MACHINE LEARNING PROJECT

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

To Grant or Not to Grant

Group 43

Beatriz Fonseca, 20201599
João Marques, 20240656
Martim Tavares, 20240508
Matilde Domingues, 20240523
Rodrigo Miranda, 20240490

Fall/Spring Semester 2024-2025

Project Available on GitHub: https://github.com/jm651120/ml_project

Index

Table of Tables

Table of Figures

Abstract

This study explores the use of machine learning models for predicting the Workers' Compensation Board's (WCB) final decision regarding the type of injury ('Claim Injury Type') assigned to claims. Accurate classification of injury types is a critical component in streamlining claims processing and improving decision-making consistency within the WCB. Using a multiclass classification framework, we developed a robust preprocessing pipeline tailored to the dataset's unique challenges, including missing data and the highly imbalanced distribution of the target variable. Various machine learning models, including XGBoost, a Neural Network, and Random Forest, were benchmarked to evaluate their performance and generalizability.

The dataset exhibited a pronounced imbalance in Claim Injury Type, with the majority class ('NON-COMP') accounting for more than 50% of all claims, while minority classes like 'PTD' and 'DEATH' representing less than 1%. This imbalance posed a significant challenge, as models tended to predict the majority class at the expense of underrepresented classes. Despite preprocessing efforts and feature selection, the models struggled to achieve robust performance, as reflected in the limited F1-Macro Score.

XGBoost emerged as the best-performing model, demonstrating reasonable performance but failing to capture meaningful patterns for minority classes. Other models, like Neural Networks, while explored as an alternative, also underperformed compared to XGBoost. These findings underscore the difficulties associated with imbalanced multiclass classification tasks and highlight the limitations of the models in accurately predicting underrepresented classes. This study emphasizes the importance of addressing these challenges to enhance the effectiveness of predictive models in real-world applications.

Introduction

Workplace injuries pose significant challenges for organizations like the New York Workers' Compensation Board (WCB), which processes millions of claims annually. Machine learning has emerged as a critical tool to address these challenges, particularly for predicting claim severity, a key factor in determining insurance premium rates.

Recent studies have compared Neural Network-based models, such as Multi-Layer Perceptrons (MLP) and TabTransformers, with Tree-based models like Decision Trees, Random Forests, and XGBoost, finding that Tree-based models often outperform Neural Networks for tabular data applications [1]. Inspired by these findings, our project aimed to predict the 'Claim Injury Type', a multiclass classification problem crucial to the WCB's operations, using advanced machine learning methods.

Our preprocessing pipeline addressed challenges such as missing data and feature inconsistencies, encoding, imputation, and outlier management techniques. Among the models tested — Logistic Regression, Decision Trees, Random Forests, Neural Networks, XGBoost, Naive Bayes and a stacking model with Naive Bayes and Logistic Regression — the XGBoost Classifier demonstrated the best performance, consistent with findings from [1]. Additionally, we explored the predictive value of the 'Agreement Reached' variable, despite its highly imbalanced distribution, by incorporating it into the primary model.

This report outlines our methodologies, highlights the challenges faced, and presents insights into improving predictive modelling for workers' compensation claims.

Data Exploration and Preprocessing

Data Exploration

Before any predictive model can be developed, one must first explore and understand the data. The subsequent workflow was implemented:

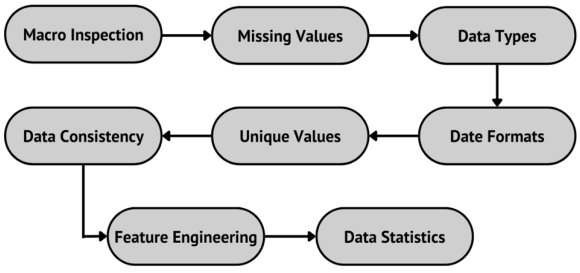


Figure 1 - Data Exploration Workflow

1. Macro Inspection: The dataset structure and the datatypes of each column were examined. During this analysis, several inconsistencies with the metadata were identified and addressed later in step 3 (See Annex 1). After, the feature 'Claim Identifier' was inspected with the aim of assigning it as the

dataframe index - indeed it had no missing values and only one duplicate value, which was dropped. This column was then successfully assigned as the new index of the dataframe.

2. Missing Values: A comprehensive analysis of missing values was performed (See Annex 2). For 'C-3 Date' and 'First Hearing Date', the missing values will be addressed using an imputation algorithm. The missing values in 'IME-4 Count' will be imputed with the value '0', while 'OIICS Nature of Injury Description' will be excluded from the dataset entirely.

3. Data Types: Upon loading the data file, a warning about mixed data types appeared. To address this, an inspection was conducted to identify the problematic columns and determine appropriate fixes. The issue was traced to the 'Zip Code' column, which contained non-numeric values - this column was converted to the string data type. Furthermore, the columns identified in step 1 were transformed to integers as part of the resolution process.

4. Date Formats: With the presence of date features in the dataset, it was essential to verify their formats and ensure consistency. After confirming that all date features followed the same format, 'Accident Date', 'Assembly Date', 'C-2 Date', 'C-3 Date' and 'First Hearing Date' were converted to the datetime data type.

5. Unique Values: By printing all unique values in the dataset, inconsistencies and class values of categorical features were identified (See Annex 3).

6. Data Consistency: To address the inconsistencies identified before, several transformations were applied to the dataset (See Annex 4).

7. Feature Engineering: New features were created with the aim of improving model performance.

Feature	Formula	Data Type	Explanation
'C-2 Missed Timing'	'C-2 Date' > 'Accident Day' + 10 days	Boolean	This form needs to be submitted within 10 days of 'Accident Date'
'C-3 Missed Timing'	'C-3 Date' > 'Accident Day' + 2 years	Boolean	This form needs to be submitted within 2 years of 'Accident Date'
'Days Difference'	'Assembly Date' - 'Accident Date'	Integer	Days between the accident and the claim assembly
'C-2 Missing'	'C-2 Date' == np.nan	Boolean	C-2 Form is missing or not
'C-3 Missing'	'C-3 Date' == np.nan	Boolean	C-3 Form is missing or not
'Has Hearing'	'First Hearing Date' ≠ np.nan	Boolean	Hearing was held if there is a non-null value for 'First Hearing Date'
'Has IME-4 Report'	'IME-4 Count' ≠ np.nan	Boolean	Has at least one IME-4 report if there is a value for this feature
Log <Feature>	log(<Feature>)	Float	Log transformation of numeric features: 'Age at Injury', 'Average Weekly Wage', 'Birth Year', 'IME-4 Count' and 'Number of Dependents'

Table 1 - Feature Engineering

8. Data Statistics: The number of missing values was significantly reduced. Summary statistics were generated for numeric, date and categorical features (See Annex 5).

Data Visualization

Data visualization is the next step to gain a better and deeper understanding of the data. To do so, several histograms and boxplots were plotted, which gave us a better understanding of the data distribution.

Next, it is of utmost importance understanding how each feature relates to the target and how helpful they could be in developing the final model. Therefore, heatmaps depicting the correlation between features is a great visual aid (See Annex 6, 7 and 8).

As for the **numeric features**, by computing the spearman correlation, we learn that 'Birth Year' and 'Age at Injury' are inversely correlated and 'Average Weekly Wage' is highly correlated with the target. Regarding the **date features**, by computing the spearman correlation, we note that 'First Hearing Date' and 'C-3 Date' have a moderate correlation with the target. Moreover, 'Accident Date', 'Accident Date' and 'C-2 Date' are highly correlated among themselves and with 'C-3 Date Year' and 'First Hearing Date Year'. To understand how relevant the **categorical features** were to the target, we computed the chi-square test and plotted the Cramér's V, which showed which features had a bigger association with the target: 'Attorney/Representative', 'Agreement Reached', 'C-2 Missing', 'C-3 Missing', 'Has Hearing' and 'Has IME-4 Report'.

Right before starting the preprocessing efforts, multivariate analysis was performed (Check Appendix 1 for some insights).

Data Preprocessing

Before starting building the preprocessing pipeline, there are still some inconsistencies left resolving:

- Setting to null any values of 'Age at Injury' below 14.
- Transforming the negative value of 'WCIO Part Of Body Code' to its absolute value.

After, simple encodings to categorical variable were applied (See Annex 9). The data is now ready to be used inside a pipeline, which will contain all the preprocessing steps, feature selection and model definition. Here is a visual implementation of the preprocessing pipeline:

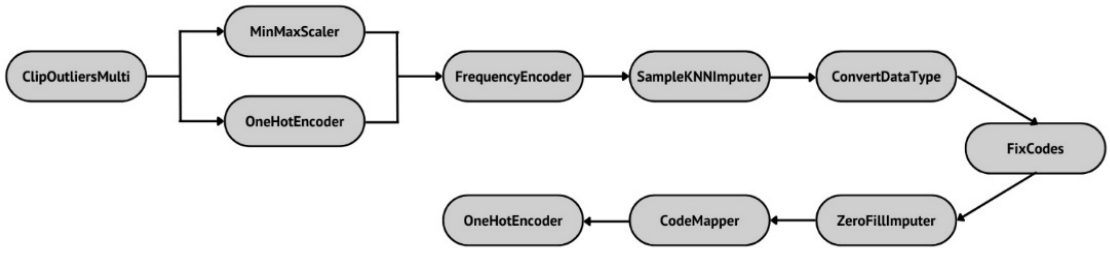


Figure 2 - Preprocessing Steps

Next, are the custom classes created to help build the pipeline:

Class	Purpose	Features
ClipOutliersMulti	Sets the values over the defined quantile/value threshold to the maximum allowed by threshold	'Age at Injury' , 'Average Weekly Wage , 'Birth Year' , 'Days Difference' , 'IME-4 Count'
FrequencyEncoder	Replaces the string-valued categories with the category frequency in percentage	'Carrier Name', 'County of Injury', 'Zip Code'
SampleKNNImputation	Runs the KNNImputer() on a sample of the data	'Alternative Dispute Resolution', 'Average Weekly Wage', 'Birth Year', 'Carrier Name', 'Gender', 'Age at Injury','Zip Code', 'Accident Date_year', 'Accident Date_month', 'Accident Date_day', 'Days Difference','Industry Code', 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code', 'C-2 Missed Timing', 'C-3 Missed Timing', 'Log Average Weekly Wage', 'Log Birth Year', 'Log IME-4 Count'
ZeroFillImputer	Imputes missing values with '0'	'IME-4 Count', 'C-2 Date_year', 'C-2 Date_month', 'C-2 Date_day','C-3 Date_year', 'C-3 Date_month', 'C-3 Date_day','First Hearing Date_year', 'First Hearing Date_month', 'First Hearing Date_day'
ImputerGridSearch	Performs hyperparameter tuning on the imputer	
CodeMapper	Maps the original Code categories to new ones	'Industry Code', 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code'
ConvertDataType	Converts a feature to a specified data type	'Industry Code', 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code'
FixCodes	Maps unmaped values to the closest valid one	'Industry Code', 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code'

Table 2 - Preprocessing Custom Functions

Multiclass Classification

After completing all preprocessing steps, we proceed to model assessment using a StratifiedKFold approach with five splits to ensure balanced representation of all classes and to maximize the usage of the dataset by utilizing all data for both training and validation across different folds. The model scores will be consolidated and presented at the end of this section for comparison.

XGBoost

XGBoost, developed by Tianqi Chen and Carlos Guestrin (XGBoost: A Scalable Tree Boosting System, 2016), is a machine learning algorithm designed to enhance the performance of gradient-boosted decision trees. It introduces innovations like L1 and L2 regularization to reduce overfitting, efficient parallelized computation for faster training, and the ability to handle missing data natively. It builds trees iteratively by focusing on minimizing residual errors, allowing it to refine predictions and improve model generalization.

The theoretical foundation of XGBoost lies in its objective function, which balances predictive accuracy with model complexity. The objective function includes a loss term, which measures the difference between predicted and actual values, and a regularization term, which penalizes overly complex models to prevent overfitting. In each iteration, the algorithm minimizes the Taylor expansion of the objective function up to the second order, allowing it to calculate precise updates for tree splits, ensuring that each tree contributes to reducing overall error while maintaining simplicity. By optimizing both the structure and the weights of the trees, XGBoost achieves a fine balance between bias and variance, making it one of the most efficient gradient-boosting methods available today.

We utilized a feature selection approach based on XGBoost's feature importance, which outperformed Recursive Feature Elimination (RFE) in terms of both speed and model performance. But this raises an

important question: how do we decide the optimal number of features to include after determining feature importance?

Since testing every number of features would be computationally expensive, we ran some ranges and understood the best number, i.e., the best combination between overfit and the f1-macro validation score, would be between 50 and 60. To minimize overfitting, we decided to maintain a maximum difference of 2 to 3 percentage points between the F1-macro scores for training and validation. Within this constraint, we aimed to maximize the validation score, which was achieved with 53 features.

We also explored the impact of applying a Spearman correlation-based filtering step (Annex 10 and 11) before using XGBoost's feature importance. In this approach, features with a correlation coefficient above 0.8 were removed to reduce redundancy. However, this lead to worse performance. We believe that variables like 'C-2 Date_year' were removed due to correlation, despite being identified by XGBoost as the 14th most important feature, although with relatively low importance (approximately 1%).

To further enhance generalization and reduce overfitting, we implemented Lasso and Ridge regression during the training process. This step was crucial and was the one that allowed us to eliminate a big part of the overfitting we were getting while maintaining the f1-macro score on the validation data.

Parameter	Description	Value	Justification
n_estimators	number of trees	200	Increasing the number of trees or the learning rate caused the model to overfit without improving the score on the test data. Decreasing it led to a poorer performance both on training and test data. In relation to learning_rate a smaller value improves generalization by taking smaller and more stable steps but requires more boosting rounds.
learning_rate	update the weights of the trees	0.1	
max_depth	max depth of the trees	6	Ensures reasonable model complexity, capturing interactions without becoming overly sensitive to noise or outliers.
reg_alpha	L1 regularization	5	This combination of Lasso (L1) and Ridge (L2) regression lead to significantly reduce overfitting while maintaining the test score. L1 removes irrelevant features and L2 stabilizes feature weights.
reg_lambda	L2 regularization	10	

Table 3 - XGBoost Parameters

This was the model where we got the best score. Looking at the confusion matrix (See Annex 12), the minority classes were badly predicted, and this explains why the score is not higher.

Neural Network

The MLPClassifier() was the second model we decided to test. We initiate the model with a simple set of parameters. To perform hyperparameter optimization, the GridSearchCV was used (See Annex 13). As a result, we got the parameters highlighted below as the best parameters for the model, with which the pipeline was run.

Parameter	Description	Value	Justification
hidden_layer_sizes	Number of neurons of each layer of the neural network	(250, 120)	Allows for two hidden layers with 250 and 120 neurons, balancing model complexity and computational efficiency
batch_size	Number of samples per batch	500	Helps optimize computational performance while providing sufficient data for stable updates
max_iter	Maximum number of iterations	400	Ensures the model has enough time to converge without excessive computational time
learning_rate	Strategy for adjusting the learning rate during training	constant	Keeps the learning rate steady
learning_rate_init	Initial learning rate for the optimization	0.01	Provides a moderate starting point for training, balancing convergence speed and stability
solver	Algorithm used for optimization	adam	It is recommended for large datasets, working well in terms of training time and validation score

Table 4 4 - MLP Parameters

In terms of class prediction, this model was able to predict reasonably well all labels except for classes '6. PPD NSL' and '7. PTD', for which it was unable to predict a single value (See Annex 14). We believe this was the main reason for the score to not be higher. In terms of overfitting, the model was able to generalize quite well, with scores 0.416 and 0.407 for the f1 macro in the train and validation sets, respectively (See Annex 15 for more metric scores).

Decision Trees

The **DecisionTreeClassifier()** was the next model we decided to test. We started by analysing each parameter's possible values to be included in the GridSearchCV to perform hyperparameter optimization, based on the highest f1-score for validation, and always considering over and underfitting. In Annex 16 are the parameters tested, and the values chosen for the GridSearchCV. The best parameters for this model are criterion=**'entropy'**, max_depth=**13** and min_samples_leaf=**10**. For the remaining parameters the chosen value was the default.

After, we tried adding **SMOTE** to the model to improve class imbalance since there are classes with few observations. We also attempted the **1vsALL** approach and **Bagging**, which is an ensemble learning technique. Here are the results for the 4 models tested so far in Decision Trees:

Mean F1 Macro	Decision Tree	with SMOTE	1vsAll	Bagging
train	0.4444	0.4179	0.4798	0.4315
val	0.4047	0.3873	0.4079	0.4020
train - val	0.0397	0.0306	0.0719	0.0295

Table 55 - Decision Tree Approaches Scores

The model 1vsALL has the highest validation score; however, it is also the one with the most overfitting. Therefore, we choose the **Bagging** model, which has a good trade-off between validation and train scores, and much less overfitting. Choosing the Decision Tree over Bagging does not compensate the increase in overfit (0.0102) over the increase in the validation score (0.0027). Now we do **feature selection** to see if it can still improve.

We first performed a GridSearchCV to obtain the best number of features for **SelectKBest** and then used the evaluate_feature_intervals function to get the optimal number of features using **XGBFeatureSelector**. After, we used the split criteria 'gini' and 'entropy' with DecisionTreeClassifier()

to obtain the best number of features for **DT_FeatureSelector**. The outputs are in the Annex 17, and the results are in the following table:

Feature Selection	nº features	F1 Macro train	F1 Macro val
without	all	0.4315	0.4020
SelectKBest	57	0.4319	0.4068
XGBFeatureSelector	39	0.4230	0.4058
DT_FeatureSelector	19	0.4226	0.3924

Table 66 - Decision Tree Feature Selection Results

The model with SelectKBest achieved the highest validation score, however the one with XGBFeatureSelector has less overfitting and a high validation score as well. So, our best model in Decision Trees was **Bagging with XGBFeatureSelector**. The confusion matrix and classification report are in Annex 18.

Random Forest

The next model was **Random Forest().** We started by running the model with default parameters to establish a baseline for comparison. After, **SMOTE** was added to address class imbalance, which resulted in better scores. Then, **RandomizedSearchCV** was used to analyze which parameters were the best by performing one based on the f1-score metric (See Annex 19).

After hyperparameter optimization, we performed **feature selection** with and without **SMOTE** as a complement, both using **SelectKBest** based on the f1 macro score. Applying SMOTE initially improved predictions on minority classes but resulted in significant overfitting, therefore we decided not to use it. All the metrics are represented in Annex 20.

F1 - Score	Training F1-Macro	Validation F1-Macro	train - val
without SMOTE	0.4029	0.3246	0.0783
with SMOTE	0.5509	0.3876	0.1633
Final Model	0.4029	0.3246	0.0783

Table 77 - Random Forest Approaches Scores

The **Random Forest** model achieved a good f1-macro score, though it was lower than XGBoost. The model struggled with minority classes, that's where smote helped improve prediction, but the impact on overfitting was high, therefore it wasn't included on the final model. The confusion matrix shows the model is good at predicting majority classes but not with minority classes (See Annex 21).

Future work could explore combining Random Forest with different approaches for class imbalance, such as class-weight training, boosting techniques and ensemble models.

Logistic Regression

Logistic Regression was implemented, using GridSearchCV to optimize the model hyperparameters. The parameters considered include Inverse of Regularization Strength (C), Penalty, Solver and the ratio

between l1 and l2 solvers, which is only applicable when using *elasticnet* as penalty (See results in Annex 22 and 23).

Having the model parameters defined, we moved on to feature selection. To establish a baseline, no feature selection technique was used and SelectKBest and XGBoost-based feature selection were tested, due to their established effectiveness in handling high-dimensional datasets and their computational efficiency.

Feature Selection	Number of Features	Mean Train F1-Macro	Mean Val F1-Macro	Val Std Deviation	Mean Val Recall-Macro	Mean Val Precision-Macro
No feature selection	All features	0.4655	0.4027	0.0289	0.3855	0.5092
SelectKBest	67	0.3736	0.374	0.054	0.3609	0.5005
XGBoost	68	0.3762	0.3743	0.0044	0.3615	0.5021

Table 8 - Logistic Regression Feature Selection

Both feature selection methods were able to reduce overfit tremendously. While using all features provides better scores, the overfit makes it less favorable for practical use. Among the feature selection approaches, the XGBoost-based marginally outperformed SelectKBest in terms of precision and recall, showing better score stability across the folds, making it the preferred choice.

In general, Logistic Regression revealed a good performance for labels '2. NON-COMP' and '4. TEMPORARY' and a poor performance for minority labels such as '6. PPD NSL' and '7. PTD' (See Annex 24). This outcome reflects the limitations of Logistic Regression when handling imbalanced datasets and complex feature interactions.

Naive Bayes and Stacking Model with Logistic Regression and Naive Bayes

We attempted to create a stacking model to see if we could improve the performance. Although Naive Bayes performed poorly as an independent model, it was primarily defined to serve as a base learner in a stacking model.

The stacking classifier combines two base learners - Logistic Regression and Naive Bayes - with a Logistic Regression model as the final estimator. Despite the combination of models in the stacking approach, it did not outperform Logistic Regression alone and showed slightly worse performance. This highlights that adding complexity through stacking does not always yield better results and may depend on the complementary nature of the base models. Note that we tried different stacking combinations, and this was the one that performed the better.

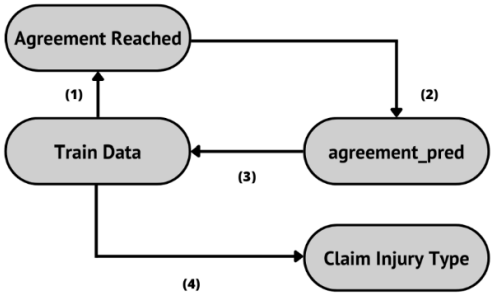
Model Evaluation

F1 Macro	XGBoost	Neural Networks	Decision Tree	Bagging	Logistic Regression	Random Forest	Stacking
train	0.4534	0.4156	0.4444	0.4230	0.3762	0.4029	0.3932
val	0.4317	0.4071	0.4047	0.4058	0.3743	0.3246	0.3903
train - val	0.0217	0.0085	0.0397	0.0172	0.0019	0.0783	0.0029

Table 99 - Model Comparison

Open-Ended Section

In our project, there are two variables that are not present in the test data: 'Agreement Reached' and 'WCB Decision'. Given that 'WCB Decision' does not exhibit variability, we focused on building a model to predict 'Agreement Reached', incorporating the predictions into the model predicting 'Claim Injury Type', and analyzing whether the results improved. Here is a diagram depicting the entire process:



(1) – Using the training data to train the model that predicts Agreement Reached.
(2) & (3) – Out-of-Fold predictions for Agreement Reached and incorporating the feature agreement_pred into the training data.
(4) – Using the same data, we train the model and make predictions for Claim Injury Type and evaluate the impact of the new feature agreement_pred on it.

Figure 3 - Open-Ended Section Workflow

The dataset shows a heavily imbalanced distribution for 'Agreement Reached', with class '1' representing only around 5% of the data. This imbalance poses significant challenges for the model, as it struggles to learn meaningful patterns for class '1' due to its rarity and is likely to default to predicting class '0' most of the time.

Additionally, the dominance of class '0' results in low variance for this variable, limiting its capacity to provide meaningful information for predicting 'Claim Injury Type.' Including such a variable may not improve the model's performance and could even introduce noise, increasing complexity without adding real value. Therefore, it is crucial to carefully assess whether this variable meaningfully enhances the model or if excluding it would promote a simpler and more robust approach.

The preprocessing pipeline used for the 'Claim Injury Type' prediction task was adapted for the new model. From the previously documented models, only XGBoost, Neural Network, and Logistic Regression were selected, maintain the chosen hyperparameters except for Logistic Regression, since a different set of parameters provided better scores.

Agreement Reached	XGBoost	Neural Networks	Logistic Regression
train	0.6227	0.5849	0.5711
val	0.6115	0.5641	0.5709

Table 1010 - Open-Ended Model Comparison

The results aligned with expectations, confirming the model's difficulty in predicting class '1'. The confusion matrix highlights this issue (Annex 25), showing a high number of false negatives for class '1' which underscores the model's bias towards the majority class '0'. This reinforces our understanding of the challenges posed by the dataset's distribution and the limitations of this variable for improving the 'Claim Injury Type' prediction.

After incorporating the variable agreement_pred in the model that predicts 'Claim Injury Type', the result was the following using XGBoost:

	XGBoost
train	0.4536
val	0.4332

Table 1111 - Score after adding predicted 'Agreement Reached'

The overall score remained almost unchanged, as expected. After checking feature importance based on XGBoost, besides agreement_pred being one of the most important features, its relative importance was still quite low (0.0479), contributing minimally to the model's predictive power. This limited impact can be attributed to its low variance and the challenges associated with predicting 'Agreement Reached' effectively.

Conclusion

This project aimed to predict the Workers' Compensation Board's (WCB) decision for Claim Injury Type addressing key challenges such as class imbalance. Using several machine learning models and techniques, we were able to improve predictions and automate claims processing.

Out of all the models we tested, XGBoost performed the best, achieving the highest F1-Macro Score and outperforming Neural Networks (NN) and Decision Trees (DT). Surprisingly, the model's result went against what the studies suggested [1], where Tree-based models often outperform Neural Networks for tabular data. This shows how results can depend on the datasets and setups.

As said before, XGBoost scored the highest F1-Macro score, however the model struggled with minority classes such as 'PDT' and 'Death', being this one of the critical areas for improvement. Some techniques like Lasso and Ridge were used to reduce overfitting, and XGBoost's importance for feature selection, which really helped on the model's performance.

From the open-ended section, we concluded that while adding features like 'Agreement Reached' might seem useful, its low variability limited its contribution to the model performance, which showed how important it is to assess feature relevance before adding them to the pipeline.

Moreover, this project lays out the path ahead, such as identifying limitations and improvements of the work and hinting at what could be done in the future. These could be exploring class-weighted training or other techniques for class imbalance, using more feature selection methods and many others.

In conclusion, this project provided valuable insights about predictive modeling and gave a strong Foundation for automating claims classification on this case. While the challenges from the dataset limited the model's performance, they also motivated us to explore and study several solutions which contributed to the development of practical skills for real-world scenarios.

References

S. Tjahjono, H. Murfi and S. Devila, "Claim Severity Prediction of Workers' Compensation Using Tree and Neural Networks-Based Models," 2024 11th IEEE Swiss Conference on Data Science (SDS), Zurich, Switzerland, 2024, pp. 225-228, doi: 10.1109/SDS60720.2024.00039. keywords: {Accuracy;Pandemics;Neural networks;Insurance;Predictive models;Transformers;Market research;claims severity prediction;machine learning;deep learning;tree-based models;neural networks-based models},

Injury Description Codes Part of Body. (n.d.). Retrieved December 17, 2024, from https://www.dir.ca.gov/dwc/WCIS/Part_of_Body.pdf

Injury Description Codes Nature of Injury. (n.d.). Retrieved December 17, 2024, from https://www.dir.ca.gov/dwc/WCIS/Nature_Of_Injury.pdf

Injury Description Codes Cause Of Injury. (n.d.). Retrieved December 17, 2024, from https://www.dir.ca.gov/dwc/WCIS/Cause_Of_Injury.pdf

C-2 Form. (n.d.). Retrieved December 22, 2024, from https://www.wcb.ny.gov/content/main/forms/c2F.pdf

File a Claim. (n.d.). Retrieved December 22, 2024, from https://www.wcb.ny.gov/content/main/Workers/file-claim.jsp

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). arXiv. https://arxiv.org/abs/1603.02754

XGBoost Documentation. (n.d.). Retrieved December 22, 2024, from https://xgboost.readthedocs.io/en/latest/index.html

XGBoost Team. (n.d.). Model training tutorial: A step-by-step guide to using XGBoost. Retrieved December 22, 2024, from https://xgboost.readthedocs.io/en/stable/tutorials/model.html

Annexes

Annex 1 – Macro Inspection Insights

- 'Age at Injury', 'Birth Year', 'IME-4 Count', 'Industry Code', 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code' and 'Number of Dependents' were defined as objects and were corrected to integers.

- 'Agreement Reached' was defined as object and was corrected to integer, to preserve the '0' and '1' values.

Annex 2 – Missing Values Insights

- 'C-3 Date': Over 50% of the data for this feature is missing, likely due to process status - the employee has not yet submitted the report.

- 'First Hearing Date': Approximately 75% of the data is missing, indicating that hearings have not been held.

- 'IME-4 Count': More than 75% of the data is missing, potentially because the independent examiner has not submitted the report.

- 'OIICS Nature of Injury Description': This feature only contains null values.

Annex 3 – Unique Values Insights

- 'Age at Injury': Since this dataset concerns workplace injuries, values under 14 (legal working age in the US) are weird.

- 'Birth Year': Contains '0.0' values.

- 'OIICS Nature of Injury Description': Contains no values.

- 'WCIO Part Of Body Code': Includes a negative value.

- 'WCB Decision': Contains only one unique value.

Annex 4 – Data Consistency Insights

- **'Birth Year':** The '0' values were fixed by subtracting 'Age at Injury' from the year component of 'Accident Date'. Any remaining inconsistent values were set to null for later imputation.

- **'Age at Injury':** The values below 14 were adjusted by subtracting 'Birth Year' from the year component of 'Accident Date'. Inconsistent values were then set to null for later imputation.

- **'Gender':** Four categories were identified; those that were neither 'M' or 'F' were set to null for later imputation.

- **'Accident Date':** This should be the smallest value among the date columns. Values not meeting this condition were swapped with the smallest value among 'Assembly Date', 'C-2 Date', 'C-3 Date' and 'First Hearing Date'.

- **'First Hearing Date':** This should be the largest value among the date columns. Values not meeting this condition were swapped with the largest value among 'Assembly Date', 'C-2 Date', 'C-3 Date' and 'Accident Date'.

- **'Assembly Date':** Due to previous transformations, inconsistencies arose in this feature. The year value was set to the maximum possible - '2022'. Following, the previous two conditions were rechecked and adjusted.

- **'Zip Code':** Some values were encoded as the string 'nan'. These were replaced with the pandas' default numpy.nan function.

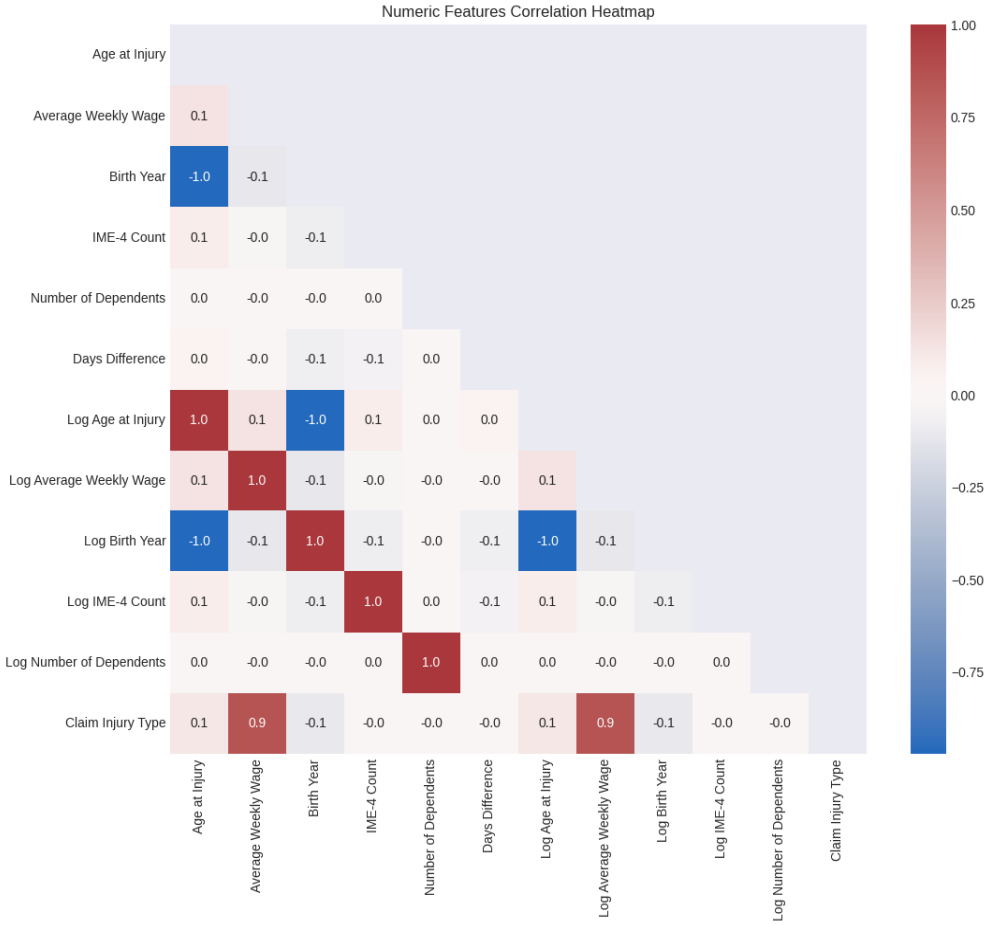
- **Duplicated Values:** The dataset contained duplicate rows where all columns were empty aside from 'Assembly Date', except for one row. All duplicate columns were dropped from the dataset.

- **'Claim Injury Target':** The target variable contained some missing values, which were promptly dropped.

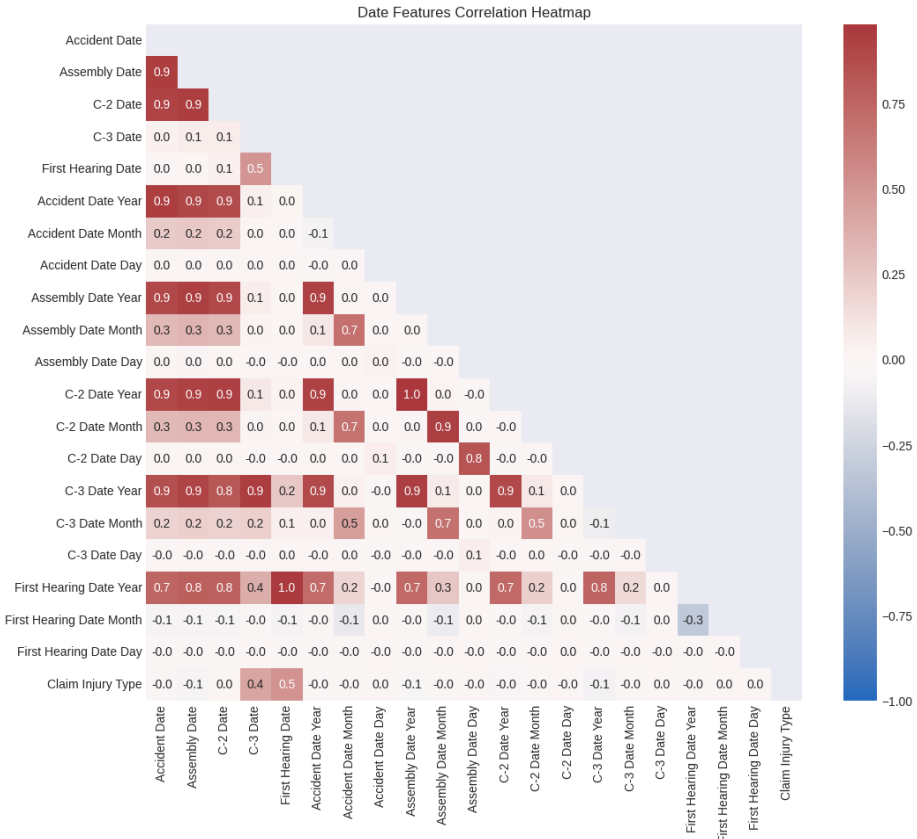
Annex 5 - Summary Statistics

Feature Group	Feature	Insight
Numeric	'Age at Injury'	The average age at injury is 42 years. Extremely low and high values (e.g., above typical retirement age of 63–65) may need correction
	'Average Weekly Wage'	Over half the dataset has a value of 0, while the highest recorded wage exceeds $2.8B, indicating potential anomalies
	'IME-4 Count'	The average is 3 reports per claim; the maximum of 73 reports appears unusual
	'Number of Dependents'	On average, individuals have 3 dependents
	'Days Difference'	It takes 3 months on average to assemble a claim, but extreme values exist, with the 75th percentile at just 40 days
	Log Transformations	Applying log transformations normalized most distributions, reducing variability, except for 'Number of Dependents', which originally resembled a uniform distribution
Date	'Accident Date'	Over 75% of accidents happened after 2020, eventhough the first occurred in 1960
	'First Hearing Date'	Over 50% of hearings were held in 2022
Categorical	'Attorney/Representative'	Most claims lack legal representation
	'Gender'	Males are more prone to workplace injuries
	'COVID-19 Indicator'	About 95% of claims are unrelated to COVID-19
	'Industry Code Description'	Health Care and Social Assistance has the highest number of claims
	'WCIO Cause of Injury Description'	Lifting is the most common cause
	'WCIO Nature of Injury Description'	Strain or Tear is the most frequent injury type
	'WCIO Part Of Body Description'	The Lower Back Area is the most commonly injured
	'Zip Code'	Claims are geographically dispersed with no dominant region
	'Agreement Reached'	Most claims require WCB involvement to close
	'C-2 Missed Timing'	The C-2 form timing is missed over half the time
	'C-3 Missed Timing'	The C-3 form timing is missed in more than 75% of cases
	'C-2 Missing'	Over 75% of claims lack a C-2 form entirely
	'Has IME-4 Report'	More than 75% of claims lack an examiner report

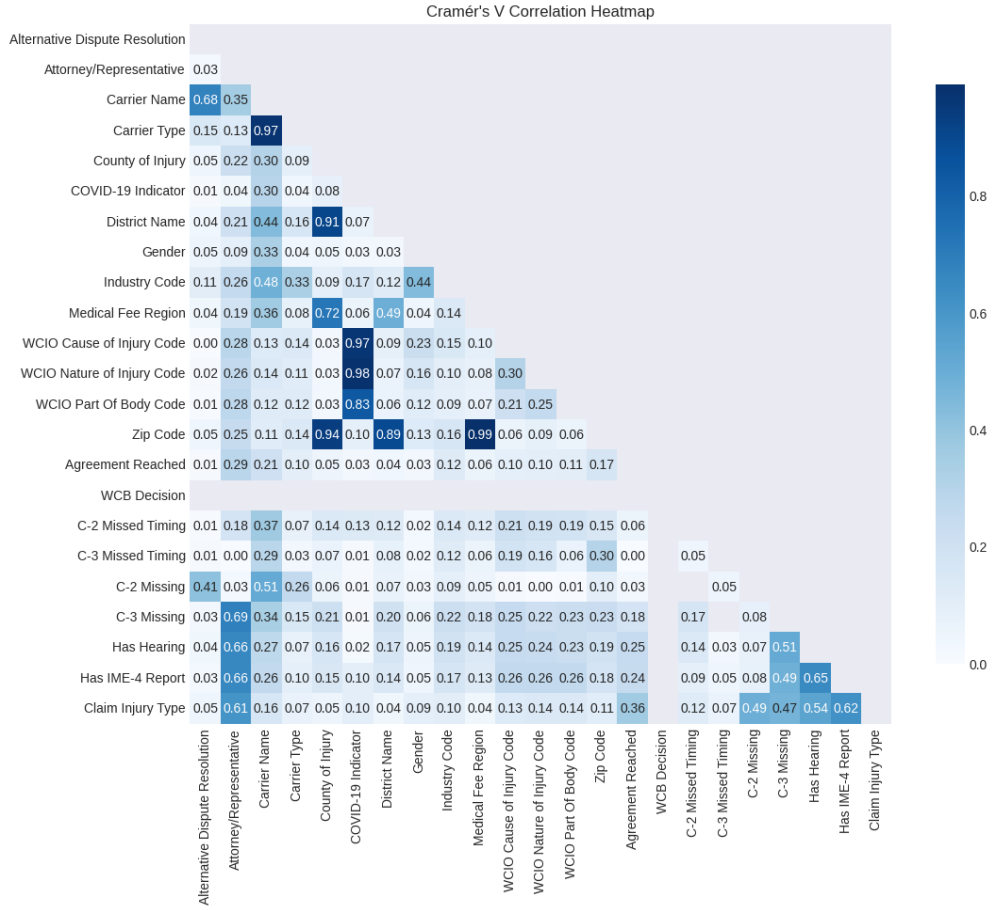
Annex 6 - Numeric Features Spearman Correlation Heatmap



Annex 7 - Date Features Spearman Correlation Heatmap



Annex 8 - Categorical Features Cramér's V Heatmap



Annex 9 – Simple Encodings

- Values 'U' in 'Alternative Dispute Resolution' are set to null.
- Values 'Y' and 'N' of 'Alternative Dispute Resolution', 'Attorney/Representative' and 'COVID-19 Indicator' are set to '1' and '0', respectively.
- Values 'M' and 'F' of 'Gender' are set to '1' and '0', respectively.
- 'Accident Date', 'Assembly Date', 'C-2 Date', 'C-3 Date' and 'First Hearing Date' were separated into 3 columns each, one for each time component year, month and day.
- Encoding 'Claim Injury Type' using OrdinalEncoder(), since numbering indicates severity.

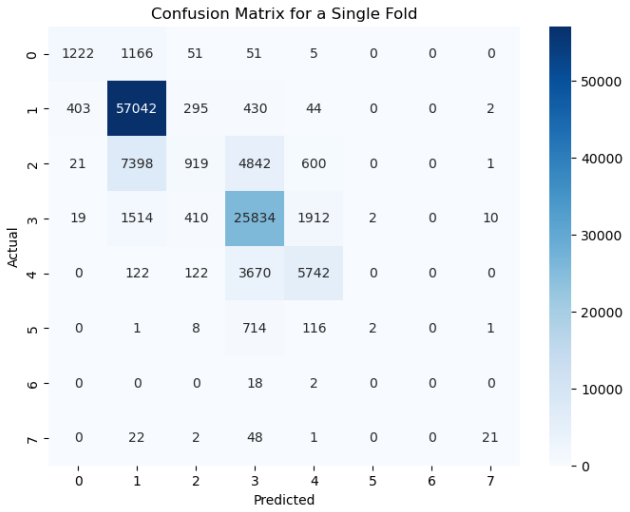
Annex 10 - Feature Selection Custom Classes

Class	Purpose
RemoveHighSpearmanCorrelation	Removes highly correlated features based on Spearman Correlation
XGBFeatureSelector	Uses XGBClassifier() and feature_importances_ to select the n best features
DT_FeatureSelector	Uses DecisionTreeClassifier() and feature_importances_ to select the n best features

Annex 11 - Other Helper Functions

Function	Purpose
ensure_dataframe	Converts the input data into a pandas DataFrame
inspect_columns	Prints the list of columns
evaluate_feature_intervals	Evaluates different number of features using cross validation
avg_score	Calculates the f1 score for training and validation data using cross validation
show_results	Returns a dataframe with the results of the 'avg_score' function

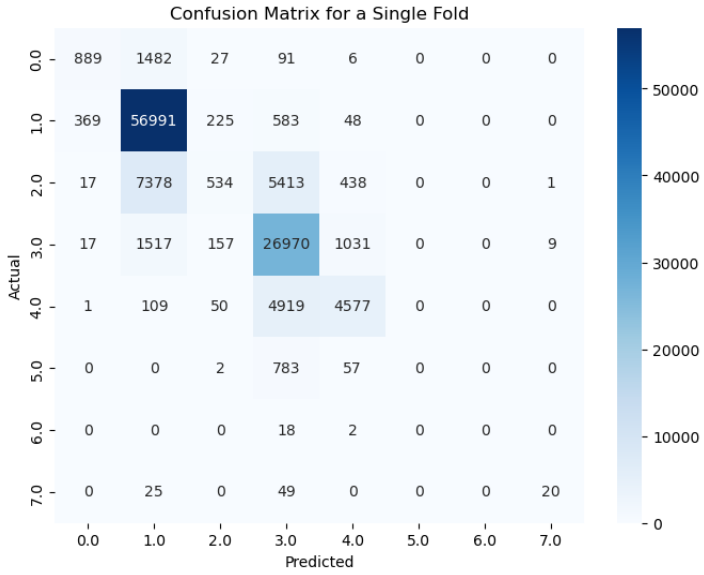
Annex 12 – XGBoost Confusion Matrix



Annex 13 – MLP Grid Search Parameters

Algorithm	Parameter	Values
SelectKBest	k	range(5, 50, 3)
MLPClassifier	solver	'lbfgs', 'sgd', 'adam'
	learning_rate	'constant', 'invscaling', 'adaptive'
	learning_rate_init	0.1, 0.01

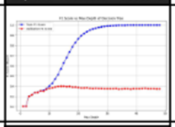
Annex 14 - MLP Confusion Matrix



Annex 15 - MLP Scores

```
Model Evaluation X_train:
                 Neural Networks
Accuracy Macro                NaN
Precision Macro          0.591247
Recall Macro             0.396287
F1 Macro                 0.415608
-----------------------------------
Model Evaluation X_val:
                 Neural Networks
Accuracy Macro                NaN
Precision Macro          0.544429
Recall Macro             0.389821
F1 Macro                 0.407112
```

Annex 16 - DecisionTreeClassifier Parameters Decision

Parameter	Description	Values tested	Output	Values chosen	Justification
criterion	metric for split quality	Gini, Entropy	Train Val Gini 0.999995 0.370986 Entropy 0.999995 0.379927	Entropy	highest val score
splitter	strategy for splitting nodes	Best, Random	Train Val best 0.999995 0.377527 random 0.999995 0.370234	Best	highest val score
max_depth	limits tree depth, avoiding overfitting by constraining the number of splits	None,20,30,40	Train Val full 0.999995 0.378286 depth20 0.840339 0.3946 depth30 0.985667 0.380737 depth40 0.999379 0.380348		do a graph to check all values between 1 and 48 because we know that full depth = 48 nodes
		range(1,49)		12,13,14	highest val score with less probability of overfitting (the train score is lower in these values)
min_samples_split	minimum samples required to split a node	2,10,50,100	Train Val Original 0.446715 0.394797 dt_min10 0.431154 0.394172 dt_min50 0.412968 0.394977 dt_min100 0.401198 0.391429	2,50	highest val scores (really close to each other so we will include both)
min_samples_leaf	minimum samples per leaf	1,10,50,100	Train Val Original 0.446692 0.394863 dt_min_sam10 0.416611 0.395178 dt_min_sam50 0.399234 0.391905 dt_min_sam100 0.401234 0.397615	10,100	highest val scores (really close to each other). We didn't choose 1 due to overfitting
max_features	number of features considered for split, reduces complexity	none,2,0.5,sqrt,log	Train Val None (Original) 0.446741 0.394882 2 0.318096 0.293825 0.5 0.448948 0.398383 Sqrt 0.40509 0.363377 Log2 0.376725 0.337013	none,0.5	highest val scores (really close to each other so we will include both)
max_leaf_nodes	maximum number of leaf nodes	none,10,20,30,40	Train Val None (Original) 0.445954 0.395134 dt_maxleaf10 0.301563 0.301548 dt_maxleaf20 0.339337 0.339291 dt_maxleaf30 0.338571 0.33863 dt_maxleaf40 0.347504 0.347465	none	highest val score
min_impurity_decrease	minimum impurity reduction required to split	0,0.02	Train Val Original 0.446305 0.394725 dt_impurity02 0.301563 0.301548	0	highest val score

Annex 17 - Feature Selection - Number Of Features Output

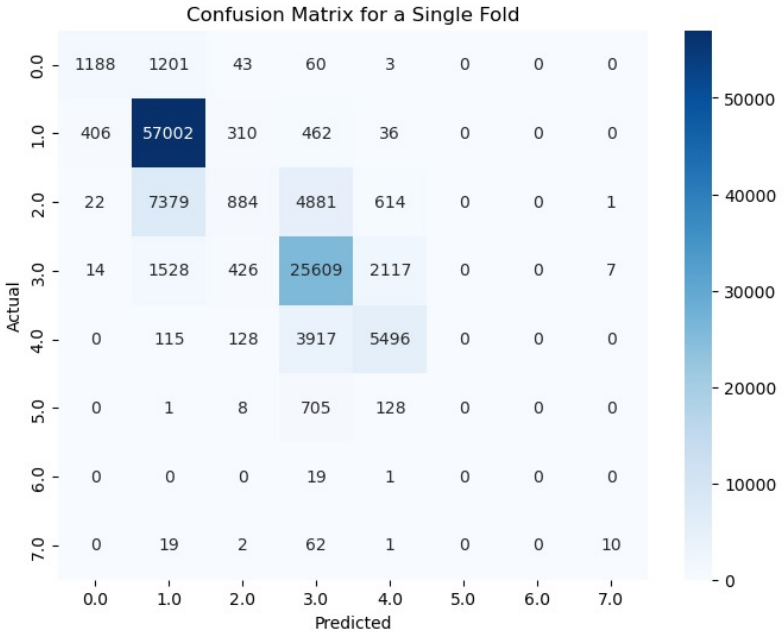
1. XGBoost

	n_features	mean_train_score	mean_val_score	overfit
0	30	0.390586	0.348990	0.041596
1	33	0.393472	0.347011	0.046462
2	36	0.395679	0.349796	0.045884
3	39	0.396184	0.353410	0.042774
4	42	0.397581	0.347486	0.050094
5	45	0.403307	0.350658	0.052649
6	48	0.403081	0.351232	0.051849
7	51	0.402498	0.346174	0.056325
8	54	0.409496	0.351783	0.057712
9	57	0.411396	0.348630	0.062766
10	60	0.411002	0.349767	0.061234
11	63	0.414910	0.347196	0.067714
12	66	0.415425	0.347162	0.068262
13	69	0.416568	0.348687	0.067881

2. Decision trees

	Treshold	Number of Features	Train F1	Val F1
0	0.025	0	0.413261	0.348611
1	0.020	1	0.415204	0.345579
2	0.015	3	0.413730	0.346695
3	0.010	9	0.415895	0.348459
4	0.005	19	0.413688	0.348938
5	0.001	35	0.417709	0.347439
6	0.000	44	0.415672	0.348593

Annex 18- DT - Confusion Matrix and Classification Report



Classification Report for a Single Fold:

	precision	recall	f1-score	support
0.0	0.7288	0.4762	0.5760	2495
1.0	0.8477	0.9791	0.9087	58216
2.0	0.4908	0.0641	0.1135	13781
3.0	0.7170	0.8622	0.7830	29701
4.0	0.6546	0.5692	0.6089	9656
5.0	0.0000	0.0000	0.0000	842
6.0	0.0000	0.0000	0.0000	20
7.0	0.5556	0.1064	0.1786	94
accuracy			0.7856	114805
macro avg	0.4993	0.3822	0.3961	114805
weighted avg	0.7456	0.7856	0.7408	114805

Annex 19 - Random Forest Parameters Decision

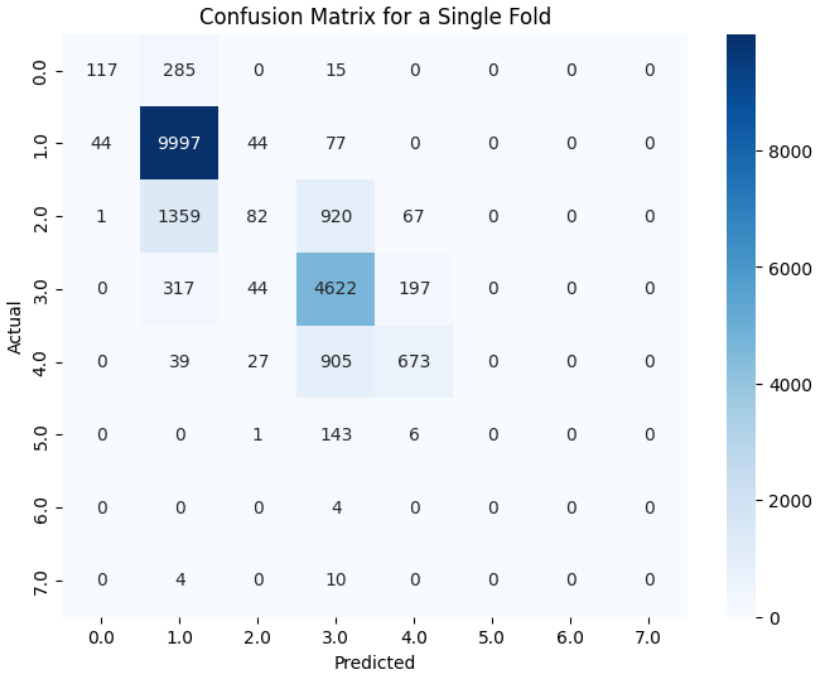
Parameter	Description	Values	Final
Number of Estimators	Number of decision trees in the forest	[100,200,300]	100
Maximum Depth	Maximum depth of each decision tree	[10,20,None]	10
Minimum Samples per Split	Minimum number of samples required to split a node	[2,5,10]	2
Minimum Samples per Leaf	Minimum number of samples required to be a leaf node	[1,2,4]	1

Annex 20 - Random Forest Evaluation Metrics

```
Model Evaluation X_train:
                 Random Forest + SelectKBest
Accuracy Macro                       0.813825
Precision Macro                      0.626347
Recall Macro                         0.376723
F1 Macro                             0.402916
-------------------------------------------------
Model Evaluation X_val:
                 Random Forest + SelectKBest
Accuracy Macro                       0.773375
Precision Macro                      0.426069
Recall Macro                         0.316996
F1 Macro                             0.324618
```

```
Model Evaluation X_train:
                 Random Forest + SMOTE + SelectKBest
Accuracy Macro                         0.750537
Precision Macro                        0.543863
Recall Macro                           0.678718
F1 Macro                               0.550978
-------------------------------------------------------
Model Evaluation X_val:
                 Random Forest + SMOTE + SelectKBest
Accuracy Macro                         0.713300
Precision Macro                        0.379054
Recall Macro                           0.478396
F1 Macro                               0.387608
```

Annex 21 - Random Forest Confusion Matrix



Annex 22 - Logistic Regression Best Parameters

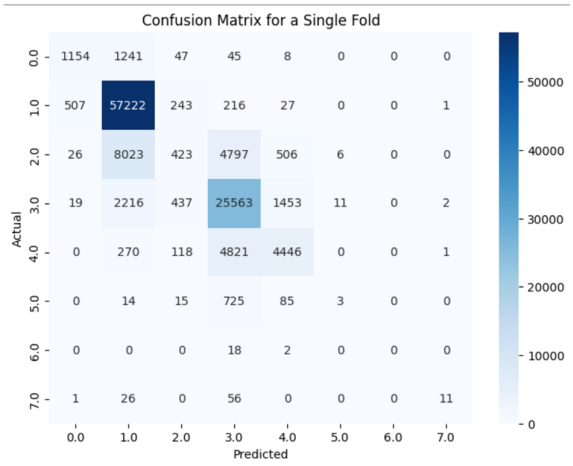
```
Fitting 5 folds for each of 384 candidates, totalling 1920 fits
Best Parameters: {'model__C': 10, 'model__l1_ratio': 0.2, 'model__penalty': 'l1', 'model__solver': 'liblinear'}
Best Score: 0.40897484704172876
F1-score weighted:  0.72336836598011
F1-score macro:  0.39291706072616295
```

Annex 23 – Logistic Regression Parameters

Parameter	Description	Values	Best Values
C	Inverse of Regularization Strength	[0,001, 0.01, 0.1, 1, 10, 100]	10
penalty	Penalty	['l1', 'l2', elasticnet', None]	l1
solver	Model Solver	['liblinear', 'lbfgs', 'saga', 'newton-cg']	liblinear
l1_ratio	Elastic Net Mixing Parameter	[0.2, 0.5, 0.8, None]	-

0,40897

Annex 24 - Logistic Regression XGBoost Based Feature Selector Confusion Matrix and Classification Report

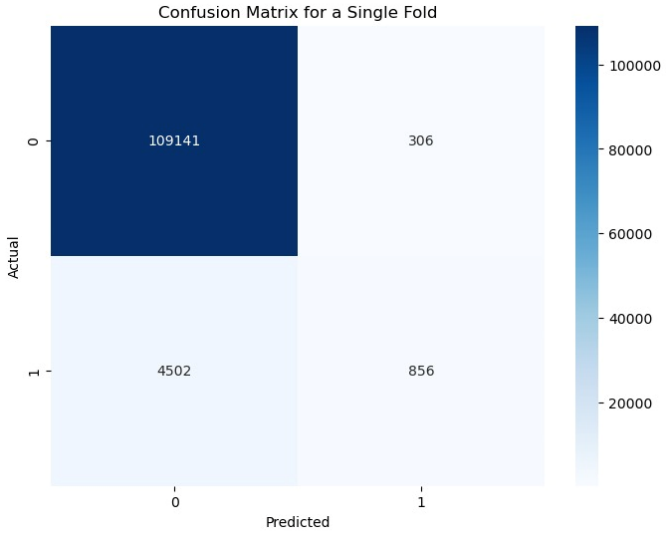



```
Classification Report for a Single Fold:
              precision    recall  f1-score   support

         0.0     0.6760    0.4625    0.5493      2495
         1.0     0.8292    0.9829    0.8995     58216
         2.0     0.3297    0.0307    0.0562     13781
         3.0     0.7054    0.8607    0.7753     29701
         4.0     0.6812    0.4604    0.5495      9656
         5.0     0.1500    0.0036    0.0070       842
         6.0     0.0000    0.0000    0.0000        20
         7.0     0.7333    0.1170    0.2018        94

    accuracy                         0.7737    114805
   macro avg     0.5131    0.3647    0.3798    114805
weighted avg     0.7162    0.7737    0.7218    114805
```

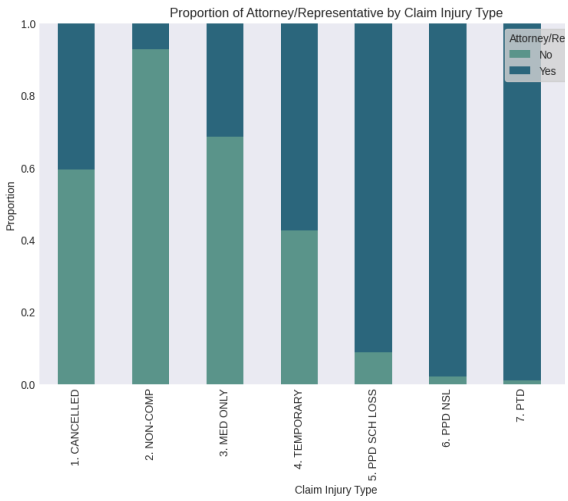
Annex 25 – Ageement Reached Confusion Matrix



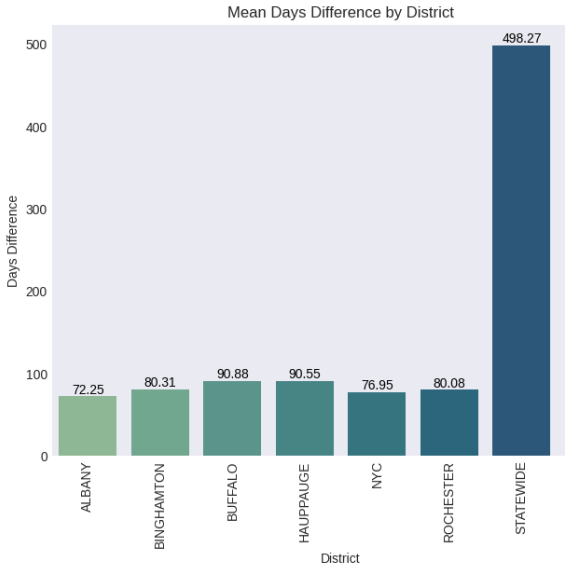
Appendix

Appendix 1 - Multivariate Analysis

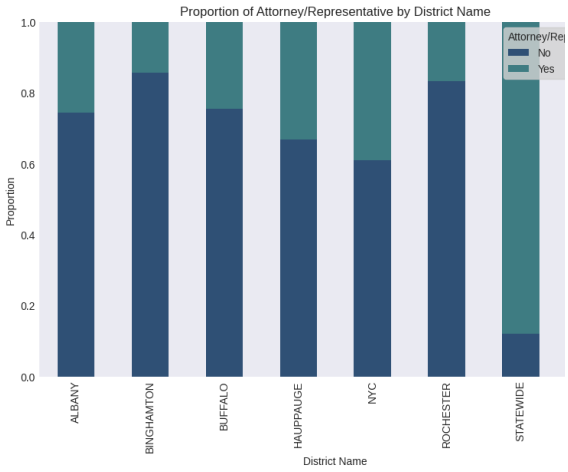
Regarding multivariate analysis, we highlight the following plots:



Higher values of 'Claim Injury Type' highly benefit from having 'Attorney/ Representative', with values above 5 being almost exclusively associated with claims from individuals with legal representation.



On average, claims are assembled within three months of the accident date. However, in the Statewide district, the average assembly time is significantly longer at 1.5 years.



The graph shows that 85% of claims in the Statewide district involve legal representation. This, combined with the slower claim assembly process in the district, suggests a potential link between legal representation and delayed assembly times