0213
	After	0.2404	0.5974	0.9940	0.0054

Atlanticus®

KENNESAW STATE UNIVERSITY

Calibrating Our Responder Model



Calibrating Our Responder Model