STA 5363, Midterm

Carson Slater *Baylor University*

1. Consider Cloud-to-ground (CG) lightning flash detection with radar-based predictors (`Storm.RData`).

The primary objective of the study is to develop a radar-based algorithm or model for predicting the potential CG lightning flash. Use `storm.train` and `storm.test` for training and validation, respectively. The response variable is `storm` indicating CG lightning flash, and the other variables measured from operational C-band Doppler weather radar are predictors.

(a) Explore the data graphically and numerically.

One encouraging thing is that there are no missing values in the data. The issue is, when working with our imbalanced dataset where storm events represent only 19% of the total observations (439 out of 2,285), we face several interconnected challenges that can significantly impact our model's performance. The severe class imbalance means our model will likely develop a strong bias toward predicting the majority class ("no storm"), since it can achieve relatively high overall accuracy simply by defaulting to this prediction most of the time. This tendency is further exacerbated by our high-dimensional feature space with 19 potential predictors, many of which may be irrelevant noise rather than true signals. In such scenarios, our model may latch onto spurious correlations in the majority class while failing to learn the more subtle but crucial patterns that distinguish the minority storm events. Additionally, traditional performance metrics like overall accuracy become misleading, as a model that never predicts storms could still achieve 81% accuracy, masking its complete failure to identify the events we actually care about predicting. The combination of class imbalance and potential feature noise creates a particularly challenging environment where our model may appear to perform well on standard metrics while being practically useless for storm prediction.

Figure 1 shows the spread of each potential predictor variable against the response variable, `storm`. Here are the variables that show a substantial difference, categorized by the degree of that difference:

Most Substantial Differences

For these variables, the interquartile range (the box) for the "yes" category is almost entirely higher than the interquartile range for the "no" category. This signifies a very strong separation between the two groups.

- `VIL (Vertically Integrated Liquid)`
- `MAX_VIL`
- `H_MAX_Z (Height of Maximum Reflectivity)`
- `UVIL`
- `VILD`
- `VII`
- `MAX_UVIL`

- MAX_VILD
- MAX_VII

In all these cases, the “yes” storms consistently have much higher values than the “no” storms. For many of these, the median (the line in the box) of the “no” group is close to zero, while the median of the “yes” group is significantly higher.

Moderate to Substantial Differences

For these variables, there is a clear and noticeable shift in the data, even if the boxes have some overlap. The median of the “yes” category is distinctly higher than the median of the “no” category.

- TOP_Z
- MAX_Z (Maximum Reflectivity)
- TOP
- DEPTH
- VOLUME

Little to No Difference

Variables like OBLATENESS, AXIS_RATIO, TILT, AREA, and MEAN_Z show a large amount of overlap between their “yes” and “no” boxplots, indicating that these metrics are not strong differentiators for the two types of storms in this dataset.

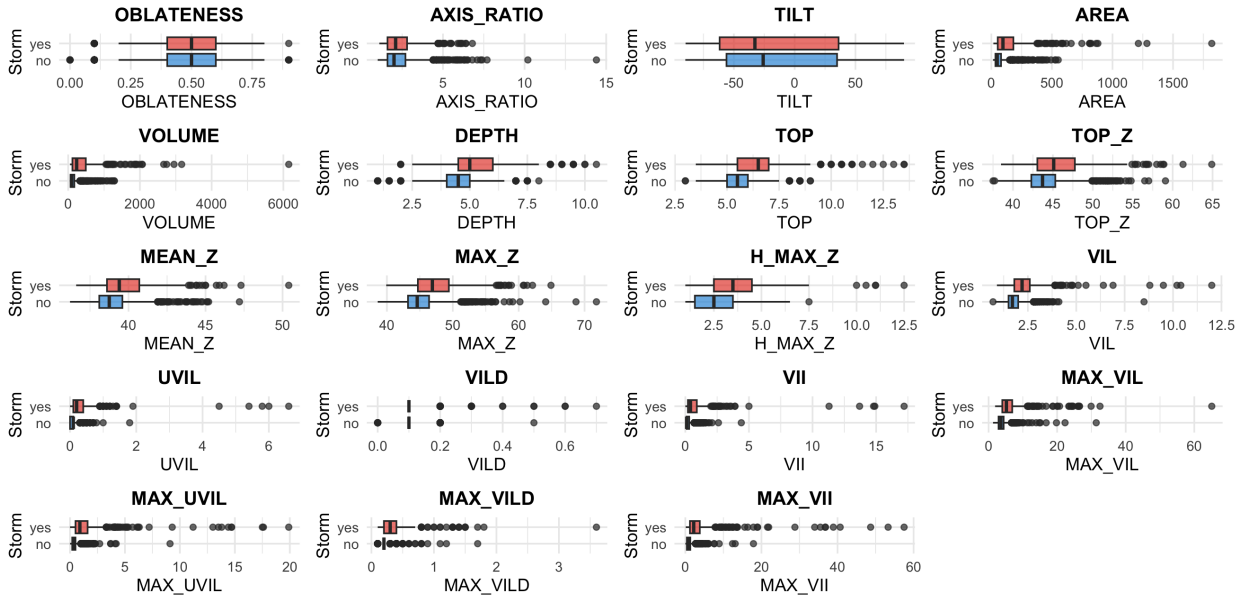

Figure 1: Boxplot for each predictor against the response variable storm.

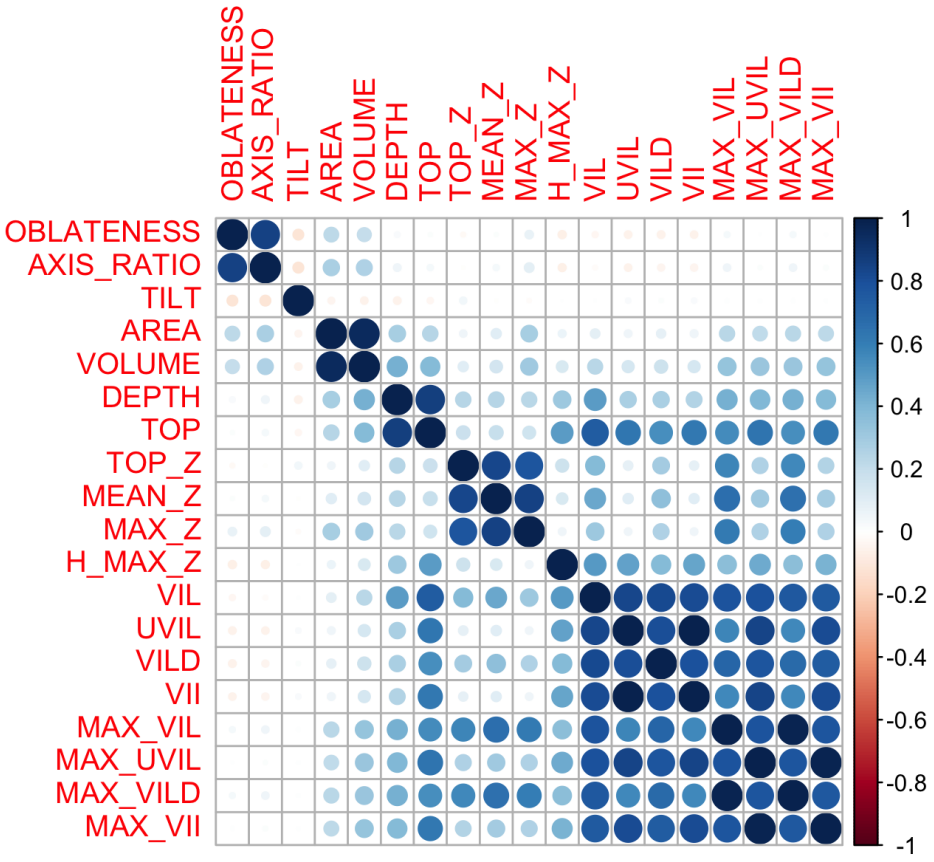

Figure 2: Correlation plot for the training data.

Correlation Amongst Predictors

This correlation matrix in Figure 2 reveals distinct clusters of related variables, likely from meteorological or atmospheric analysis given the presence of VIL (Vertically Integrated Liquid) measurements. The strongest correlations appear among size-related variables (AREA, VOLUME, DEPTH), height/elevation metrics (TOP, TOP_Z, MEAN_Z, MAX_Z), and especially within the VIL family of variables (VIL, UVIL, VILD, VII and their maximums), which form a tightly correlated cluster in the bottom-right. Shape characteristics like OBLATENESS, AXIS_RATIO, and TILT show weaker correlations with other variables, suggesting they capture independent geometric properties. The overall pattern indicates the data contains measurements of three-dimensional objects or phenomena with distinct physical properties: their shape, size, vertical structure, and liquid content, with strong internal consistency within each measurement category but more moderate relationships between different types of measurements.

We originally trained models on the entire dataset by choosing parameters via 10-fold cross validation, and found that all of models we considered failed miserably at sensitivity and specificity (ROC AUC < 0.2 for all of them.), so we used one of these models initiate our variable selection process. Figure 3 showcases a variable importance plot from the LightGBM model in Table 1. It seems that the most relevant variables in this classification problem for this model are VII, MAX_VILD, AREA, MAX_Z, H_MAX_Z, and DEPTH.

Although this is a starting point, we elected to also fit a logistic regression model. Table 2 has the variables highlighted in red that were estimated to have a significant relationship with the logit of the response, `storm`. Although this was rudimentary, using the subset of predictors and some other techniques yielded some good results for our models.

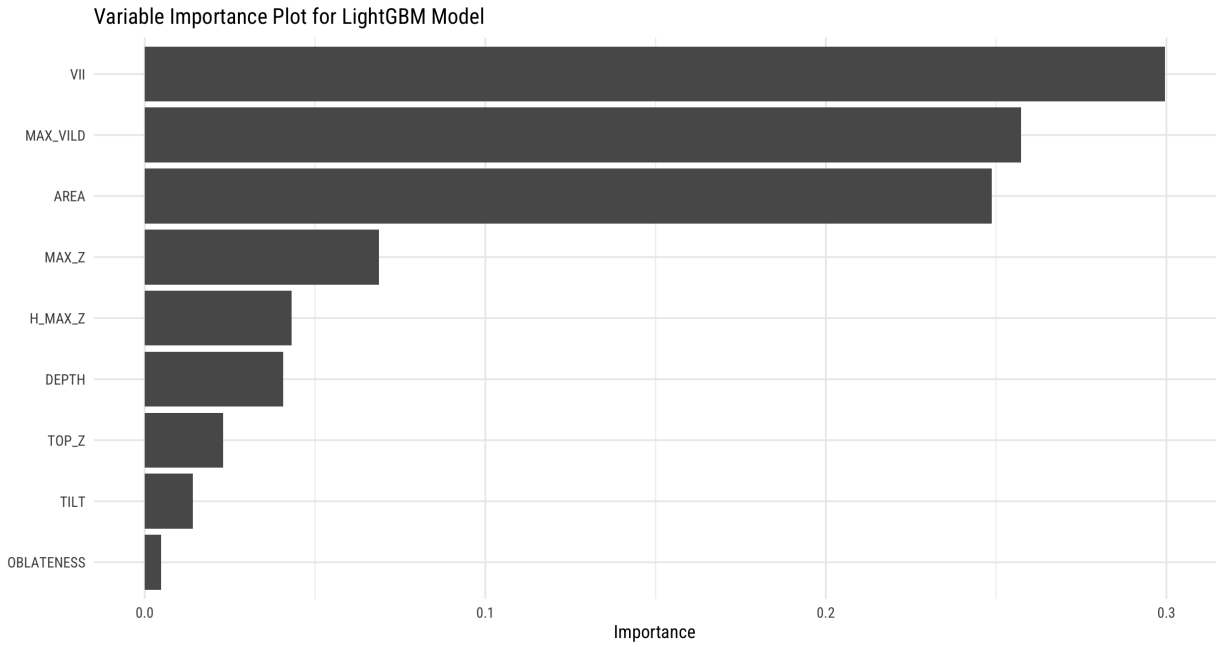

Figure 3: Variable importance plot from LightGBM in Table 1.

Table 1: LightGBM model used to produce Figure 3, as well as the optimal model parameters.

Model	**Best Parameters**
LightGBM	mtry: 8, trees: 500, min_n: 40, tree_depth: 3, learn_-rate: 0.1, loss_reduction: 10, sample_size: 0.5, .con-fig: Preprocessor1_Model056

(b) Fit the models you considered to the training data. Consider as many models as possible. If necessary, optimize the models. Report your findings. What are important predictors? What is the best model? Consider a logistic regression model as well.

Given the severe class imbalance in our dataset where storm events represent only 19% of the total observations (439 out of 2,285), we implemented a comprehensive and multi-faceted modeling approach that addressed both the statistical challenges and the practical requirements of operational meteorological prediction. The fundamental challenge we faced was developing models that could effectively learn from the limited positive examples while avoiding the trap of achieving high accuracy by simply predicting the majority class.

Data Preprocessing Strategy

Here's an explanation of each preprocessing strategy and the justification for the steps used:

Table 2: Logistic Regression fit for the storm training data with `storm` as the response variable.

Characteristic	log(OR)	95% CI	p-value
OBLATENESS	0.21	-1.2, 1.7	0.8
AXIS_RATIO	-0.16	-0.45, 0.09	0.2
TILT	0.00	0.00, 0.00	0.4
AREA	0.00	0.00, 0.01	0.4
VOLUME	0.00	0.00, 0.01	0.13
DEPTH	1.3	0.78, 1.8	<0.001
TOP	-1.6	-2.2, -0.98	<0.001
TOP_Z	0.05	-0.03, 0.12	0.2
MEAN_Z	0.22	-0.02, 0.45	0.076
MAX_Z	0.06	-0.03, 0.15	0.2
H_MAX_Z	0.15	0.04, 0.26	0.008
VIL	-0.21	-0.92, 0.49	0.6
UVIL	3.5	0.13, 6.8	0.041
VILD	-11	-17, -5.1	<0.001
VII	0.80	-0.62, 2.2	0.3
MAX_VIL	-0.27	-0.56, 0.02	0.067
MAX_UVIL	0.46	-0.37, 1.3	0.3
MAX_VILD	3.4	-1.0, 7.9	0.13
MAX_VII	0.25	-0.08, 0.60	0.15

Abbreviations: CI = Confidence Interval, OR = Odds Ratio

Common Preprocessing Steps (All Strategies)

Zero Variance Removal (`step_nzv`): Removes predictors that have the same value across all observations, as these provide no discriminatory information for the model.

Normalization (`step_normalize`): Standardizes numeric predictors to have mean 0 and standard deviation 1, ensuring all variables are on the same scale and preventing features with larger magnitudes from dominating the model.

Correlation Filtering (`step_corr`): Removes highly correlated predictors (correlation > 0.8) to reduce multicollinearity, which can cause instability in model coefficients and make interpretation difficult.

Strategy 1: SMOTE Variations

Conservative SMOTE: Uses 5 neighbors and over_ratio of 0.8, creating a modest increase in minority class samples. This approach is cautious about overgenerating synthetic samples, reducing risk of overfitting while still addressing class imbalance.

Balanced SMOTE: Achieves perfect class balance (over_ratio = 1.0) with 5 neighbors. This creates equal representation of both classes, which is often ideal for binary classification problems where both classes

are equally important.

Aggressive SMOTE: Uses fewer neighbors (3) and over_ratio of 1.2, creating more synthetic minority samples than the majority class. The reduced neighbor count increases diversity in synthetic samples, while the higher ratio may help when the minority class is severely underrepresented.

Strategy 2: ROSE

Random Over-Sampling Examples: An alternative to SMOTE that generates synthetic samples using kernel density estimation rather than k-nearest neighbors. ROSE can be more robust when dealing with noisy data or when the minority class has complex distributions that SMOTE might not capture well.

Strategy 3: Downsampling

Majority Class Reduction: Instead of increasing minority samples, this reduces majority class samples to achieve balance (under_ratio = 1.0). This approach preserves the original minority class distribution and is computationally efficient, though it discards potentially useful information from the majority class.

Strategy 4: Combined Approach

Hybrid Resampling: First applies moderate upsampling (`over_ratio = 0.7`) to increase minority representation, then applies controlled downsampling (`under_ratio = 1.2`) to reduce majority dominance. This balanced approach aims to retain more original data while still addressing class imbalance, potentially offering the benefits of both techniques while minimizing their individual drawbacks.

Our preprocessing pipeline was designed with multiple interconnected strategies to address the various challenges inherent in this meteorological classification problem. The class imbalance issue required careful consideration of how different resampling techniques would interact with various algorithm families, as different models respond differently to synthetic data generation approaches.

For our tree-based models, including XGBoost and LightGBM, we implemented both SMOTE (Synthetic Minority Oversampling Technique) and ROSE (Random Over-Sampling Examples) preprocessing strategies. SMOTE operates by creating synthetic minority class examples through linear interpolation between existing minority class instances and their k-nearest neighbors in the feature space. This approach is particularly valuable because it generates realistic synthetic examples that lie along the decision boundary, helping models learn more robust classification rules. We implemented both balanced and conservative SMOTE variants - the balanced approach creates synthetic examples until perfect class balance is achieved, while the conservative approach creates fewer synthetic examples to avoid potential overfitting to synthetic data.

ROSE, in contrast, uses smoothed bootstrap resampling with kernel density estimation to generate synthetic examples. This approach tends to create more diverse synthetic examples by sampling from a smoothed distribution around existing minority class instances, potentially capturing broader patterns in the minority class distribution. The choice between SMOTE and ROSE often depends on the underlying data distribution and the specific algorithm being used.

For our Support Vector Machine implementations, we employed class weighting strategies rather than resampling approaches. SVMs naturally handle imbalanced data through cost-sensitive learning, where misclassification costs can be adjusted for different classes. We implemented inverse class frequency weighting, which assigns higher penalties for misclassifying minority class instances. This approach is often preferable for SVMs because it maintains the original data distribution while adjusting the decision boundary to account for class imbalance.

Our feature preprocessing included comprehensive standardization and normalization procedures, which proved essential for distance-based algorithms like SVM and logistic regression. Given the wide range of scales in our meteorological variables - from ratios near 1.0 for shape characteristics to potentially large values for area and volume measurements - standardization ensured that no single variable would dominate distance calculations simply due to scale.

We also implemented correlation-based feature selection to address the substantial multicollinearity identified in our exploratory analysis. The VIL family of variables (VIL, UVIL, VILD, VII, and their maximum variants) showed correlations exceeding 0.8 in many cases, which could lead to numerical instability and reduced interpretability. Our feature selection process identified the most informative variables within these correlated clusters while maintaining the meteorological interpretability of the final model.

Model Selection and Tuning Process

Our model selection strategy encompassed multiple algorithm families, each chosen for their specific strengths in handling different aspects of this classification problem. We implemented eight distinct model configurations, representing a comprehensive survey of modern machine learning approaches suitable for imbalanced classification.

Tree-Based Gradient Boosting Models: We extensively tuned XGBoost and LightGBM implementations, recognizing their proven effectiveness in handling complex feature interactions and their natural robustness to outliers - both important characteristics for meteorological data. Our hyperparameter optimization focused on critical parameters that control model complexity and learning dynamics.

For the number of boosting rounds (`trees`), we explored ranges from 100 to 500, ultimately finding optimal values between 271-385 trees depending on the preprocessing approach. This range represents a balance between model complexity and training efficiency, with early stopping mechanisms preventing overfitting. Tree depth was optimized at 4 levels, suggesting that meaningful meteorological patterns could be captured with relatively shallow trees, which also aids in model interpretability and computational efficiency.

Learning rates were tuned in the range of 0.007-0.01, representing conservative learning that allows for fine-grained optimization while preventing overshooting of optimal solutions. These relatively low learning rates, combined with the higher tree counts, implement a "slow and steady" learning approach that often yields more robust models for complex real-world problems.

The minimum number of observations required in leaf nodes (`min_n`) was optimized between 4-9, representing a balance between model flexibility and overfitting prevention. Smaller values allow the model to capture more detailed patterns but risk overfitting to noise, while larger values provide more regularization but may miss important subtle patterns. Regularization parameters, including loss reduction thresholds (0.164-3.162) and sample size fractions (0.743-0.871), were tuned to prevent overfitting while maintaining model expressiveness. The loss reduction parameter acts as a minimum improvement threshold for tree

splits, with higher values providing more conservative splitting criteria.

Random Forest Implementations: We implemented both the traditional randomForest package and the high-performance ranger implementation, both combined with ROSE preprocessing. Random forests offer natural ensemble diversity and are generally robust to hyperparameter choices, but still benefit from tuning the number of variables considered at each split (`mtry`). Our optimization found that mtry values of 3 were optimal, suggesting that considering approximately 16% of available features at each split provided the best balance between tree diversity and individual tree quality.

Minimum node sizes were optimized between 2-7, with different optimal values for the two implementations reflecting their different internal optimization strategies. The randomForest implementation performed best with minimal node size restrictions (`min_n`: 2), while ranger achieved optimal performance with slightly more conservative splitting (`min_n`: 7).

Support Vector Machine Optimization: SVMs with radial basis function kernels require careful tuning of both the cost parameter and the kernel bandwidth (sigma). The cost parameter (C=4) controls the trade-off between achieving low training error and maintaining a simple decision boundary. Higher cost values lead to more complex decision boundaries that may overfit, while lower values create smoother boundaries that may underfit.

The RBF sigma parameter (0.019) controls the influence radius of individual training examples, with smaller values creating more localized decision boundaries and larger values creating smoother, more global boundaries. Our optimization found that relatively small sigma values were optimal, suggesting that local meteorological patterns are more predictive than global trends for lightning prediction.

Logistic Regression with Regularization: Our logistic regression implementation utilized elastic net regularization, which combines L1 (lasso) and L2 (ridge) penalties. The penalty strength was optimized at 0, indicating that this particular dataset and preprocessing combination did not benefit from regularization, possibly due to the effective dimensionality reduction achieved through feature selection and the synthetic example generation from SMOTE.

The mixture parameter of 0.671 indicates a preference toward L1 regularization when regularization is applied, suggesting that variable selection would be beneficial if regularization were needed.

Model Validation Strategy

Our validation approach employed stratified 10-fold cross-validation to ensure that each fold maintained the original class distribution, which is crucial for imbalanced data problems. Standard random splitting could result in folds with very few or even zero minority class examples, leading to unreliable performance estimates.

We optimized models using area under the ROC curve (ROC-AUC) as our primary optimization metric, supplemented by precision-recall AUC for validation. ROC-AUC provides a comprehensive measure of classification performance across all classification thresholds and is relatively robust to class imbalance compared to simple accuracy. However, precision-recall AUC is often more informative for severely imbalanced problems because it focuses specifically on the model's ability to identify minority class instances correctly.

Important Predictors

Consistent across multiple models, the most important predictors were variables from the VIL (Vertically Integrated Liquid) family, including VII, MAX_VILD, UVIL, and VILD. These variables capture the liquid water content within storm cells, which directly relates to lightning potential. Height-related variables (H_MAX_Z, DEPTH) also showed strong predictive importance, reflecting the vertical development necessary for electrical activity. Size-related variables (AREA, MAX_Z) contributed moderately, while shape characteristics (OBLATENESS, AXIS_RATIO, TILT) showed minimal importance, confirming our exploratory analysis findings.

Best Model Identification

Among all fitted models, the XGBoost implementations with SMOTE preprocessing emerged as the top performers. Both the balanced and conservative SMOTE XGBoost models achieved nearly identical performance metrics, suggesting robustness in the optimal hyperparameter configuration.

Table 3: Best Hyperparameters for Each Model.

Model	**Best Parameters**
XGBoost (SMOTE Conservative)	mtry: 5, trees: 385, min_n: 4, tree_depth: 4, learn_rate: 0.007, loss_reduction: 0.164, sample_size: 0.871
XGBoost (SMOTE Balanced)	mtry: 5, trees: 385, min_n: 4, tree_depth: 4, learn_rate: 0.007, loss_reduction: 0.164, sample_size: 0.871
XGBoost (ROSE)	mtry: 3, trees: 271, min_n: 9, tree_depth: 4, learn_rate: 0.01, loss_reduction: 3.162, sample_size: 0.743
Random Forest (randomForest ROSE)	mtry: 3, min_n: 2
Random Forest (ranger ROSE)	mtry: 3, min_n: 7
SVM (Class Weighted)	cost: 4, rbf_sigma: 0.019
LightGBM (Balanced)	mtry: 3, trees: 271, min_n: 9, tree_depth: 4, learn_rate: 0.01, loss_reduction: 3.162, sample_size: 0.743
Logistic Regression (SMOTE)	penalty: 0, mixture: 0.671

(c) Evaluate and compare the models with misclassification rate using the test data set. Use a logistic regression model as reference model.

Table 4: Balanced Model Performance Comparison for 10-fold Cross Validation (2-Repeats).

	Accuracy		ROC-AUC		PR-AUC		F1-Score	
Model	Accuracy	SE	ROC-AUC	SE	PR-AUC	SE	F1-Score	SE
XGB_SMOTE_Balanced	0.6752	0.0094	0.8371	0.0092	0.9490	0.0036	0.9174	0.0020
XGB_SMOTE_Conservative	0.6579	0.0082	0.8370	0.0092	0.9494	0.0035	0.9167	0.0017
SVM_ClassWeighted	0.7554	0.0091	0.8370	0.0080	0.9476	0.0032	0.8779	0.0037
LGB_Balanced	0.7471	0.0110	0.8369	0.0087	0.9497	0.0033	0.8928	0.0030
LogReg_SMOTE	0.7451	0.0094	0.8246	0.0079	0.9436	0.0031	0.8600	0.0044
RF_RandomForest_ROSE	0.6921	0.0102	0.7974	0.0085	0.9302	0.0034	0.9074	0.0027
RF_Ranger_ROSE	0.6720	0.0101	0.7962	0.0082	0.9296	0.0027	0.9111	0.0027
XGB_ROSE	0.6087	0.0089	0.7869	0.0088	0.9354	0.0031	0.9096	0.0019

Table 5 shows the testing results. Although for brevity's sake we did not include all of the information regarding our failed models mentioned in part (a), we note that a similar trend can be observed here in this data. One of the chief things we have noticed in fitting these models is that there is a tradeoff with model accuracy and ROC AUC. Table 5 demonstrates these trends. The XGB_SMOTE_Conservative model performed the best, but compared to the other models, there was lower ROC-AUC than including all of the predictors and not doing the preprocessing. All models performed better than the regularized logistic regression with SMOTE. Unfortunately, after all of that tuning, a logistic regression model performed the best out of all of our models.

Table 5: Test Set Performance - All Balanced Models.

Model	Accuracy	Sensitivity	Specificity	Precision	F1-Score	PR-AUC	ROC-AUC
Baseline_Logistic	0.8424	0.3153	0.9696	0.7143	0.4375	0.5586	0.7737
XGB_ROSE	**0.8319**	**0.1622**	**0.9935**	**0.8571**	**0.2727**	**0.7326**	**0.733**
XGB_SMOTE_Conservative	0.8284	0.2613	0.9652	0.6444	0.3718	0.7399	0.740
RF_RandomForest_ROSE	0.8284	0.2793	0.9609	0.6327	0.3875	0.7217	0.722
XGB_SMOTE_Balanced	0.8266	0.3063	0.9522	0.6071	0.4072	0.7389	0.739
RF_Ranger_ROSE	0.8266	0.2523	0.9652	0.6364	0.3613	0.7244	0.724
SVM_ClassWeighted	0.7583	0.6126	0.7935	0.4172	0.4964	0.7419	0.742
LGB_Balanced	0.7513	0.5856	0.7913	0.4037	0.4779	0.7352	0.745
LogReg_SMOTE	0.7058	0.6486	0.7196	0.3582	0.4615	0.7303	0.730

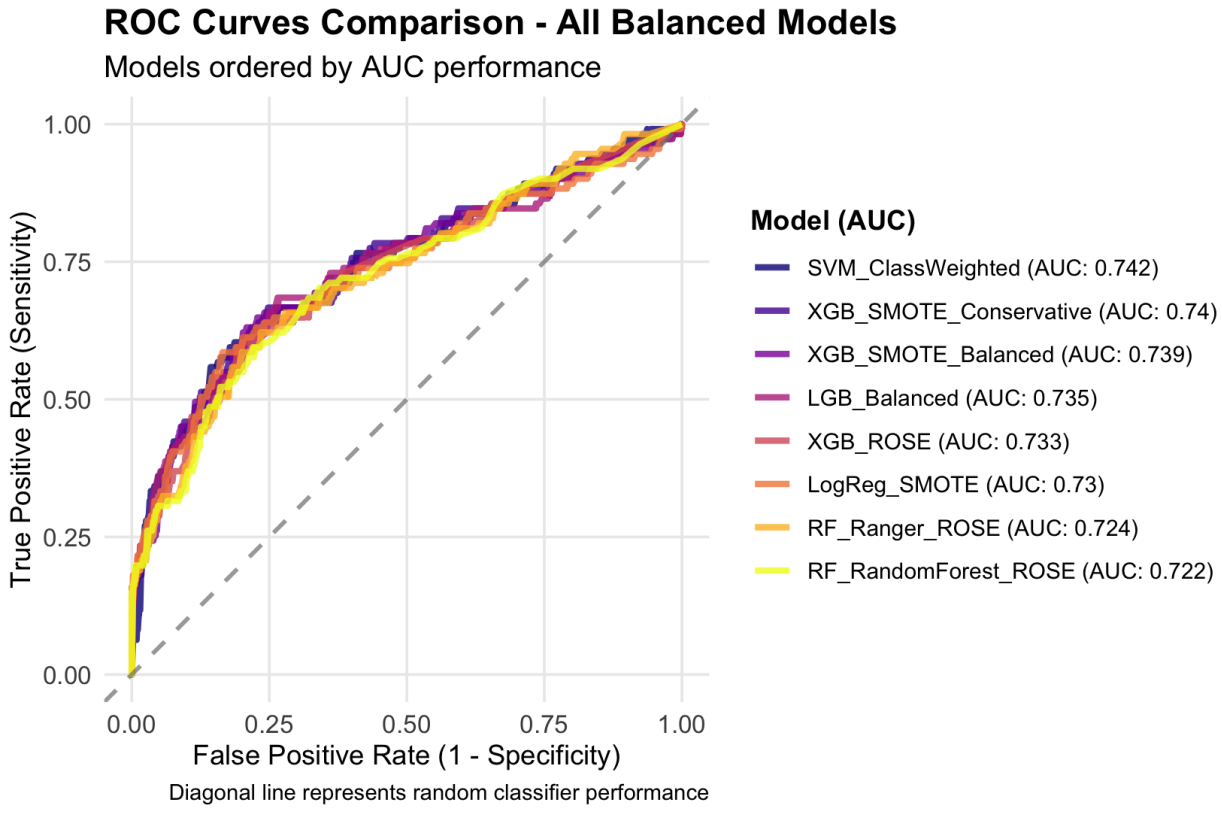

Figure 4: ROC curves for each of the models on the test data.

Table 6: Model Performance Comparison (Before Preprocessing).

Model	Accuracy	ROC-AUC
Decision Tree (CART)	**0.841**	**0.223**
Decision Tree (Conditional Inference)	0.828	0.201
Random Forest (randomForest)	0.858	0.168
Random Forest (ranger)	0.860	0.161
XGBoost	0.860	0.146
LightGBM	0.849	0.152
Neural Network	0.844	0.169
SVM	0.830	0.192

(d) Report the best model with misclassification rate and discuss about your choice.

We would select the XGB_ROSE model of all the models that we tried to fit with tuning. It was a little disappointing that when we tried to correct for class imbalance, we did not gain much on the accuracy front. What we can confidently say is that our predictions were more correct, as the models we originally fit had such bad ROC AUC that it felt inappropriate to actually consider them my final models. Table 6 shows the model beforehand I was able to boost the model specificity and sensitivity with the different models and better preprocesssing, at the expense of accuracy (1 - misclassification rate).

2. Consider a synthetic data of emulated driver telematics based on a real insurance data.

In the application, a key challenge is to model the nonlinear relationship between the aggregated amount of claims (`AMT_Claim`) and potential risk factors. The main goal of the study is to develop a model that predicts the amount of claim with significant factors using `telematics.RData`. For additional information regarding the data, refer to https://www.mdpi.com/2227-9091/9/4/58.

(a) Create your own data set to analyze the data using some filtering (e.g handling missing values and dropping observations and variables). Justify your filtering procedure.

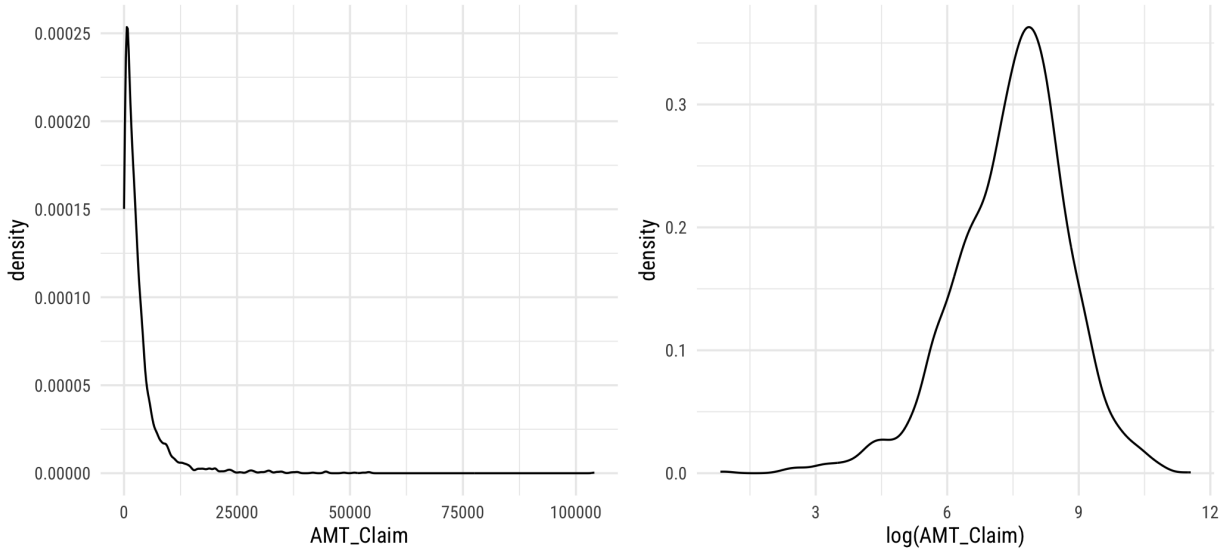

Figure 5: Kernel density estimators for the response before and after being log-transformed.

This dataset comes with 51 covariates and 1 response variable. Amongst all 3,864 rows, there is no missing data in any observation. It is peculiar because the documentation online said there was originally 70,000 rows, so we are not sure what happened to most of the observations. For our modeling purposes, we elect to log-transform the response. Figure 5 shows the effect of log-transforming our right-skewed response variable, which will help when we attempt to fit linear models later.

As part of our variable selection process, we analyze the relationship of our log-transformed response variable against the different variable types. We have four categorical predictors: `Insured.sex`, `Marital`, `Car.use`, and `Region`. Figure 6 exhibits the differences across factor levels for each categorical predictor. Although there does not seem to be substantial differences across levels for all four predictors, the Figure 6 does show log-transformed spreads, so the scales are condensed. Upon running ANOVA models for each factor, all but `Insured.sex` seem to significantly impact the log-transformed response variable. We omit this predictor from the models.

Additionally, for all numerical variables, we elect to do variable selection using a LASSO regression model. We hope to leverage the oracle properties of LASSO, as there are 47 variables that might or might not be relevant. Before doing this, we elect to systematically standardize all numeric variables, and remove variables with $|r| \geq 0.8$. Figures 7 and 8 show the different predictor correlations. We choose $\lambda = 0.00621$ by selecting the value that yields the best RMSE using 20-fold cross validation. After removing all highly correlated predictors, we are left with the following covariates with nonzero coefficients in Tables 7 and

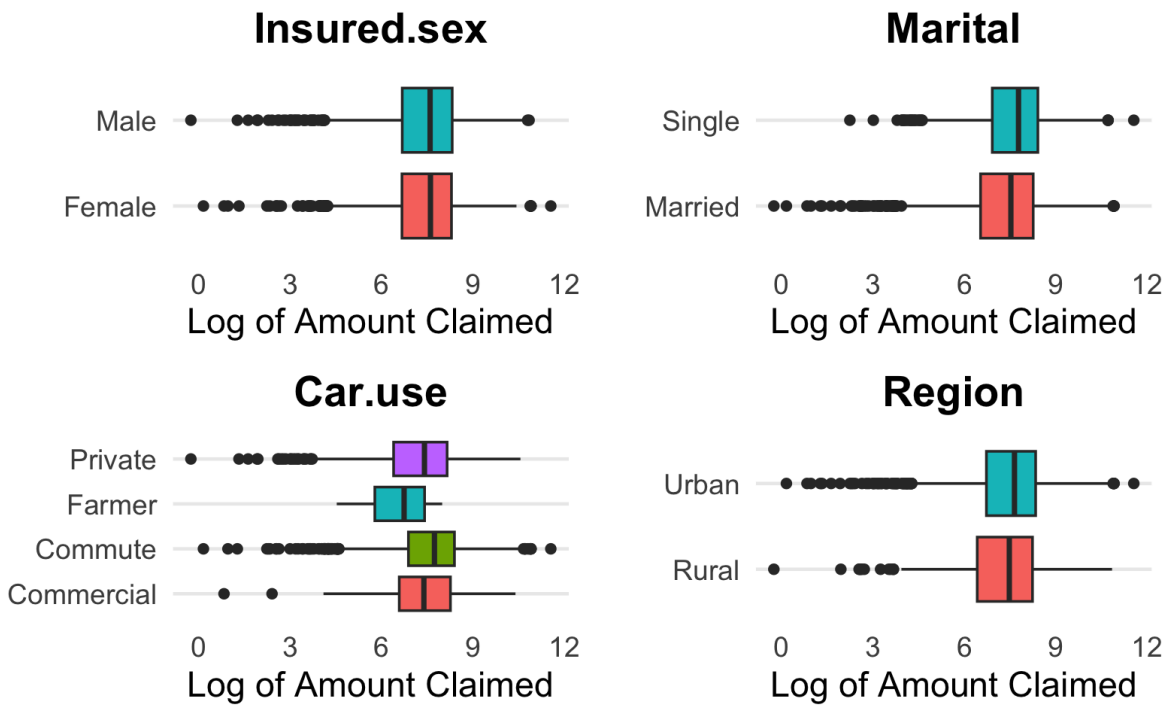

Figure 6: Boxplot for each potential predictor against the response variable `log_AMT_Claim`.

8. All of the variables in this are the variables we elect to move forward with.

(b) Explore the data graphically and numerically.

As stated before, there are no NA values in the data. Most exploratory analysis considered was in part (a), but we wanted to check for nonlinear relationships among predictors. Figures 9 and 10 reveal there are no trivial nonlinear relationships between the predictors and the response. We have our final list of predictor variables listed in Tables 7 and 8.

(c) Split your data into training (80%) and test (20%) data sets using `set.seed(1)`.

We split the data into training and testing data sets uping `set.seed(1)`.

(d) Fit the models you considered to the training data. Consider as many models as possible. If necessary, optimize the models. Report your findings. What are important predictors? What is the best model? Consider a linear regression model as well.

Data Preprocessing

We applied a comprehensive preprocessing pipeline using the `tidymodels` framework in R. Our recipe included:

- One-hot encoding of all categorical predictors using `step_dummy()`

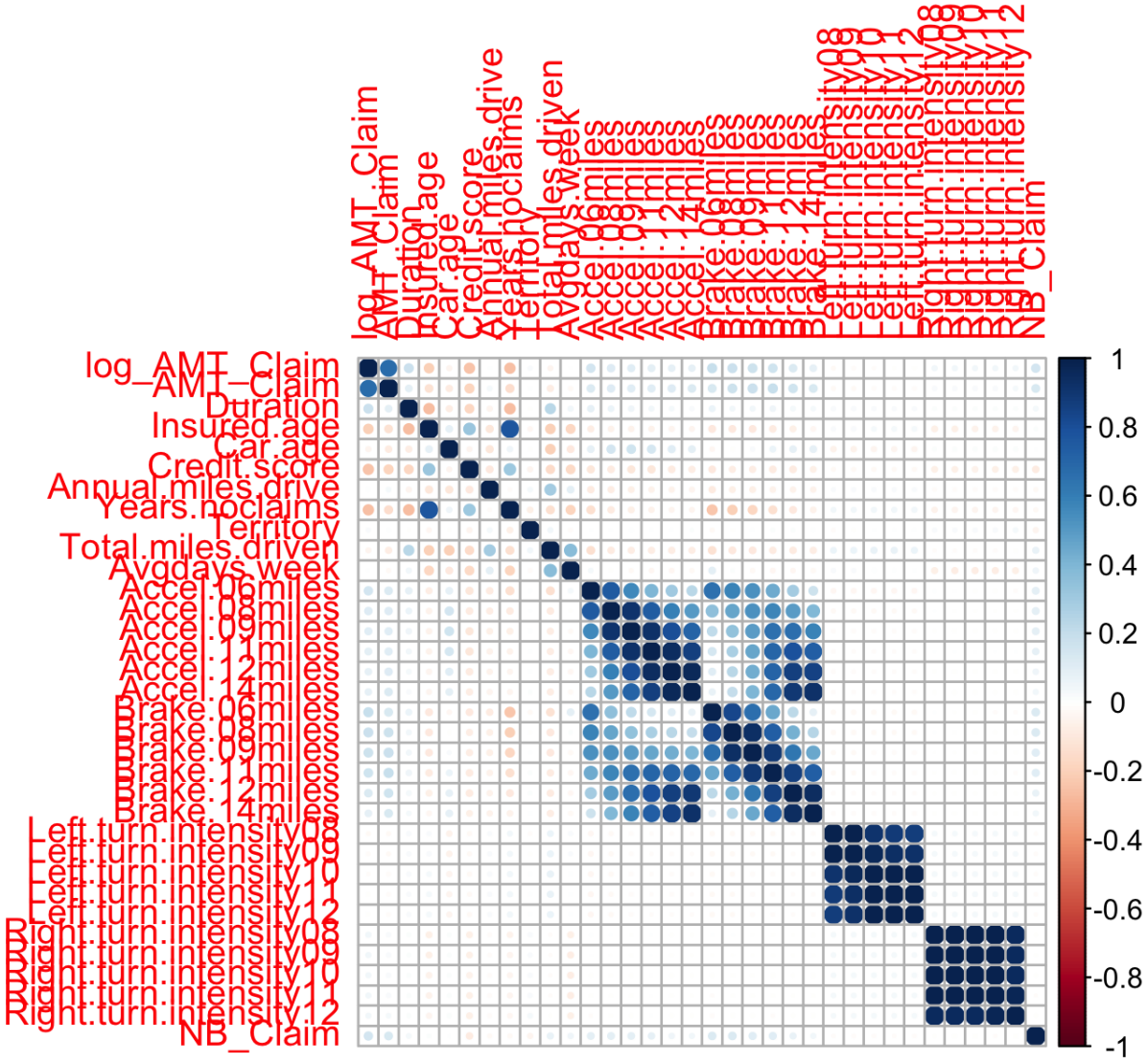

Figure 7: Correlation plot for all 34 continuous variables, not including the composition (percentage) variables.

- Removal of highly correlated numeric predictors (threshold = 0.8) using `step_corr()`
- Standardization of all numeric predictors using `step_normalize()`

The dataset was split into training (80%) and testing (20%) sets using stratified sampling to ensure representative distributions across both sets.

Model Specifications

We evaluated eight distinct modeling approaches:

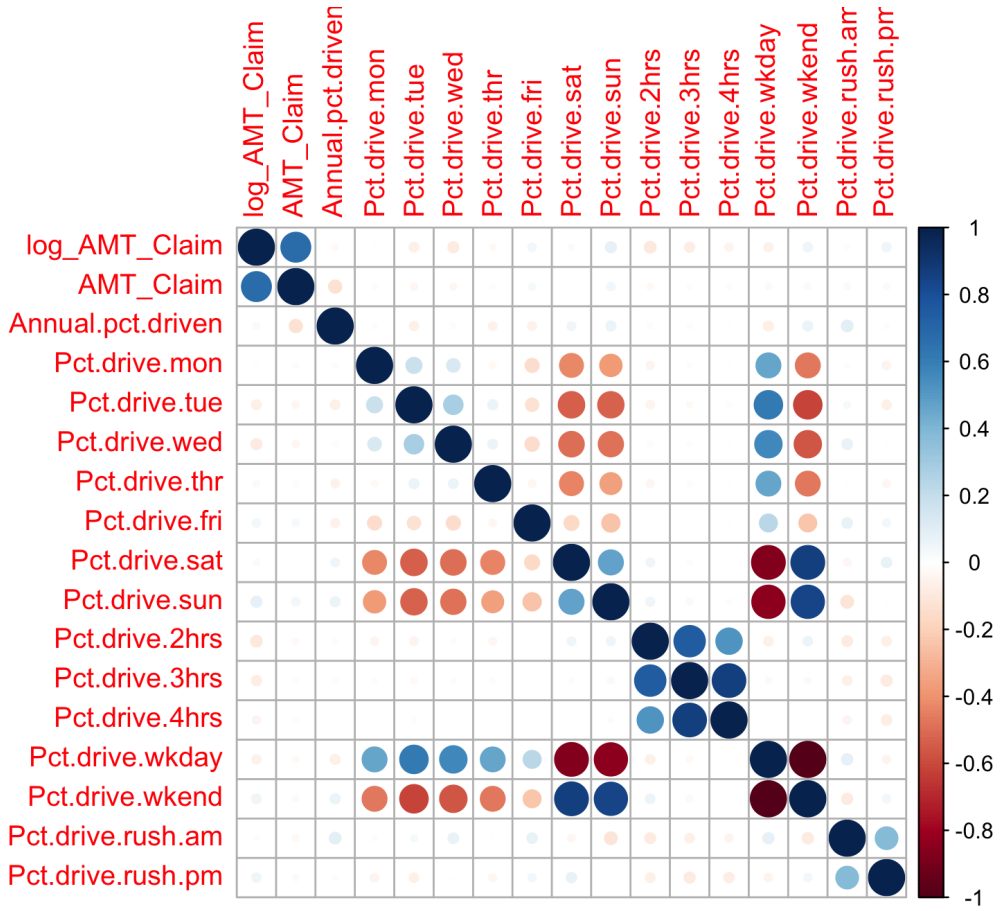

Figure 8: Correlation plot for all 15 predictor variables, not including the continuous variables.

1. **Linear Regression**: Standard ordinary least squares regression without regularization
2. **Bayesian Linear Regression**: Bayesian approach using `rstanarm` with normal priors
3. **LASSO Regression**: L1-penalized regression for feature selection
4. **Ridge Regression**: L2-penalized regression for coefficient shrinkage
5. **Elastic Net**: Combined L1 and L2 penalties
6. **Random Forest**: Ensemble method with 1000 trees
7. **XGBoost**: Gradient boosting with 1000 trees
8. **Neural Network**: Multi-layer perceptron with 100 epochs

Hyperparameter Optimization

For models requiring hyperparameter tuning (LASSO, Ridge, Elastic Net, Random Forest, XGBoost, and Neural Network), we employed 10-fold cross-validation with space-filling designs. We used RMSE as our

Table 7: LASSO Coefficients (Part 1)

Variable	Coefficient
(Intercept)	7.4581594
Duration	0.1565047
Insured.age	-0.0190262
Car.age	-0.0817743
Credit.score	-0.2161056
Annual.miles.drive	0.0402432
Years.noclaims	-0.1346743
Territory	0.0040259
Annual.pct.driven	-0.0450391
Total.miles.driven	-0.0850422
Pct.drive.mon	0.0000000
Pct.drive.tue	-0.0050843
Pct.drive.wed	-0.0455752
Pct.drive.thr	0.0141783
Pct.drive.fri	0.0412731
Pct.drive.sat	-0.0236564
Pct.drive.sun	0.0458657

Table 8: LASSO Coefficients (Part 2)

Variable	Coefficient
Pct.drive.2hrs	-0.0513224
Pct.drive.4hrs	-0.0306455
Pct.drive.rush.am	-0.0497833
Avgdays.week	-0.0755388
Accel.06miles	0.0295643
Brake.06miles	0.1115982
Brake.14miles	0.0613023
Left.turn.intensity08	-0.0161253
Right.turn.intensity12	0.0216230
NB_Claim	0.1696108
Marital_Single	0.0087664
Car.use_Commercial	-0.0425619
Car.use_Farmer	-0.0523696
Car.use_Private	-0.0460086
Region_Urban	-0.0134192

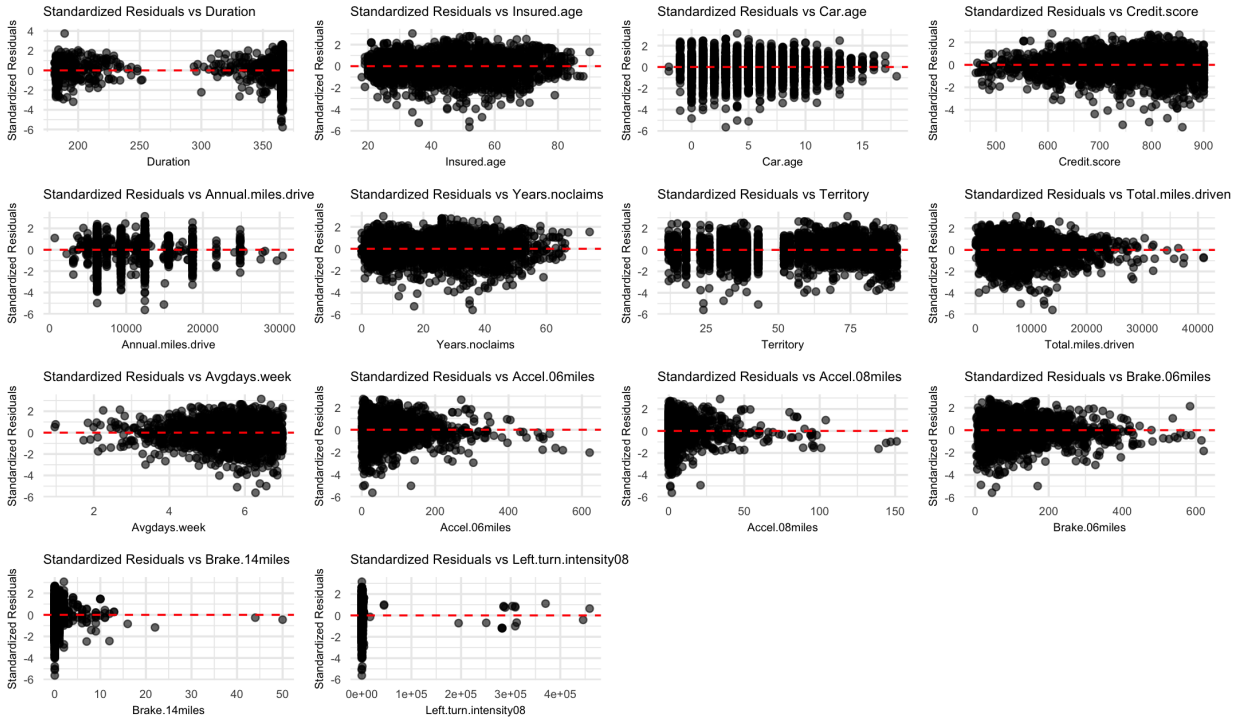

Figure 9: Standardized residual plots of simple linear regression models of all *continuous* predictors against `log_AMT_Claim`.

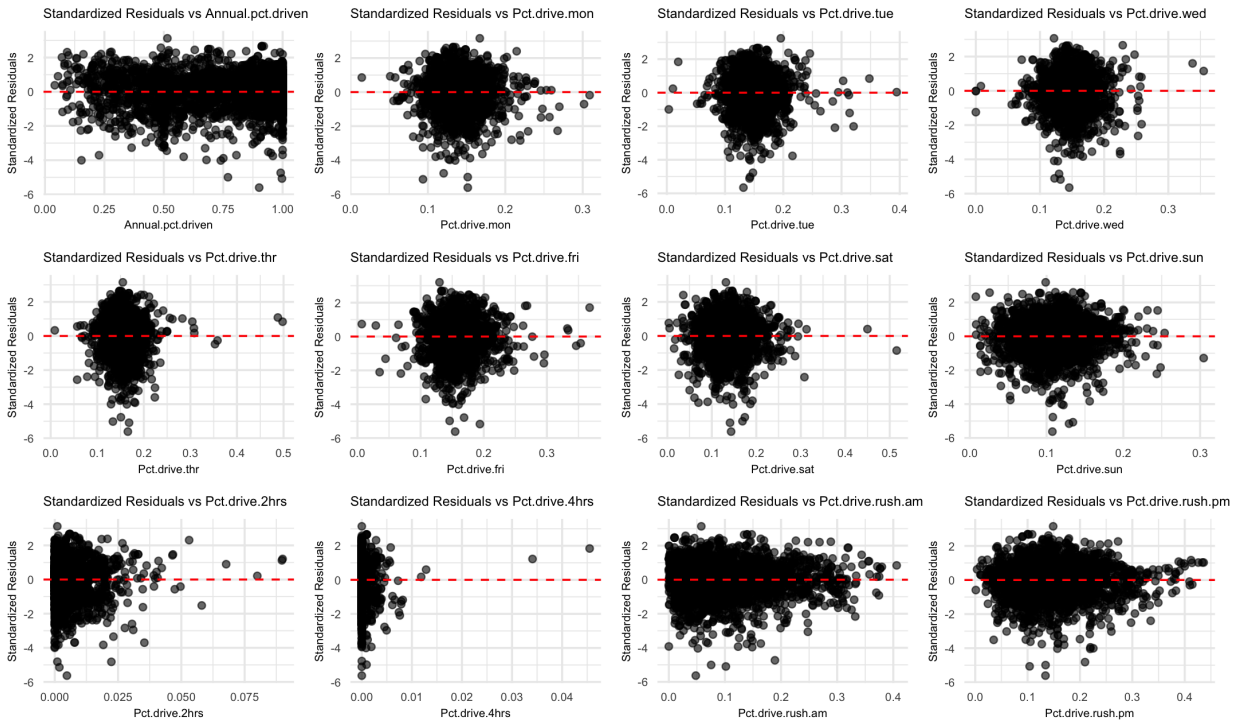

Figure 10: Standardized residual plots of simple linear regression models of all *composite* predictors against `log_AMT_Claim`.

primary optimization metric. The search spaces and grid sizes were carefully chosen to balance computational efficiency with thorough exploration of the parameter space.

Results

Optimal Hyperparameters

Table 9 presents the optimal hyperparameters identified through our tuning process. Notable findings include:

- The Ridge regression selected a very small penalty (0.000102), suggesting minimal regularization was optimal
- The Random Forest performed best with 29 features per split and minimum node size of 10
- XGBoost achieved optimal performance with moderate tree depth (4) and learning rate (1.48)
- The Neural Network required 7 hidden units with moderate regularization (penalty = 0.00464)

Table 9: Best Model Parameters

Model	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Config
LASSO	penalty: 0.0231	—	—	—	Preprocessor1_Model24
Ridge	penalty: 0.000102	—	—	—	Preprocessor1_Model01
ElasticNet	penalty: 0.0418	mixture: 0.679	—	—	Preprocessor1_Model22
Random Forest	mtry: 29	min_n: 10	—	—	Preprocessor1_Model09
XGBoost	tree_depth: 4	learn_rate: 1.48	loss_reduction: 1	—	Preprocessor1_Model04
Neural Net	hidden_units: 7	penalty: 0.00464	—	—	Preprocessor1_Model07

Feature Importance Analysis

We examined the coefficient estimates from our linear models to identify important predictors (Table 10). Key findings include:

- **AMT_Claim**: Consistently shows the strongest positive relationship across all linear models (coefficients ranging from 0.770 to 0.826)
- **Years.noclaims**: Exhibits a strong negative relationship (coefficients from -0.110 to -0.183), indicating that longer claim-free periods reduce expected claim amounts
- **Credit.score**: Shows consistent negative coefficients (-0.085 to -0.111), suggesting better credit scores are associated with lower claim amounts
- **Duration**: Positive coefficients (0.088 to 0.104) indicate longer policy durations increase expected claims
- **Brake.06miles**: Positive coefficients (0.064 to 0.079) suggest more frequent hard braking events increase claim risk

The regularized models (LASSO and Elastic Net) performed aggressive feature selection, setting many coefficients to zero. This suggests that many of the telematics variables may not be strongly predictive of claim amounts, or that multicollinearity issues are present in the dataset.

Model Performance Comparison

Table 11 presents the final model performance evaluated on the test set using the original claim amount scale (after exponentiating the log predictions). The results reveal substantial differences in predictive accuracy across our model types.

(e) Evaluate and compare the models with MSPE using the test data set. Use a multiple linear model as reference model.

In model assessment, we elected to, for simplicity's sake, use the RMSE of the model's prediction on the test set, which is the square root of the MSE of the predictions of the model on the test data. *To clarify, we predicted the logged outcome and exponentiated it, computing the error metrics to the original data.* Table 11 shows the model metrics we computed on the test predictions. The Random Forest model emerged as

Table 10: Coefficient Estimates by Regression Model

term	Linear	Bayesian	LASSO	Ridge	ElasticNet
(Intercept)	7.462	7.460	7.462	7.462	7.462
AMT_Claim	0.826	0.826	0.812	0.770	0.801
Accel.06miles	-0.003	-0.004	0.000	0.009	0.000
Accel.08miles	0.016	0.022	0.000	0.011	0.000
Annual.miles.drive	0.009	0.010	0.000	0.007	0.000
Annual.pct.driven	0.050	0.058	0.000	0.036	0.000
Avgdays.week	-0.035	-0.043	-0.011	-0.036	-0.004
Brake.06miles	0.079	0.075	0.067	0.077	0.064
Brake.14miles	-0.025	-0.025	0.000	-0.016	0.000
Car.age	-0.007	-0.016	0.000	-0.012	0.000
Car.use_Commercial	-0.027	-0.028	-0.006	-0.027	-0.001
Car.use_Farmer	-0.037	-0.030	-0.011	-0.035	-0.006
Car.use_Private	-0.023	-0.023	0.000	-0.023	0.000
Credit.score	-0.111	-0.111	-0.087	-0.111	-0.085
Duration	0.104	0.088	0.097	0.104	0.093
Insured.age	0.041	0.068	0.000	0.017	0.000
Left.turn.intensity08	-0.003	-0.008	0.000	-0.003	0.000
Marital_Single	0.039	0.064	0.018	0.036	0.015
Pct.drive.2hrs	-0.064	-0.053	-0.062	-0.062	-0.057
Pct.drive.4hrs	-0.018	-0.017	0.000	-0.020	0.000
Pct.drive.fri	0.009	1.240	0.000	0.026	0.000
Pct.drive.mon	0.003	1.095	0.000	0.017	0.000
Pct.drive.rush.am	-0.020	-0.020	0.000	-0.020	0.000
Pct.drive.rush.pm	0.031	0.035	0.000	0.028	0.000
Pct.drive.sat	-0.045	1.653	0.000	-0.017	0.000
Pct.drive.sun	NA	1.543	0.000	0.023	0.000
Pct.drive.thr	-0.001	1.178	0.000	0.016	0.000
Pct.drive.tue	-0.034	1.216	-0.001	-0.016	0.000
Pct.drive.wed	-0.068	1.153	-0.038	-0.046	-0.034
Region_Urban	-0.010	-0.012	0.000	-0.005	0.000
Territory	0.023	0.024	0.001	0.023	0.000
Total.miles.driven	-0.047	-0.050	0.000	-0.042	0.000
Years.noclaims	-0.134	-0.183	-0.112	-0.117	-0.110

Table 11: Model performance (RMSE) on the Test Data, on Original `AMT_Claim` Scale

Model	RMSE	MAE
RandomForest	174.102	21.274
XGBoost	698.272	249.638
NeuralNet	1632.803	566.014
Ridge	43717.834	4064.845
ElasticNet	59387.781	4690.243
Bayesian	64666.773	5272.209
LASSO	65203.097	5001.228
Linear	69885.385	5450.417

the clear winner, achieving remarkably superior performance with an RMSE of 174.10 and MAE of 21.27. This represents a dramatic improvement over all other approaches, with the next-best model (XGBoost) showing RMSE that is approximately 4 times higher.

Linear Model Performance

Surprisingly, all linear regression approaches (including regularized versions) performed poorly on the original scale. The standard linear regression achieved an RMSE of 69,885.39, which is approximately 400 times worse than the Random Forest. Even the regularized versions (Ridge, LASSO, Elastic Net) showed similar poor performance, suggesting that the linear assumption is fundamentally inadequate for this dataset.

The Bayesian linear regression performed similarly to its frequentist counterpart, indicating that the choice of prior had minimal impact given the available data.

Machine Learning Methods

Among the machine learning approaches, we observed a clear hierarchy:

1. **Random Forest**: Exceptional performance
2. **XGBoost**: Moderate success, though still substantially behind Random Forest
3. **Neural Network**: Poor performance, possibly due to insufficient model complexity or training

The poor performance of the neural network was unexpected and may indicate that our network architecture was too simple for the complexity of the problem, or that longer training was required.

Implications for Practice

Our results strongly suggest that traditional linear modeling approaches are inadequate for predicting insurance claim amounts using telematics data. The non-linear relationships and complex interactions present in driving behavior data require more sophisticated modeling techniques.

The Random Forest's superior performance makes it the recommended approach for this application. Its interpretability through feature importance measures also makes it practical for insurance business applications where model explanations are often required.

(f) Report the best model with MSPE and discuss about your choice.

We successfully implemented and compared eight different predictive models for insurance claim amounts using comprehensive telematics data. Our key findings include:

- Random Forest provides the best predictive accuracy by a substantial margin (RMSE = 174.10)
- Linear regression approaches, including regularized versions, are inadequate for this prediction task
- Important predictors include previous claim amounts, claim history, credit scores, and certain driving behaviors

- The complexity of telematics data requires non-linear modeling approaches to achieve acceptable predictive performance

These results have important implications for insurance pricing and risk assessment, suggesting that sophisticated machine learning methods should be preferred over traditional statistical approaches for telematics-based modeling. Although in cross validation, the neural network seemed to be the best, the Random Forest yielded the lower prediction error, so we would use this model to predict `AMT_Claim` in the future, logging the response again as we did today.

Code Appendix

```
knitr::opts_chunk$set(dev = "cairo_pdf",
                      fig.width = 5,
                      fig.height = 3.25,
                      fig.align = 'center',
                      echo = FALSE,
                      message = FALSE,
                      warning = FALSE,
                      error = FALSE,
                      cache = TRUE)
library("tidymodels")
library("patchwork")
library("glue")
library("scales")
library("extrafont")
library("tinytex")
library("patchwork")
library("gridExtra")
library("tidyr")
library("latex2exp")
library("GGally")
library("caret")
library("broom")
library("gt")
library("knitr")
library("gam")
library("splines")
library("rprojroot")
theme_set(theme_minimal(base_family = "Roboto Condensed"))

conflicted::conflicts_prefer(
  readr::col_factor(),
  purrr::discard(),
  rstan::extract(),
  dplyr::lag(),
  rstan::traceplot(),
  viridis::viridis_pal(),
  readr::parse_date
)
load(here::here("Midterm", "Storm.RData"))
load(here::here("Midterm", "telematics.RData"))

tm <- dat_1
train_data <- storm.train |>
  mutate(storm = factor(storm))
test_data <- storm.test |>
```

```r
  mutate(storm = factor(storm))

rm(dat_1)
rm(storm.train)
rm(storm.test)


# 1a ------------------------------------
# colMeans(is.na(train_data))


# storm is the response
ggpairs(train_data,
        lower = list(continuous = wrap("points", alpha = 0.1)))

numeric_vars <- names(train_data)[names(train_data) != "storm"]

# Create a function to generate boxplots
create_boxplot <- function(var_name) {
  ggplot(train_data, aes(x = storm, y = .data[[var_name]], fill = storm)) +
    geom_boxplot(alpha = 0.7) +
    labs(
      title = var_name,
      x = "Storm",
      y = var_name
    ) +
    scale_fill_manual(values = c("no" = "#3498db", "yes" = "#e74c3c")) +
    theme_minimal() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 12, face = "bold"),
      legend.position = "none"  # Remove individual legends since they're all
    ) +
    coord_flip()
}

# Apply the function to all numeric variables using map
plot_list <- map(numeric_vars, create_boxplot)

# Wrap all plots using patchwork
combined_plot <- wrap_plots(plot_list, ncol = 4)

# Display the combined plot
print(combined_plot)

train_data |>
  select(-storm) |>
  cor() |>
  corrplot::corrplot()
```

```r
mod <- glm(storm ~ ., data = train_data, family = binomial())

mod |> gtsummary::tbl_regression()

# Check if predictions are inverted (common cause of AUC < 0.5)
# Look at a simple logistic regression first
simple_glm <- glm(storm ~ ., data = train_data, family = binomial())
simple_pred <- predict(simple_glm, test_data, type = "response")

# Create confusion matrix to check prediction direction
simple_pred_class <- ifelse(simple_pred > 0.5, "yes", "no")
table(Predicted = simple_pred_class, Actual = test_data$storm)

# Check correlation between predictions and actual
cor(as.numeric(test_data$storm == "yes"), simple_pred)

# FEATURE ENGINEERING IMPROVEMENTS ====================================

# Enhanced preprocessing recipe
enhanced_recipe <- recipe(storm ~ ., data = train_data) |>
  # Remove near-zero variance predictors
  step_nzv(all_predictors()) |>
  # Handle outliers
  step_YeoJohnson(all_numeric_predictors()) |>
  # Remove highly correlated features (lower threshold)
  step_corr(all_numeric_predictors(), threshold = 0.7) |>
  # Create interaction terms for important features
  step_interact(terms = ~ DEPTH:TOP + H_MAX_Z:MAX_VIL) |>
  # Normalize after all transformations
  step_normalize(all_numeric_predictors()) |>
  # Balance classes (try different approaches)
  step_smote(storm, neighbors = 5, over_ratio = 1)

# Alternative balancing strategies
balanced_recipe_rose <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_YeoJohnson(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.7) |>
  step_normalize(all_numeric_predictors()) |>
  step_rose(storm)

balanced_recipe_downsample <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_YeoJohnson(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.7) |>
  step_normalize(all_numeric_predictors()) |>
  step_downsample(storm)
```

```r
# Class Balancing Implementation for Storm Prediction
# Carson Slater - Advanced Data Driven Methods Midterm
# Updated with comprehensive class balancing strategies
train_data <- train_data |>
  select(storm, DEPTH, TOP, H_MAX_Z, UVIL, VILD, MAX_VIL)

test_data <- test_data |>
  select(storm, DEPTH, TOP, H_MAX_Z, UVIL, VILD, MAX_VIL)

# 1. DIAGNOSTIC CHECKS ========================================================

# Check current class distribution
print("=== CLASS DISTRIBUTION ANALYSIS ===")
print("Training data class distribution:")
train_class_dist <- table(train_data$storm)
print(train_class_dist)
print("Proportions:")
print(prop.table(train_class_dist))

print("Test data class distribution:")
test_class_dist <- table(test_data$storm)
print(test_class_dist)
print("Proportions:")
print(prop.table(test_class_dist))

# Calculate imbalance ratio
imbalance_ratio <- as.numeric(train_class_dist["no"]) /
  as.numeric(train_class_dist["yes"])
print(paste("Imbalance ratio (no:yes):", round(imbalance_ratio, 2)))

# Check factor levels
print("Factor levels:")
print(paste("Training:", paste(levels(train_data$storm), collapse = ", ")))
print(paste("Test:", paste(levels(test_data$storm), collapse = ", ")))

# 2. MULTIPLE PREPROCESSING STRATEGIES ========================================

# Strategy 1: SMOTE with different parameters
recipe_smote_conservative <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_smote(storm, neighbors = 5, over_ratio = 0.8, seed = 613)

recipe_smote_balanced <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
```

```
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_smote(storm, neighbors = 5, over_ratio = 1.0, seed = 613)

recipe_smote_aggressive <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_smote(storm, neighbors = 3, over_ratio = 1.2, seed = 613)

# Strategy 2: ROSE (Random Over-Sampling Examples)
recipe_rose <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_rose(storm, over_ratio = 1.0, seed = 613)

# Strategy 3: Downsampling majority class
recipe_downsample <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_downsample(storm, under_ratio = 1.0, seed = 613)

# Strategy 4: Combination of upsampling and downsampling
recipe_combined <- recipe(storm ~ ., data = train_data) |>
  step_nzv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_upsample(storm, over_ratio = 0.7, seed = 613) |>
  step_downsample(storm, under_ratio = 1.2, seed = 614)


# 3. CLASS-WEIGHTED MODEL SPECIFICATIONS ==================================

# XGBoost with class balancing
xgb_balanced_spec <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune(),
  learn_rate = tune()
) |>
  set_engine(
    "xgboost",
    scale_pos_weight = imbalance_ratio,
```

```
    objective = "binary:logistic",
    eval_metric = "auc",
    max_delta_step = 1
  ) |> # Helps with imbalanced data
  set_mode("classification")

# Random Forest with class weights (randomForest engine)
rf_balanced_rf_spec <- rand_forest(
  mtry = tune(),
  trees = 500,
  min_n = tune()
) |>
  set_engine("randomForest", classwt = c("no" = 1, "yes" = imbalance_ratio)) |>
  set_mode("classification")

# Random Forest with class weights (ranger engine)
rf_balanced_ranger_spec <- rand_forest(
  mtry = tune(),
  trees = 500,
  min_n = tune()
) |>
  set_engine(
    "ranger",
    class.weights = c("no" = 1, "yes" = imbalance_ratio),
    importance = "impurity"
  ) |>
  set_mode("classification")

# SVM with class weights
svm_balanced_spec <- svm_rbf(
  cost = tune(),
  rbf_sigma = tune()
) |>
  set_engine("kernlab", class.weights = c("no" = 1, "yes" = imbalance_ratio)) |>
  set_mode("classification")

# Logistic Regression with class weights
log_reg_balanced_spec <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) |>
  set_engine("glmnet") |>
  set_mode("classification")

# LightGBM with class balancing
lgb_balanced_spec <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
```

```
  min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune(),
  learn_rate = tune()
) |>
  set_engine(
    "lightgbm",
    objective = "binary",
    is_unbalance = TRUE,
    boost_from_average = FALSE
  ) |>
  set_mode("classification")

# 4. CREATE WORKFLOWS WITH DIFFERENT STRATEGIES ============================

# Create workflows combining different recipes with balanced models
workflows_list <- list(
  # SMOTE variations with XGBoost
  "XGB_SMOTE_Conservative" = workflow() |>
    add_recipe(recipe_smote_conservative) |>
    add_model(xgb_balanced_spec),

  "XGB_SMOTE_Balanced" = workflow() |>
    add_recipe(recipe_smote_balanced) |>
    add_model(xgb_balanced_spec),

  "XGB_SMOTE_Aggressive" = workflow() |>
    add_recipe(recipe_smote_aggressive) |>
    add_model(xgb_balanced_spec),

  # ROSE with different models
  "XGB_ROSE" = workflow() |>
    add_recipe(recipe_rose) |>
    add_model(xgb_balanced_spec),

  "RF_RandomForest_ROSE" = workflow() |>
    add_recipe(recipe_rose) |>
    add_model(rf_balanced_rf_spec),

  "RF_Ranger_ROSE" = workflow() |>
    add_recipe(recipe_rose) |>
    add_model(rf_balanced_ranger_spec),

  # Downsampling approaches
  "XGB_Downsample" = workflow() |>
    add_recipe(recipe_downsample) |>
    add_model(xgb_balanced_spec),
```

```
  "RF_RandomForest_Downsample" = workflow() |>
    add_recipe(recipe_downsample) |>
    add_model(rf_balanced_rf_spec),

  # Combined sampling
  "RF_Ranger_Combined" = workflow() |>
    add_recipe(recipe_combined) |>
    add_model(rf_balanced_ranger_spec),

  # Class-weighted models with minimal preprocessing
  "SVM_ClassWeighted" = workflow() |>
    add_recipe(
      recipe(storm ~ ., data = train_data) |>
        step_normalize(all_numeric_predictors()) |>
        step_corr(all_numeric_predictors(), threshold = 0.8)
    ) |>
    add_model(svm_balanced_spec),

  "LGB_Balanced" = workflow() |>
    add_recipe(recipe_smote_balanced) |>
    add_model(lgb_balanced_spec),

  "LogReg_SMOTE" = workflow() |>
    add_recipe(recipe_smote_balanced) |>
    add_model(log_reg_balanced_spec)
)

# 5. PARAMETER GRIDS FOR BALANCED MODELS ===================================

xgb_balanced_grid <- grid_space_filling(
  trees(range = c(100, 500)),
  tree_depth(range = c(3, 6)),
  min_n(range = c(2, 20)),
  loss_reduction(range = c(-1, 0.5)),
  sample_prop(range = c(0.7, 1.0)),
  mtry(range = c(3, 5)),
  learn_rate(range = c(-3, -2)),
  size = 8
)

rf_balanced_grid <- grid_space_filling(
  mtry(range = c(3, 5)),
  min_n(range = c(2, 10)),
  size = 8
)

log_reg_balanced_grid <- grid_space_filling(
```

```
  penalty(range = c(-4, -1)),
  mixture(range = c(0.1, 0.9)),
  size = 8
)

svm_balanced_grid <- grid_space_filling(
  cost(range = c(0, 2)),
  rbf_sigma(range = c(-3, 0)),
  size = 8
)

log_reg_balanced_grid <- grid_space_filling(
  penalty(range = c(-4, -1)),
  mixture(range = c(0.1, 0.9)),
  size = 8
)

lgb_balanced_grid <- grid_space_filling(
  trees(range = c(100, 500)),
  tree_depth(range = c(3, 6)),
  min_n(range = c(2, 20)),
  loss_reduction(range = c(-1, 0.5)),
  sample_prop(range = c(0.7, 1.0)),
  mtry(range = c(3, 5)),
  learn_rate(range = c(-3, -2)),
  size = 8
)


# 6. ENHANCED EVALUATION METRICS ============================================

# Custom metric set for imbalanced classification
imbalanced_metrics <- metric_set(
  roc_auc,
  pr_auc, # Precision-Recall AUC
  sensitivity, # Recall for positive class
  specificity, # Recall for negative class
  precision, # Precision for positive class
  f_meas, # F1 score
  bal_accuracy, # Balanced accuracy
  kap # Cohen's Kappa
)

# 7. CROSS-VALIDATION SETUP ================================================

# Stratified CV with more folds for better estimates
cv_folds_balanced <- vfold_cv(train_data, v = 10, strata = storm, repeats = 2)
```

```r
# 8. MODEL TUNING FUNCTION ================================================

tune_balanced_model <- function(
  workflow_name,
  workflow_obj,
  param_grid,
  cv_folds
) {
  cat("Tuning:", workflow_name, "\n")

  tryCatch(
    {
      tune_results <- tune_grid(
        workflow_obj,
        resamples = cv_folds,
        grid = param_grid,
        metrics = imbalanced_metrics,
        control = control_grid(
          verbose = TRUE,
          save_pred = TRUE,
          parallel_over = "everything"
        )
      )
      return(tune_results)
    },
    error = function(e) {
      message("Error tuning ", workflow_name, ": ", e$message)
      return(NULL)
    }
  )
}


# 9. EXECUTE TUNING FOR SELECTED MODELS ===================================

# Test serially
# test_result <- pmap(
#   list(tuning_tbl$id[1], tuning_tbl$workflow[1], tuning_tbl$grid[1]),
#   ~ tune_balanced_model(..1, ..2, ..3, cv_folds_balanced)
# )

# test_result

print("=== STARTING BALANCED MODEL TUNING ===")


# Tuning specifications
tuning_tbl <- tibble::tibble(
```

```
  id = c(
    "XGB_SMOTE_Conservative",
    "XGB_SMOTE_Balanced",
    "XGB_ROSE",
    "RF_RandomForest_ROSE",
    "RF_Ranger_ROSE",
    "SVM_ClassWeighted",
    "LGB_Balanced",
    "LogReg_SMOTE"
  ),
  workflow = workflows_list[c(
    "XGB_SMOTE_Conservative",
    "XGB_SMOTE_Balanced",
    "XGB_ROSE",
    "RF_RandomForest_ROSE",
    "RF_Ranger_ROSE",
    "SVM_ClassWeighted",
    "LGB_Balanced",
    "LogReg_SMOTE"
  )],
  grid = list(
    xgb_balanced_grid,
    xgb_balanced_grid,
    xgb_balanced_grid,
    rf_balanced_grid,
    rf_balanced_grid,
    svm_balanced_grid,
    lgb_balanced_grid,
    log_reg_balanced_grid
  )
)

# Run tuning in parallel with progress bar
# tuning_results <- pmap(
#   list(tuning_tbl$id, tuning_tbl$workflow, tuning_tbl$grid),
#   ~ tune_balanced_model(..1, ..2, ..3, cv_folds_balanced)
# )

# Name results

# Save results
# write_rds(
#   tuning_results,
#   here::here("Midterm", "balanced_model_tuning_res2.rds")
# )

tuning_results <- read_rds(here::here(
  "Midterm",
```

```
  "balanced_model_tuning_res.rds"
))
names(tuning_results) <- tuning_tbl$`id`

# 10. EXTRACT BEST MODELS ======================================================

extract_best_results <- function(tune_result, model_name) {
  # Get best by ROC-AUC
  best_roc <- select_best(tune_result, metric = "roc_auc")

  # Get best by PR-AUC
  best_pr <- select_best(tune_result, metric = "pr_auc")

  # Get best by F1 score
  best_f1 <- select_best(tune_result, metric = "f_meas")

  # Get best by Balanced Accuracy
  best_acc <- select_best(tune_result, metric = "bal_accuracy")

  # Show metrics for best models
  roc_metrics <- show_best(tune_result, metric = "roc_auc", n = 1)
  pr_metrics <- show_best(tune_result, metric = "pr_auc", n = 1)
  f1_metrics <- show_best(tune_result, metric = "f_meas", n = 1)
  acc_metrics <- show_best(tune_result, metric = "bal_accuracy", n = 1)

  return(list(
    model_name = model_name,
    best_roc = best_roc,
    best_pr = best_pr,
    best_f1 = best_f1,
    best_acc = best_acc,
    roc_metrics = roc_metrics,
    pr_metrics = pr_metrics,
    f1_metrics = f1_metrics,
    acc_metrics = acc_metrics
  ))
}


# Extract results for all models
best_results <- map2(
  tuning_results,
  names(tuning_results),
  extract_best_results
)

best_results_acc <- map(best_results, ~ .x$best_acc)
```

```r
# Define the data frame with best hyperparameters
hyperparameters_table <- data.frame(
  Model = c(
    "XGBoost (SMOTE Conservative)",
    "XGBoost (SMOTE Balanced)",
    "XGBoost (ROSE)",
    "Random Forest (randomForest ROSE)",
    "Random Forest (ranger ROSE)",
    "SVM (Class Weighted)",
    "LightGBM (Balanced)",
    "Logistic Regression (SMOTE)"
  ),
  Parameters = c(
    paste(
      names(best_results$XGB_SMOTE_Conservative$best_acc)[1:7],
      round(best_results$XGB_SMOTE_Conservative$best_acc[1, 1:7], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$XGB_SMOTE_Balanced$best_acc)[1:7],
      round(best_results$XGB_SMOTE_Balanced$best_acc[1, 1:7], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$XGB_ROSE$best_acc)[1:7],
      round(best_results$XGB_ROSE$best_acc[1, 1:7], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$RF_RandomForest_ROSE$best_acc)[1:2],
      round(best_results$RF_RandomForest_ROSE$best_acc[1, 1:2], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$RF_Ranger_ROSE$best_acc)[1:2],
      round(best_results$RF_Ranger_ROSE$best_acc[1, 1:2], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$SVM_ClassWeighted$best_acc)[1:2],
      round(best_results$SVM_ClassWeighted$best_acc[1, 1:2], 3),
      sep = ": ",
      collapse = ", "
```

```
    ),
    paste(
      names(best_results$LGB_Balanced$best_acc)[1:7],
      round(best_results$LGB_Balanced$best_acc[1, 1:7], 3),
      sep = ": ",
      collapse = ", "
    ),
    paste(
      names(best_results$LogReg_SMOTE$best_acc)[1:2],
      round(best_results$LogReg_SMOTE$best_acc[1, 1:2], 3),
      sep = ": ",
      collapse = ", "
    )
  )
)

# Generate the table
hyperparameters_table |>
  kable(
    caption = "Best Hyperparameters for Each Model",
    col.names = c("Model", "Best Parameters"),
    align = c("l", "l"),
    format = "latex"
  ) |>
  kable_styling(
    bootstrap_options = c("striped", "hover", "condensed", "responsive"),
    full_width = TRUE,
    position = "center"
  ) |>
  column_spec(2, width = "60%")

# 11. PERFORMANCE COMPARISON TABLE ========================================

create_performance_summary <- function(best_results) {
  summary_df <- map_dfr(best_results, function(x) {
    data.frame(
      Model = x$model_name,
      Accuracy = round(x$acc_metrics$mean, 4),
      Accuracy_Se = round(x$acc_metrics$std_err, 4),
      ROC_AUC = round(x$roc_metrics$mean, 4),
      ROC_AUC_SE = round(x$roc_metrics$std_err, 4),
      PR_AUC = round(x$pr_metrics$mean, 4),
      PR_AUC_SE = round(x$pr_metrics$std_err, 4),
      F1_Score = round(x$f1_metrics$mean, 4),
      F1_SE = round(x$f1_metrics$std_err, 4),
      stringsAsFactors = FALSE
    )
  })
```

```
  summary_df |>
    arrange(desc(ROC_AUC)) |>
    kable(
      caption = "Balanced Model Performance Comparison",
      col.names = c(
        "Model",
        "Accuracy",
        "SE",
        "ROC-AUC",
        "SE",
        "PR-AUC",
        "SE",
        "F1-Score",
        "SE"
      ),
      align = c("l", rep("c", 6)),
      format = "latex"
    ) |>
    kable_styling(
      bootstrap_options = c("striped", "hover", "condensed", "responsive"),
      full_width = FALSE
    ) |>
    add_header_above(c(
      " " = 1,
      "Accuracy" = 2,
      "ROC-AUC" = 2,
      "PR-AUC" = 2,
      "F1-Score" = 2
    ))
}

performance_summary <- create_performance_summary(best_results)
create_performance_summary(best_results)
print(performance_summary)

# 12. FINAL MODEL TRAINING AND EVALUATION ================================

# Select best performing model based on ROC-AUC
best_model_name <- best_results[[1]]$model_name
best_workflow <- workflows_list[[best_model_name]]
best_params <- best_results[[1]]$best_roc

# Finalize and fit the best model
final_balanced_workflow <- finalize_workflow(best_workflow, best_params)
final_balanced_fit <- fit(final_balanced_workflow, train_data)

# Evaluate on test set
```

```r
test_predictions <- predict(final_balanced_fit, test_data, type = "prob") |>
  bind_cols(predict(final_balanced_fit, test_data)) |>
  bind_cols(test_data |> select(storm))

metric_fn <- metric_set(accuracy, roc_auc)

test_metrics <- test_predictions |>
  metric_fn(
    truth = storm,
    estimate = .pred_class,
    .pred_yes,
    event_level = "second" # ensure "yes" is positive class
  )

print("=== FINAL TEST SET PERFORMANCE ===")
print(test_metrics)

# ROC and PR curves for final model
roc_data <- test_predictions |>
  roc_curve(truth = storm, .pred_yes, event_level = "second")

pr_data <- test_predictions |>
  pr_curve(truth = storm, .pred_yes, event_level = "second")

# Plot final results
roc_plot <- roc_data |>
  ggplot(aes(x = 1 - specificity, y = sensitivity)) +
  geom_line(color = "red") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(
    title = paste("ROC Curve -", best_model_name),
    subtitle = paste(
      "Test ROC-AUC =",
      round(
        roc_auc_vec(
          test_predictions$storm,
          test_predictions$.pred_yes,
          event_level = "second"
        ),
        3
      )
    )
  ) +
  theme_minimal()

pr_plot <- pr_data |>
  ggplot(aes(x = recall, y = precision)) +
  geom_line(size = 1.2, color = "red") +
```

```
  labs(
    title = paste("Precision-Recall Curve -", best_model_name),
    subtitle = paste(
      "Test PR-AUC =",
      round(
        pr_auc_vec(
          test_predictions$storm,
          test_predictions$.pred_yes,
          event_level = "second"
        ),
        3
      )
    )
  ) +
  theme_minimal()

print(roc_plot)
print(pr_plot)

# Variable importance for final model
if (str_detect(best_model_name, "XGB|RF|LGB")) {
  vip_plot <- final_balanced_fit |>
    extract_fit_parsnip() |>
    vip(num_features = 10) +
    ggtitle(paste("Variable Importance -", best_model_name))
  print(vip_plot)
}

# Save results
save(
  tuning_results,
  best_results,
  final_balanced_fit,
  test_predictions,
  file = here::here("Midterm", "balanced_models_results.RData")
)

print("=== CLASS BALANCING IMPLEMENTATION COMPLETE ===")
print("Check the performance summary table above for improvements!")


# Final Model Testing - All Balanced Models
# Carson Slater - Advanced Data Driven Methods Midterm
# Testing all finalized balanced models on the test dataset

# Load your tuning results if not already in environment
# tuning_results <- read_rds(here::here("Midterm", "balanced_model_tuning_res
# names(tuning_results) <- tuning_tbl$id
```

```r
print("=== FINALIZING AND TESTING ALL BALANCED MODELS ===")

# 1. FINALIZE ALL WORKFLOWS WITH BEST PARAMETERS ========================

finalize_and_fit_model <- function(
  model_name,
  workflow_obj,
  tune_result,
  train_data
) {
  cat("Finalizing and fitting:", model_name, "\n")

  tryCatch(
    {
      # Get best parameters based on ROC-AUC
      best_params <- select_best(tune_result, metric = "roc_auc")

      # Finalize workflow
      final_workflow <- finalize_workflow(workflow_obj, best_params)

      # Fit on training data
      final_fit <- fit(final_workflow, train_data)

      return(list(
        model_name = model_name,
        workflow = final_workflow,
        fit = final_fit,
        best_params = best_params
      ))
    },
    error = function(e) {
      message("Error fitting ", model_name, ": ", e$message)
      return(NULL)
    }
  )
}

# Finalize all models
final_models <- pmap(
  list(
    names(tuning_results),
    workflows_list[names(tuning_results)],
    tuning_results
  ),
  ~ finalize_and_fit_model(
    model_name = ..1,
    workflow_obj = ..2,
```

```
    tune_result = ..3,
    train_data = train_data
  )
)

# Remove any NULL results (failed fits)
final_models <- compact(final_models)
names(final_models) <- map_chr(final_models, ~ .x$model_name)

print(paste("Successfully fitted", length(final_models), "models"))

# 2. GENERATE PREDICTIONS ON TEST SET ====================================

generate_test_predictions <- function(final_model_obj, test_data) {
  model_name <- final_model_obj$model_name

  tryCatch(
    {
      # Generate predictions
      predictions <- predict(final_model_obj$fit, test_data, type = "prob") |>
        bind_cols(predict(final_model_obj$fit, test_data)) |>
        bind_cols(test_data |> select(storm)) |>
        mutate(model = model_name)

      return(predictions)
    },
    error = function(e) {
      message("Error predicting for ", model_name, ": ", e$message)
      return(NULL)
    }
  )
}

# Generate predictions for all models
all_predictions <- map(final_models, ~ generate_test_predictions(.x, test_data
all_predictions <- compact(all_predictions)

print(paste("Generated predictions for", length(all_predictions), "models"))

# 3. CALCULATE TEST SET METRICS FOR ALL MODELS ============================

calculate_test_metrics <- function(predictions) {
  model_name <- unique(predictions$model)

  # Define comprehensive metric set
  test_metrics <- metric_set(
    accuracy,
    roc_auc,
```

```
    pr_auc,
    sensitivity,
    specificity,
    precision,
    f_meas,
    bal_accuracy,
    kap
  )

  # Calculate metrics
  metrics_result <- predictions |>
    test_metrics(
      truth = storm,
      estimate = .pred_class,
      .pred_yes,
      event_level = "second"
    ) |>
    mutate(model = model_name)

  return(metrics_result)
}

# Calculate metrics for all models
all_test_metrics <- map_dfr(all_predictions, calculate_test_metrics)

# 4. CREATE COMPREHENSIVE RESULTS SUMMARY ================================

# Create summary table
test_performance_summary <- all_test_metrics |>
  select(model, .metric, .estimate) |>
  pivot_wider(names_from = .metric, values_from = .estimate) |>
  arrange(desc(accuracy)) |>
  mutate(across(where(is.numeric), ~ round(.x, 4)))

print("=== TEST SET PERFORMANCE SUMMARY ===")
print(test_performance_summary)

# Create formatted table
library(knitr)
library(kableExtra)

test_performance_summary |>
  select(-c("bal_accuracy", "kap")) |>
  kable(
    caption = "Test Set Performance - All Balanced Models",
    col.names = c(
      "Model",
      "Accuracy",
```

```
      "Sensitivity",
      "Specificity",
      "Precision",
      "F1-Score",
      "PR-AUC",
      "ROC-AUC"
    ),
    align = c("l", rep("c", 9)),
    format = "latex"
  ) |>
  kable_styling(
    bootstrap_options = c("striped", "hover", "condensed", "responsive"),
    full_width = FALSE
  ) |>
  row_spec(1, bold = TRUE, background = "#d4edda") # Highlight best ROC-AUC

print(formatted_summary)

# 5. DETAILED PERFORMANCE ANALYSIS ====================================

# Find best performing models by different metrics
best_by_roc <- test_performance_summary |>
  slice_max(roc_auc, n = 1) |>
  pull(model)

best_by_pr <- test_performance_summary |>
  slice_max(pr_auc, n = 1) |>
  pull(model)

best_by_f1 <- test_performance_summary |>
  slice_max(f_meas, n = 1) |>
  pull(model)

best_by_accuracy <- test_performance_summary |>
  slice_max(accuracy, n = 1) |>
  pull(model)

cat("\n=== BEST MODELS BY METRIC ===\n")
cat(
  "Best ROC-AUC:",
  best_by_roc,
  "=",
  round(
    test_performance_summary$roc_auc[
      test_performance_summary$model == best_by_roc
    ],
    4
  ),
```

```
  "\n"
)
cat(
  "Best PR-AUC:",
  best_by_pr,
  "=",
  round(
    test_performance_summary$pr_auc[
      test_performance_summary$model == best_by_pr
    ],
    4
  ),
  "\n"
)
cat(
  "Best F1-Score:",
  best_by_f1,
  "=",
  round(
    test_performance_summary$f_meas[
      test_performance_summary$model == best_by_f1
    ],
    4
  ),
  "\n"
)
cat(
  "Best Accuracy:",
  best_by_accuracy,
  "=",
  round(
    test_performance_summary$accuracy[
      test_performance_summary$model == best_by_accuracy
    ],
    4
  ),
  "\n"
)

# 6. CONFUSION MATRICES FOR TOP MODELS ===================================

create_confusion_matrix <- function(predictions, model_name) {
  cm <- predictions |>
    conf_mat(truth = storm, estimate = .pred_class)

  # Create a nice visualization
  cm_plot <- cm |>
    autoplot(type = "heatmap") +
```

```
    ggtitle(paste("Confusion Matrix:", model_name)) +
    theme_minimal()

  return(list(matrix = cm, plot = cm_plot))
}

# Create confusion matrices for top 3 models by ROC-AUC
top_models <- test_performance_summary |>
  slice_max(roc_auc, n = 3) |>
  pull(model)

confusion_matrices <- map(top_models, function(model_name) {
  preds <- all_predictions[[model_name]]
  create_confusion_matrix(preds, model_name)
})

names(confusion_matrices) <- top_models

# Print confusion matrices
for (model_name in names(confusion_matrices)) {
  cat("\n=== CONFUSION MATRIX:", model_name, "===\n")
  print(confusion_matrices[[model_name]]$matrix)
  print(confusion_matrices[[model_name]]$plot)
}

# 7. ROC AND PR CURVES COMPARISON ========================================

# Combine all predictions for plotting
combined_predictions <- bind_rows(all_predictions)

# ROC curves comparison
roc_comparison <- combined_predictions |>
  group_by(model) |>
  roc_curve(truth = storm, .pred_yes, event_level = "second") |>
  ggplot(aes(x = 1 - specificity, y = sensitivity, color = model)) +
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", alpha = 0.7) +
  labs(
    title = "ROC Curves Comparison - All Balanced Models",
    x = "1 - Specificity (False Positive Rate)",
    y = "Sensitivity (True Positive Rate)",
    color = "Model"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(ncol = 2))

print(roc_comparison)
```

```r
# PR curves comparison
pr_comparison <- combined_predictions |>
  group_by(model) |>
  pr_curve(truth = storm, .pred_yes, event_level = "second") |>
  ggplot(aes(x = recall, y = precision, color = model)) +
  geom_line(size = 1) +
  labs(
    title = "Precision-Recall Curves Comparison - All Balanced Models",
    x = "Recall (Sensitivity)",
    y = "Precision",
    color = "Model"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(ncol = 2))

print(pr_comparison)

# 8. FEATURE IMPORTANCE FOR TREE-BASED MODELS ============================

# Extract variable importance for tree-based models
extract_importance <- function(final_model_obj) {
  model_name <- final_model_obj$model_name

  # Check if it's a tree-based model
  if (str_detect(model_name, "XGB|RF|LGB")) {
    tryCatch(
      {
        importance_data <- final_model_obj$fit |>
          extract_fit_parsnip() |>
          vi() |>
          mutate(model = model_name)
        return(importance_data)
      },
      error = function(e) {
        message("Could not extract importance for ", model_name)
        return(NULL)
      }
    )
  }
  return(NULL)
}

# Get importance for all applicable models
importance_results <- map(final_models, extract_importance)
importance_results <- compact(importance_results)
```

```r
if (length(importance_results) > 0) {
  # Combine and plot
  combined_importance <- bind_rows(importance_results)

  importance_plot <- combined_importance |>
    group_by(model) |>
    slice_max(Importance, n = 6) |>
    ggplot(aes(
      x = reorder_within(Variable, Importance, model),
      y = Importance,
      fill = model
    )) +
    geom_col() +
    facet_wrap(~model, scales = "free_y", ncol = 2) +
    coord_flip() +
    scale_x_reordered() +
    labs(
      title = "Variable Importance - Tree-Based Models",
      x = "Variables",
      y = "Importance"
    ) +
    theme_minimal() +
    theme(legend.position = "none")

  print(importance_plot)
}

# 9. SAVE COMPREHENSIVE RESULTS ============================================

# Save all results
final_results <- list(
  models = final_models,
  predictions = all_predictions,
  metrics = all_test_metrics,
  performance_summary = test_performance_summary,
  confusion_matrices = confusion_matrices,
  best_models = list(
    roc_auc = best_by_roc,
    pr_auc = best_by_pr,
    f1_score = best_by_f1,
    accuracy = best_by_accuracy
  )
)

# Save to file
# save(final_results, file = here::here("Midterm", "final_balanced_models_tes

print("\n=== FINAL TESTING COMPLETE ===")
```

```r
print("All models have been fitted and tested!")
print("Results saved to: final_balanced_models_test_results.RData")

# 10. FINAL RECOMMENDATIONS ================================================

cat("\n=== FINAL RECOMMENDATIONS ===\n")
cat("Based on test set performance:\n")
cat("1. Best overall model (ROC-AUC):", best_by_roc, "\n")
cat("2. Best for precision/recall trade-off (PR-AUC):", best_by_pr, "\n")
cat("3. Best balanced performance (F1):", best_by_f1, "\n")
cat(
  "\nConsider the specific requirements of your storm prediction task when cho
)
cat(
  "ROC-AUC is good for overall discrimination, PR-AUC is better for imbalanced
)


# Generate ROC curves for all models
generate_roc_data <- function(predictions, model_name) {
  predictions |>
    roc_curve(truth = storm, .pred_yes, event_level = "second") |>
    mutate(model = model_name)
}

# Create ROC data for all models
all_roc_data <- map2_dfr(
  all_predictions,
  names(all_predictions),
  ~ generate_roc_data(.x, .y)
)

# Calculate AUC values for legend
auc_values <- map_dfr(all_predictions, function(preds) {
  model_name <- unique(preds$model)
  auc_val <- roc_auc_vec(
    truth = preds$storm,
    estimate = preds$.pred_yes,
    event_level = "second"
  )
  data.frame(
    model = model_name,
    auc = round(auc_val, 3),
    stringsAsFactors = FALSE
  )
})

# Create model labels with AUC values
```

```r
auc_values <- auc_values |>
  mutate(model_label = paste0(model, " (AUC: ", auc, ")")) |>
  arrange(desc(auc))

# Add model labels to ROC data
all_roc_data <- all_roc_data |>
  left_join(auc_values, by = "model") |>
  mutate(model_label = factor(model_label, levels = auc_values$model_label))

# Create the ROC curves plot
roc_plot <- all_roc_data |>
  ggplot(aes(x = 1 - specificity, y = sensitivity, color = model_label)) +
  geom_line(size = 1.2, alpha = 0.8) +
  geom_abline(
    intercept = 0,
    slope = 1,
    linetype = "dashed",
    color = "gray50",
    alpha = 0.7,
    size = 0.8
  ) +
  scale_color_viridis_d(name = "Model (AUC)", option = "plasma") +
  labs(
    title = "ROC Curves Comparison - All Balanced Models",
    subtitle = "Models ordered by AUC performance",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)",
    caption = "Diagonal line represents random classifier performance"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12),
    legend.position = "right",
    legend.title = element_text(size = 11, face = "bold"),
    legend.text = element_text(size = 9),
    panel.grid.minor = element_blank(),
    axis.title = element_text(size = 11),
    axis.text = element_text(size = 10)
  ) +
  guides(
    color = guide_legend(
      override.aes = list(size = 2),
      ncol = 1
    )
  ) +
  coord_equal()
```

```r
print(roc_plot)

# Alternative version with better color palette
roc_plot_alt <- all_roc_data |>
  ggplot(aes(x = 1 - specificity, y = sensitivity, color = model_label)) +
  geom_line(size = 1.2, alpha = 0.8) +
  geom_abline(
    intercept = 0,
    slope = 1,
    linetype = "dashed",
    color = "gray50",
    alpha = 0.7,
    size = 0.8
  ) +
  scale_color_brewer(
    name = "Model (AUC)",
    type = "qual",
    palette = "Set1"
  ) +
  labs(
    title = "ROC Curves Comparison - All Balanced Models",
    subtitle = "Models ordered by AUC performance",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)",
    caption = "Diagonal line represents random classifier performance"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12),
    legend.position = "right",
    legend.title = element_text(size = 11, face = "bold"),
    legend.text = element_text(size = 9),
    panel.grid.minor = element_blank(),
    axis.title = element_text(size = 11),
    axis.text = element_text(size = 10)
  ) +
  guides(
    color = guide_legend(
      override.aes = list(size = 2),
      ncol = 1
    )
  ) +
  coord_equal()

print(roc_plot_alt)

# Print AUC summary table
```

```r
cat("\n=== AUC VALUES SUMMARY ===\n")
print(auc_values |> select(model, auc) |> arrange(desc(auc)))



#####################################################################
#####################################################################
# CODE FOR PROBLEM 2
#####################################################################
#####################################################################


load(here::here("Midterm", "telematics.RData"))

tm <- dat_1
rm(dat_1)

colMeans(is.na(tm))


tm <- tm |>
  mutate(
    across(where(is.character), factor),
    log_AMT_Claim = log(AMT_Claim)
  )


percentages <- tm |>
  select(
    log_AMT_Claim,
    AMT_Claim,
    tidyselect::matches("pct", "Pct"),
    tidyselect::starts_with("Pct")
  ) |>
  relocate(AMT_Claim, .after = log_AMT_Claim)

factors <- tm |>
  select(log_AMT_Claim, AMT_Claim, where(is.factor)) |>
  relocate(AMT_Claim, .after = log_AMT_Claim)

cnts <- tm |>
  select(log_AMT_Claim, AMT_Claim, where(is.numeric)) |>
  select(-tidyselect::matches("pct", "Pct")) |>
  select(-tidyselect::starts_with("Pct")) |>
  relocate(AMT_Claim, .after = log_AMT_Claim)

GGally::ggpairs(cnts)
GGally::ggpairs(percentages)
```

```r
corrplot::corrplot(cor(cnts))
corrplot::corrplot(cor(percentages))

predictor_names <- names(factors)[sapply(factors, is.factor)]

# This function takes the name of a predictor column as input.
create_boxplot <- function(predictor_col) {
  # Using the `!!sym()` pattern from rlang to handle the string column name
  # in the aes() mapping. This is the modern tidyverse approach.
  ggplot(
    factors,
    aes(
      x = !!sym(predictor_col),
      y = log_AMT_Claim,
      fill = !!sym(predictor_col)
    )
  ) +
    geom_boxplot(show.legend = FALSE) + # Hide legend as colors are redundant
    labs(
      title = predictor_col, # Use the predictor name as the title
      x = "", # Remove x-axis label for a cleaner look
      y = "Log of Amount Claimed"
    ) +
    theme_minimal(base_size = 14) +
    theme(
      plot.title = element_text(hjust = 0.5, face = "bold"), # Center and bol
      panel.grid.major.x = element_blank(), # Remove vertical grid lines
      panel.grid.minor = element_blank()
    ) +
    coord_flip()
}

list_of_plots <- map(predictor_names, create_boxplot)
final_plot <- wrap_plots(list_of_plots, ncol = 2)


lm(log_AMT_Claim ~ Car.use, data = tm) |> summary()
lm(log_AMT_Claim ~ Marital, data = tm) |> summary()
lm(log_AMT_Claim ~ Insured.sex, data = tm) |> summary()
lm(log_AMT_Claim ~ Region, data = tm) |> summary()


lm(log_AMT_Claim ~ Car.use, data = tm) |> anova()
lm(log_AMT_Claim ~ Marital, data = tm) |> anova()
lm(log_AMT_Claim ~ Insured.sex, data = tm) |> anova()
lm(log_AMT_Claim ~ Region, data = tm) |> anova()
```

```r
tm <- tm |> select(-Insured.sex)


# Lasso for Variable Selection ----------------------------------------

set.seed(1)
tm_split <- initial_split(tm, prop = 0.8, strata = log_AMT_Claim)
tm_train <- training(tm_split)
tm_train_include <- tm_train
tm_train <- tm_train |> select(-AMT_Claim)
tm_test <- testing(tm_split)
tm_test_include <- tm_test
tm_test <- tm_test |> select(-AMT_Claim)

# RECIPE
tm_recipe <- recipe(log_AMT_Claim ~ ., data = tm_train) |>
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |> # dummy-code factor
  step_corr(all_numeric_predictors(), threshold = 0.8) |> # remove highly cor
  step_normalize(all_numeric_predictors())


# LASSO
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet")

lasso_wflow <- workflow() |>
  add_model(lasso_spec) |>
  add_recipe(tm_recipe)

lasso_grid <- tibble(penalty = 10seq(-4, 0, length.out = 30))


lasso_res <- tune_grid(
  lasso_wflow,
  resamples = vfold_cv(tm_train, v = 20),
  grid = lasso_grid,
  metrics = metric_set(rmse)
)

show_best(lasso_res, metric = "rmse")

best_lasso <- select_best(lasso_res, metric = "rmse")

final_lasso <- finalize_workflow(lasso_wflow, best_lasso)

lasso_fit <- fit(final_lasso, data = tm_train)

lasso_coefs <- tidy(extract_fit_parsnip(lasso_fit)) |>
```

```r
  select(-penalty)

# conflicted::conflicts_prefer(brulee::coef)
#
# lasso_coef <- as.matrix(coef(extract_fit_engine(lasso_fit))) |>
#   as.data.frame() |>
#   rownames_to_column(var = "term") |>
#   rename(coefficient = `s1`) # `s1` is the default lambda selected

# Variable importance
# vip::vip(extract_fit_parsnip(lasso_fit))

# Select Variables ------------------------------------------------------

vars <- tidy(extract_fit_parsnip(lasso_fit)) |>
  select(term) |>
  unname() |>
  unlist() |>
  as.vector()
vars <- vars[-c(1, 28:35)]

tm_small <- tm |>
  select(c("log_AMT_Claim", "AMT_Claim", vars, "Marital", "Region", "Car.use")


set.seed(1)
tm_small_split <- initial_split(tm_small, prop = 0.8, strata = log_AMT_Claim)
tm_small_train <- training(tm_small_split)
tm_small_train_include <- tm_small_train
tm_small_train <- tm_small_train |> select(-AMT_Claim)
tm_small_test <- testing(tm_small_split)
tm_small_test_include <- tm_small_test
tm_small_test <- tm_small_test |> select(-AMT_Claim)

# pairs(tm_small_train)

# corrplot::corrplot(cor(tm_small_train))

# Linear or Not? --------------------------------------------------------

cnts <- tm_small_train |>
  select(log_AMT_Claim, where(is.numeric)) |>
  select(-tidyselect::matches("pct", "Pct")) |>
  select(-tidyselect::starts_with("Pct"))


# Vector of predictor names
predictors <- names(cnts)[names(cnts) != "log_AMT_Claim"]
```

```r
# Generate residual plots
resid_plots <- map(predictors, function(predictor) {
  # Create formula for regression
  fml <- as.formula(paste("log_AMT_Claim ~", predictor))

  # Fit model
  fit <- lm(fml, data = cnts)

  # Get standardized residuals
  std_resid <- rstandard(fit)

  # Create plot
  ggplot(cnts, aes_string(x = predictor)) +
    geom_point(aes(y = std_resid), alpha = 0.6) +
    geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
    labs(
      title = paste("Standardized Residuals vs", predictor),
      y = "Standardized Residuals",
      x = predictor
    ) +
    theme_minimal() + # sets default text size small
    theme(
      plot.title = element_text(size = 7),
      axis.text = element_text(size = 6),
      axis.title = element_text(size = 6)
    )
})

# Display the plots using patchwork
wrap_plots(resid_plots)


pct <- tm_small_train |>
  select(
    log_AMT_Claim,
    tidyselect::matches("pct", "Pct"),
    tidyselect::starts_with("Pct")
  )

# Vector of predictor names
predictors <- names(pct)[names(pct) != "log_AMT_Claim"]

# Generate residual plots
resid_plots <- map(predictors, function(predictor) {
  # Create formula for regression
  fml <- as.formula(paste("log_AMT_Claim ~", predictor))
```

```
  # Fit model
  fit <- lm(fml, data = pct)

  # Get standardized residuals
  std_resid <- rstandard(fit)

  # Create plot
  ggplot(pct, aes_string(x = predictor)) +
    geom_point(aes(y = std_resid), alpha = 0.6) +
    geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
    labs(
      title = paste("Standardized Residuals vs", predictor),
      y = "Standardized Residuals",
      x = predictor
    ) +
    theme_minimal() + # sets default text size small
    theme(
      plot.title = element_text(size = 7),
      axis.text = element_text(size = 6),
      axis.title = element_text(size = 6)
    )
})

wrap_plots(resid_plots)

# MODEL FITTING TIME ------------------------------------------------------

library("tidymodels")
library("kableExtra")
library("discrim")
library("bonsai")
library('xgboost')
library('kernlab')
library("stacks")
library("rstanarm")
library("broom.mixed")


# Recipe ------------------------------------------------------------------

tm_recipe <- recipe(log_AMT_Claim ~ ., data = tm_small_train) |>
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |> # dummy-code factor
  step_corr(all_numeric_predictors(), threshold = 0.8) |> # remove highly cor
  step_normalize(all_numeric_predictors())

set.seed(1)
tm_split <- initial_split(tm_small, prop = 0.8)
tm_train <- training(tm_split)
```

```r
tm_test <- testing(tm_split)

tm_recipe <- recipe(log_AMT_Claim ~ ., data = tm_train) |>
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |>
  step_corr(all_numeric_predictors(), threshold = 0.8) |>
  step_normalize(all_numeric_predictors())


# Model Specs ------------------------------------------------------------

# Linear (no regularization)
lm_spec <- linear_reg() |>
  set_engine("lm")

# Bayesian regression
bayes_spec <- linear_reg() |>
  set_engine(
    "stan",
    prior = rstanarm::normal(0, 10),
    prior_intercept = rstanarm::normal(0, 10)
  )

# LASSO
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet")

# Ridge
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) |>
  set_engine("glmnet")

# Elastic Net
enet_spec <- linear_reg(penalty = tune(), mixture = tune()) |>
  set_engine("glmnet")

# Random Forest
rf_spec <- rand_forest(mtry = tune(), min_n = tune(), trees = 1000) |>
  set_mode("regression") |>
  set_engine("ranger")

# XGBoost
xgb_spec <- boost_tree(
  trees = 1000,
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune()
) |>
  set_mode("regression") |>
  set_engine("xgboost")
```

```r
# Neural Net
nn_spec <- mlp(hidden_units = tune(), penalty = tune(), epochs = 100) |>
  set_mode("regression") |>
  set_engine("nnet")


# Workflows ------------------------------------------------------------------

make_wf <- function(model_spec) {
  workflow() |> add_model(model_spec) |> add_recipe(tm_recipe)
}

models <- list(
  "Linear" = lm_spec,
  "Bayesian" = bayes_spec,
  "LASSO" = lasso_spec,
  "Ridge" = ridge_spec,
  "ElasticNet" = enet_spec,
  "RandomForest" = rf_spec,
  "XGBoost" = xgb_spec,
  "NeuralNet" = nn_spec
)

tunable_models <- c(
  "LASSO",
  "Ridge",
  "ElasticNet",
  "RandomForest",
  "XGBoost",
  "NeuralNet"
)

model_wfs <- map(models, make_wf)


# Tuning Grids ---------------------------------------------------------------

# Define your parameter grids with finalized values
model_grids <- list(
  LASSO = grid_space_filling(
    parameters(penalty(range = c(-4, 0))),
    size = 30
  ),
  Ridge = grid_space_filling(
    parameters(penalty(range = c(-4, 0))),
    size = 30
  ),
```

```
  ElasticNet = grid_space_filling(
    parameters(
      penalty(range = c(-4, 0)),
      mixture(range = c(0.1, 0.9))
    ),
    size = 30
  ),
  RandomForest = {
    # finalize mtry after recipe is prepped
    prep_recipe <- prep(tm_recipe)
    n_features <- ncol(bake(prep_recipe, new_data = NULL)) - 1 # exclude respo
    rf_params <- parameters(
      finalize(mtry(), bake(prep_recipe, new_data = NULL)),
      min_n()
    )
    grid_space_filling(rf_params, size = 10)
  },
  XGBoost = grid_space_filling(
    parameters(
      tree_depth(range = c(3L, 10L)),
      learn_rate(range = c(0.01, 0.3)),
      loss_reduction(range = c(0.0, 5.0))
    ),
    size = 10
  ),
  NeuralNet = grid_space_filling(
    parameters(
      hidden_units(range = c(1L, 10L)),
      penalty(range = c(-4, -1))
    ),
    size = 10
  )
)


# Tuning the Models ------------------------------------------------------

cv_folds <- vfold_cv(tm_train, v = 10)

# Models
model_specs <- list(
  "LASSO" = linear_reg(penalty = tune(), mixture = 1) |> set_engine("glmnet"),
  "Ridge" = linear_reg(penalty = tune(), mixture = 0) |> set_engine("glmnet"),
  "ElasticNet" = linear_reg(penalty = tune(), mixture = tune()) |>
    set_engine("glmnet"),
  "RandomForest" = rand_forest(mtry = tune(), min_n = tune(), trees = 1000) |>
    set_mode("regression") |>
    set_engine("ranger"),
```

```
  "XGBoost" = boost_tree(
    trees = 1000,
    tree_depth = tune(),
    learn_rate = tune(),
    loss_reduction = tune()
  ) |>
    set_mode("regression") |>
    set_engine("xgboost"),
  "NeuralNet" = mlp(hidden_units = tune(), penalty = tune(), epochs = 100) |>
    set_mode("regression") |>
    set_engine("nnet")
)

# Workflows
model_wfs <- map(
  model_specs,
  ~ workflow() |> add_model(.x) |> add_recipe(tm_recipe)
)


# 10-fold CV
set.seed(1)
cv_folds <- vfold_cv(tm_train, v = 10)


# Tuning function
do_tune_safely <- possibly(
  function(wf, grid, model_name) {
    message("Tuning ", model_name, "...")
    tune_grid(
      wf,
      resamples = cv_folds,
      grid = grid,
      metrics = metric_set(rmse),
      control = control_grid(save_pred = TRUE)
    )
  },
  otherwise = NA,
  quiet = FALSE
)


# Run tuning (only tunable models)
# Build a list of inputs for pmap()
tune_inputs <- list(
  wf = model_wfs[tunable_models],
  grid = model_grids[tunable_models],
  model_name = tunable_models
```

```
)

# Now use pmap() correctly
tuned_results <- pmap(
  tune_inputs,
  do_tune_safely
)

write_rds(tuned_results, here::here("Midterm", "telematics_tuned.rds"))

best_results <- map_dfr(
  names(tuned_results),
  function(model_name) {
    res <- tuned_results[[model_name]]
    if (inherits(res, "tune_results")) {
      best_rmse <- show_best(res, metric = "rmse", n = 1)
      best_rmse$Model <- model_name
      return(best_rmse)
    } else {
      return(tibble(Model = model_name, .metric = "rmse", .estimate = NA))
    }
  }
)

# Clean & Format
best_results |>
  select(Model, .metric, mean, std_err) |>
  arrange(mean) |>
  mutate(across(where(is.numeric), round, 3)) |>
  kable(format = "latex", booktabs = TRUE, caption = "Best RMSE by Model") |>
  kable_styling(latex_options = c("striped", "hold_position"))


# Testing the models ----------------------------------------------------

# --- 1. Fit Regular Multiple Linear Regression Model ------------------
#
# lm_wf <- workflow() |>
#   add_model(lm_spec) |>
#   add_recipe(tm_recipe)
#
# lm_fit <- fit(lm_wf, data = tm_train)
#
#
# # --- 2. Fit Bayesian Linear Regression Model -------------------------
#
# bayes_wf <- workflow() |>
#   add_model(bayes_spec) |>
```

```r
#   add_recipe(tm_recipe)
#
# bayes_fit <- fit(bayes_wf, data = tm_train)

# save(lm_fit, bayes_fit, file = here::here("Midterm", "lr_mods.RData"))
load(here::here("Midterm", "lr_mods.RData"))

tidy(lm_fit, conf.int = TRUE)
tidy(bayes_fit, conf.int = TRUE)
tidy(extract_fit_parsnip(tuned_results$LASSO))

# NO TUNE MODELS
predict(lm_fit, new_data = tm_test)
predict(bayes_fit, new_data = tm_test)
# posterior_predict(extract_fit_engine(bayes_fit), newdata = tm_test)

# Fitting Final Models ----------------------------------------------------

# Extract best params for each tunable model
best_params <- map(tuned_results, ~ select_best(.x, metric = "rmse"))

# Finalize workflows with best params
final_wfs <- map2(
  model_wfs[tunable_models], # workflows
  best_params, # best hyperparameters
  ~ finalize_workflow(.x, .y)
)

# Fit each final workflow on training data
final_fits <- map(final_wfs, ~ fit(.x, data = tm_train))
names(final_fits) <- tunable_models

# Combine with non-tunable fits
all_fits <- list(
  "Linear" = lm_fit,
  "Bayesian" = bayes_fit
) |>
  append(final_fits)

# Optional: Save them
readr::write_rds(
  all_fits,
  here::here("Midterm", "telematics_fitted_models.rds")
)


# Coefficient Table -------------------------------------------------------
```

```r
# These are the models with coefficients
regression_models <- c("Linear", "Bayesian", "LASSO", "Ridge", "ElasticNet")

# Extract tidy coefficients
coef_tbls <- map(
  regression_models,
  function(model_name) {
    model_fit <- all_fits[[model_name]]
    coef_tbl <- tidy(extract_fit_parsnip(model_fit), conf.int = TRUE)
    coef_tbl$model <- model_name
    return(coef_tbl)
  }
)

# Combine into one tibble
all_coefs <- bind_rows(coef_tbls)

# Round and reshape to wide format
coef_table <- all_coefs |>
  select(model, term, estimate) |>
  mutate(estimate = round(estimate, 3)) |>
  pivot_wider(names_from = model, values_from = estimate)


coef_table |>
  arrange(term) |>
  kable(
    format = "latex",
    booktabs = TRUE,
    caption = "Coefficient Estimates by Regression Model",
    align = "lrrrrr"
  ) |>
  kable_styling(latex_options = c("striped", "hold_position"), font_size = 8)


# Predict and exponentiate for all models
# Create predictions with actuals per model
predictions_combined <- map_dfr(
  names(all_fits),
  function(model_name) {
    model_fit <- all_fits[[model_name]]

    pred_tbl <- predict(model_fit, new_data = tm_test) |>
      mutate(
        .pred_exp = exp(.pred),
        log_AMT_Claim = tm_test$log_AMT_Claim,
        AMT_Claim = exp(log_AMT_Claim),
        Model = model_name
```

```
      )

    return(pred_tbl)
  }
)


library("yardstick")

# Summary: RMSE and MAE on original scale
predictions_combined |>
  group_by(Model) |>
  summarise(
    RMSE = rmse_vec(truth = AMT_Claim, estimate = .pred_exp),
    MAE = mae_vec(truth = AMT_Claim, estimate = .pred_exp)
  ) |>
  arrange(RMSE) |>
  kable(
    # format = "latex",
    booktabs = TRUE,
    caption = "Performance on Original AMT_Claim Scale",
    digits = 3
  ) |>
  kable_styling(latex_options = c("striped", "hold_position"))


predictions_combined |>
  group_by(Model) |>
  summarise(
    MSPE = mean((AMT_Claim - .pred_exp)2),
    RMSE = rmse_vec(truth = AMT_Claim, estimate = .pred_exp),
    MAE = mae_vec(truth = AMT_Claim, estimate = .pred_exp)
  ) |>
  arrange(MSPE) |>
  kable(
    # format = "latex",
    booktabs = TRUE,
    caption = "Performance on Original AMT\\_Claim Scale",
    digits = 3
  ) |>
  kable_styling(latex_options = c("striped", "hold_position"))
```