TO GRANT OR NOT TO GRANT: DECIDING ON COMPENSATION BENEFITS

Project of Machine Learning

Abstract

This project consists of creating a classification model that can accurately predict the final decision for WCB regarding the Claim Injury Type variable. To do this, several data points are provided that fall within the years 2020 to 2022. First we did an exploration of the data in order to see the data types, the missing values, check for duplicates, explore features and then, we start to preprocessing this data: deal with incoherences, handle missing values, remove outliers, create new features and encode variables. After that, we did feature selection and then used six different models: CatBoosted, XGBoosted, Decision Trees, Naive Bayes, StackEnsemble and Voting Ensemble. Finally, we improved the optimizations of the models, using Random Search and Search Space based gridsearch. We performed random search on the 4 basic models and due to the increased complexity of CatBoost and XGBoost we created a search space based on the results of the random search to iterate over both models and have a more precise optimization. We conclude that the best model to make predictions is XGBoost, so we implement the XGBoost with kfold, that showed overfitting but by far the greatest results. In adition, we create an application, named GrantApp, that enables the prediction of the 'Claim Injury Type' when new inputs are given.

Ana Beatriz Farinha, 20211514
Francisco Capontes, 20211692
Rui Lourenço, 20211639
Sofia Gomes, 20240848

Fall/Spring Semester 2024-2025

Table of Contents

1 Introduction

The WCB is responsible for deciding on compensation claims whenever it becomes aware of an accident at work, but reviewing all claims individually is a complicated and time-consuming process. As such, the WCB asked for help to automate decision-making whenever a new claim is received by creating a predictive model.

To solve their problem, our group went through several phases, namely: data exploration, cleaning the data, preprocessing, multiclass classification, among others. We used six models (CatBoosted, XGBoosted, Decision Trees, Naive Bayes, Stack Ensemble and Voting Ensemble) and in the end we compared the models in order to obtain the best decision to predict the model. To optimize the models, we created a grid search.

Through this report, you can get a detailed explanation of everything that was done to achieve our goal. The code that made this project possible is available at the following link https://github.com/Gomsofi06/Machine_Learning_NOVAIMS.git

2 Data Exploration and Preprocessing

Before creating any model, it is necessary to explore and clean the data we have because our data's quality will affect the quality of our model. As such, we started by exploring our data and then treated the problems found during the previous phase. Afterwards, we created new features and encoded the categorical ones.

2.1 Initial Exploration

First, we did an initial exploration of the data, such as looking at the number of rows and columns in each dataset, looking at the first ten rows and looking at the types of each column. After an analysis, we can conclude that the **train dataset** has eleven numerical columns and twenty-one categorical columns. There are columns with missing values ('C-3 Date', 'IME-4 Count' and 'Industry Code') and the column 'OIICS Nature of Injury Description' only contains missing values. In addition, we also started analysing the numerical features from where we found that the average age to suffer injuries is 42.11 (middle-aged people), with an unusual maximum of 117, indicating potential incoherences. As for the 'Average Weekly Wage', it has a maximum of 282,807.90, a possible outlier, and a mean of 545.37. 'Birth Year', indicates that most people were born between 1965 and 1989, consistent with working-age individuals. The variables 'Age at Injury', 'Weekly Average Wage' and 'Birth Year' all have a minimum of 0 indicating potential incoherences and missing values. As for 'IME-4 Count,' it has a minimum value of 1 and a maximum of 73, indicating a potential outlier. Notably, this variable does not include a value of 0 to signify that no complaints were made. Therefore, we assume that the missing values in this column were intended to represent 0. The **test dataset** has ten numerical columns and nineteen categorical columns. The columns containing missing values are 'C-3 Date', 'IME-4 Count' and 'First Hearing Date' and the column 'OIICS Nature of Injury Description' only contains missing values.

2.2 Data Types

To facilitate our exploration we had to verify the data types of our data and transform some to the most suitable data type.

2.3 Duplicates

In order to check the duplicates, we took two perspectives: checking the duplicates in the 'Claim Identifier', a column that should be unique because it is an identifier, and searching for duplicate rows. By analysing the first perspective, we noticed that there are two rows with the same Claim Identifier. As these two rows are almost empty, we decided to remove them from the dataset. Following, we define the column 'Claim Identifier' as our index. In the second perspective, we noticed that there are 18348 duplicate rows and decided to remove them since most of them are empty.

2.4 Missing Values

Our next step was to explore the remaining missing values so that we could treat them. First, we checked that all of our rows had a target. Then we saw the percentage of missing values per column and conclude that columns like 'OIICS Nature of Injury Description', 'IME-4 Count', and 'First Hearing Date' have over 60% missing, with some 100% missing. The missing values are going to be treated in the next phase(pre-process). To understand better the relationship between columns based on their patterns of missingness, we did a dendrogram 1. As we can see in the diagram, there are two groups: the orange one and the green one. The first one contains the variables WCIO Part of Body Description, C-2 Date, WCIO Cause of Injury Description, and WCIO Nature of Injury Description. These variables have a strong special relationship in their missing values. The green group comprises three closely related subgroups. There is a large separation between the green group and the orange group, as shown by the considerable Euclidean distance in the blue dendrogram. In order to see the correlation between missing values in different columns, we created a heatmap plot 2. We can draw some conclusions from this graph. A high correlation of 0.9 suggests that when one is missing, the other often is too, like in WCIO. A correlations of 0.5 to 0.7 imply a weaker but still noteworthy relationship, like in WCIO and Industry code. A correlation close to zero or negative imply little to no relationship, suggesting that missingness in one column does not predict missingness in the other, like in Accident date, average weekly wage and others.

2.5 Exploring Features

Our last step was to do a more in-depth exploration of our variables. To do so, we started by checking the distribution of our target 3. Here, we can conclude that the frequency of the 'Claim Injury Type' presents a large discrepancy between the values. The most frequent is NON-COMP and the ones with the lowest frequency are PTD and DEATH. Then, we check the numerical variables, which are five: Age at injury (the mean is 42 years and 75% of the people have 54 years when they suffer an injury) 4, Average Weekly Wage (the mean is 491) 5, Birth Year (range from 0 to 2018) 6, IME-4 Count (the mean is 3) 7 and Number of Dependents (The range is 0 to 6 and the mean is 3) 8. After that, we tried to find correlations between the variables with a heatmap, but those are almost non-existent because the data is not clean yet. Additionally, we did some scatter plots 9. In the first plot, we can detect some outliers of people with really high weekly wages when the age at injury is between 20 and 60 years. In the second plot, we do not see any clear relationship between the two variables. Regardless of the number of dependencies, the age at injury values don't seem to change much. In the third plot, we can see that the average only has values when the person's date of birth is greater than 1850 or equal to zero (this should be removed). In the last plot, we can see that the

number of dependents between 1 and 4 increases when the average weekly wage is higher than 1.

About the categorical variables, we saw some statistics but most of these variables are codes and the only variable that appears is 'Agreement Reached', which is binary, with a mean of 0.046665. That signifies that most of the cases did not reach to an agreement or are yet to reach. Also, we noticed a negative value in the variable 'WCIO Part Of Body Code' that corresponds to multiple injuries, so we decided not to change that value. After that, we saw the other categorical variables: Alternative Dispute Resolution (Majority with 'N') 10, Carrier Name (STATE INSURANCE FUND as the largest category) 11, Claim Injury Type (8 distinct categories), County of Injury (63 unique counties, with significant representation from SUFFOLK and QUEENS), Attorney/Representative (Majority of claims without legal representation) 12, Gender (includes unknown values such as 'U' and 'X', which will be replaced with 'Unknown'. We identified the presence of 'UNKNO' in the columns 'WCIO Cause of Injury Code', 'WCIO Nature of Injury Code', 'WCIO Part Of Body Code' and 'Zip Code' and replaced them with 'Unknown'.) 13 and Industry Code Description (Significant representation in sectors like Health Care and Public Administration). Furthermore, we created a graph with the number of cases in each county 14. From this graph, we can see a concentration of cases on the counties on the bottom right. Finally, we explore the date variables and we can see that the earliest 'Accident Date' was in 1961 but the first 'Assembly Date' was in 2020. Additionally, 'C2-Date' and 'C3-Date'where in 1996 and 1992 respectively. In addition, we did a plot see that more clearly 15. We can see that most accidents appear between the end of 2019 and the end of 2022 with the majority in January and March of 2020 and 2022 and the summer months of 2021 and 2022 also have a high number of accidents registered.

After an initial exploration, the next step is preprocessing the data. It involves preparing raw data into a suitable format for analysis or modelling. For that, we have to deal with incoherences, handle missing values, remove outliers, create new features and encode variables. After, our features will go through a process denominated feature selection where the best features to train our model will be selected.

2.6 Incoherences

That said, we found six inconsistencies. The first is related with the 'Age at Injury' column contains some values with zero and others with an age lower than fourteen, which is not legally allowed in New York. So, we applied a minimum and a maximum limits to this column and the values outside these limits are replaced by NaN. We also checked if there are people older than Age at Injury and concluded that there are not. The 'Birth Year' column has values that are outside the limits that we defined (below 1906 and above 2011), so we did the same as before. The 'Average Weekly Wage' column should not have values equal to zero, and we found that there are some occurrences of this. Here we also applied a minimum and a maximum limits to this column and the values outside these limits are replaced by NaN. We also compared whether the Industry Code and Industry columns have the same description in the dataset and we found that they do.

2.7 Handling Missing Values

During our previous exploration of the missing values, we noticed that the column 'OIICS Nature of Injury Description' only has missing values, so we removed it. As for the column 'Accident Date', while exploring the dataset, we noticed that in some cases that date is the same as 'Assembly Date'. As such, we decided to replace the missing

values with the other date. In regards to the 'Age at Injury' column, the missing values were calculated using the formula 'Age at injury' = 'Accident Date' - 'Birth year'. To calculate the 'Birth Year' column, we use the formula 'Birth year' = 'Accident date' - 'age at injury'. With regard to the 'IME-4 Count' column, as this variable represents the number of IME-4 forms received per claim and is never 0, we came to the conclusion that the missing values represent that no IME-4 forms were received. Therefore, we decided to replace the missing values with 0. In the variable 'First Hearing Date', a blank space means that the complaint has not yet had a hearing. In addition, missing values in 'C2-Date' and 'C3-Date' mean that the event has not yet taken place. To distinguish these NAs from the true missing values, we decided to replace them with -1. This will be done in the feature engineering section.

2.8 Remove Outliers

To remove the outliers, we sought extreme values that we could clearly identify in the boxplot of each of our variables. After removing outliers, we also applied transformations to some of the variables to reduce the skewness of their distribution. With regards to the 'Age at Injury' column, we consider that people aged 90 and over are already outliers, so we remove them. Regarding the 'Average Weekly Wage' column, we have removed the 5 outliers that can be seen from the graph. After, that, we checked the distribution without those outliers. It was right-skewed, so we decided to apply a log transformation to reduce the skewness. The next variable is 'Birth Year'. The distribution is already normal but some outliers can be seen in the boxplot. As such, we decided to remove them. The next feature is 'IME-4 Count'. Its distribution is extremely right skewed and we can detect some outliers in the boxplot of the variable. As such, we decided to start by removing the outliers. Our next step was to reduce the skewness of the variable. As it has the value 0, we can not apply a log distribution. As such, we decided to use the square root which has to be applied in both datasets. As for the last numerical variable 'Number of Dependents', there were no outliers and it was uniformly distributed.

2.9 Feature Engineering

To make Feature Engineering, we first created new features and then transformed existing ones. We started by creating new columns to capture the elapsed time between key dates, which can help understand delays or the timeline of events. After that, we create a column that indicates the season in which the accident happened. Another column that we created was to indicate if the accident happened on a weekend or a holiday. Another feature is a column that indicates how dangerous the industry of the worker is. Also, we created the features Relative Wage, Financial Impact Category and Age Group, the first expresses if the Average Weekly Wage was above or below of the median, the second categorizes the impact of claim based on the Average Weekly Wage and Number of Dependents and lastly we group different ranges of Age at Injury into different ranked groups. In this table we can see the features that we created and more information about them 16. After creating the new features, the next step was to modify the feature 'Carrier Type' and 'Gender'. In the Carrier Type column, there are three similar values which represent different types of special funds. Only 9 special funds could be identified, leaving 1023 unknown. As such, we decided to join all of them in the unknown special funds. In the Gender column, we replaced the values U and X with Unknown. The value U typically indicates "unknown" or unspecified gender, while X may signify a non-binary identity. By combining these two values into Unknown, we

simplify the data, making it easier to analyse and interpret 17.

2.10 Visualizations

After all the modifications that we did, lets check how our variables changed. We started with the distribution of the numerical features 18. About the graphics of 'Age at Injury', the distribution shows a bimodal pattern, with peaks around age 35 and 55. The ages range primarily from 20 to 70 years old. In the boxplot about the same variable, we can see that the median age is around 40 years, the interquartile range is approximately between 30 and 55 years and exists some outliers above age 70 and below age 20. About the graphics of 'Average Weekly Wage', there is a highly right-skewed distribution with majority of wages clustered at lower values. Also, we can see that most common wages appear to be under $20,000 weekly In the respectively boxplot, exists numerous high-value outliers, most wages concentrated in lower range and several extreme outliers above $60,000 weekly. About the graphics of Birth Year, we can see similar pattern to Age at Injury (inversely), the distribution spans from approximately 1940 to 2010 and a bimodal distribution with peaks around early 1960s at Mid-1980s. The respective boxplot show that the median birth year is around 1975 and most birth years fall between 1960 and 1990. About IME-4 Count there is a extremely right-skewed distribution with a vast majority of cases of 0 or 1. The respective boxplot shows multiple outliers above 2 counts, maximum values reaching 6 and most cases concentrated at lower values. Finaly, about Number of Dependents, we can see a discrete distribution with clear peaks at whole numbers, ranges from 0 to 6 dependents and most frequent values are 0 and 1 dependents.

In order to see the correlation between the numerical features, we create a plot 19. The correlations improve a lot since the modifications, how we can see. The column Number of dependencies is the only variable that don´t have correlation with either of the other columns. We also did a multivariate exploration in order to our target ´Claim Injury Type´. First, we create plots for the numerical features 20. About the variables Age at injury, the age spans a wide range, but there are significant concentrations in younger age groups (20-40 years). The more frequent is TEMPORARY, suggesting that younger individuals might be more prone to temporary injuries. About Birth Year, there is a strong negative relationship between birth year and age at injury (as expected, since more recent birth years correspond to younger ages). About IME-4 Count, the predominant is low (between 0 and 2), but certain injury types, such as "TEMPORARY," are associated with higher counts. This suggests that certain injury types may require more IME-4 exams for evaluation. Finally, we create plots for the categorical features 21. About Carrier Type, is more concentrated in lower values (0-2). About 'Gender', there is a clear separation of gender (0 and 1), but the relationship between gender and injury types does not seem to vary significantly. About District Name, certain districts (especially those with higher values) have a greater concentration of cases.

2.11 Variable Encoding

To use the categorical variables in most models, we have to encode them, so we used four different encoders: One-Hot Encoder (categorical features that are not ordinal and that do not have many unique values) for Alternative Dispute Resolution, Attorney/Representative, COVID-19 Indicator, Gender and Medical Fee Region; Frequency Encoder (only be applied after we split our data) for County of Injury, District Name, Zip Code, Industry Code, WCIO Cause of Injury Code, WCIO Nature of Injury Code and WCIO Part of Body Code; Sine Cosine Encoder (cyclic feature) for Accident Season; Label En-

coder (applied in our target which only exists in the training dataset) for Claim Injury Type.

3 Multiclass Classification

In this part of our project, we began by defining the macro F1-score as our evaluation metric, as it effectively addresses the class imbalance in our dataset by treating each class with equal importance. This metric was calculated three times: once for the training dataset, once for the validation dataset, and finally for the test dataset (on Kaggle).

We began by experimenting with several models for classification, including Decision Trees (Creates a classification model based on hierarchical divisions of the data), Logistic Regression (Performs binary or multiclass classification by modeling the probability of an event occurring), Instance-Based methods (makes predictions directly from the training data without creating an explicit model), Random Forest (builds multiple decision trees and combines their predictions to improve accuracy), and Neural Networks (Models complex relationships in the data using layers of connected neurons). However, we observed that the F1-score values were consistently low (22). To address this, we pivoted to testing six additional models: CatBoost, XGBoost, Decision Trees, Naive Bayes, Stack Ensemble, and Voting Ensemble. For the CatBoost and XGBoost models, we employed Grid Search to streamline the hyperparameter tuning process, allowing us to adjust parameters and identify configurations that best fit our data. Within this Grid Search, we evaluated a range of hyperparameters to optimize performance. Additionally, we implemented Random Grid Search, focusing on the CatBoost, XGBoost, and Decision Tree algorithms. An important note is that the Grid Search processes were applied both to all features in our dataset and to the reduced feature set determined during the feature selection phase.

However, these new models had a downside: they tended to overfit the training dataset. In the models that overfit, the F1-score was significantly higher in the training dataset than in the validation dataset. Interestingly, the validation F1-score for these overfitting models was approximately 0.1 higher than that of the models that did not overfit. As a result, we decided to accept the overfitting in order to achieve better performance.

3.1 Feature Selection

After scaling the features using StandardScaler, we performed feature selection to identify and select a subset of the most relevant variables for use in the models. This process incorporated at least one method from each category: filter, wrapper, and embedded approaches.

For numerical variables, we applied various feature selection techniques, including XGBoost, Light Gradient Boosting, Decision Tree, Lasso, and Logistic Regression using Recursive Feature Elimination (RFE). For categorical variables, we implemented methods such as Mutual Information, XGBoost, Light Gradient Boosting, Decision Tree, Lasso, Logistic Regression RFE, and the Chi-square test. The results and conclusions for numerical variables are presented in Table 23, while those for categorical variables are detailed in Table 24.

The criteria we used to evaluate features were based on a scoring system, where the score represents the number of selection methods that identified the feature as relevant. Features with scores in the range of 0–2 were dropped from consideration. Features scoring 3–4 were subjected to further experimentation by including and excluding them to evaluate their impact on model performance. Features with scores in the range of 5–6

were retained for modeling. As such, we created 3 sets of features to test our models. The first one is the features that were considered essential, the second one adds the features that we may keep or remove and the last one is all the features.

3.2 Models

The process of training our models was similar for all of them. The only thing we've changed is the algorithm and the definitions associated with each one. Firstly, we define the settings and apply the model with them, then the next step is check the performance. After that we split the data, removed outliers, applied frequency encoding, NA imputed, created new features, fited the scaler, fit the model and predict the model.

3.2.1 CatBoosted

The CatBoosted model is a gradient boosting model that efficiently handles categorical data without requiring extensive preprocessing and reduces overfitting using techniques like order boosting. It creates a series of decision trees, where each tree tries to correct the errors of the previous ones. For this model, the settings that we defined are "iterations": 1000, "learning_rate": 0.11, "depth": 6, "l2_leaf_reg": 5, "bagging_temperature": 0.4, "random_state": random_state. The performance that we have got was an average score of 0.46 for train F1 and 0.42 for test F1.

3.2.2 XGBoosted

The XGBoosted model is a gradient boosting model optimized for speed, accuracy, and efficiency on structured data. It builds decision trees sequentially, minimizing the residual error at each step with a gradient boosting approach. For this model, the settings that we defined are "n_estimators": 200, "learning_rate": 0.2, "max_depth": 7, "subsample": 0.9, "colsample_bytree": 0.9, "gamma": 0.3, "random_state": random_state. The performance that we got was an average score of 0.65 for train F1 and 0.42 for test F1.

3.2.3 Decision Tree

The decision tree model splits data into simple hierarchical rules for interpretable decision-making. It analyzes the data by dividing it into subsets based on conditions until each subset is sufficiently pure or meets a predefined criterion. For this model, the settings that we defined are "min_samples_split": 10, "min_samples_leaf": 4, "max_depth": 20, "criterion": "entropy". The performance that we have got was an average score of 0.43 for train F1 and 0.31 for test F1.

3.2.4 Naive Bayes

The Naive Bayes model is a probabilistic classifier based on Bayes' theorem and the assumption of independence between variables. Calculates the probability of each class for a data instance, assuming that all independent variables contribute equally to the result, and classifies the instance in the class with the highest probability. For this model, we don´t need to define the settings. Just apply the GaussianNB model. The performance that we have got was an average score of 0.25 for train F1 and 0.24 for test F1. This algorithm has good recall and works well on ensembles, but has bad scores.

3.2.5 Stack Ensemble

The stack ensemble combines multiple models using a meta-model to improve overall performance. It trains various base models, captures their predictions and uses them as input for an additional model (meta-model) that learns to combine these predictions optimally. The models that we combine was NaiveBayes, CatBoost, XGBoost and Decision Tree. For each model, we have to adjust the settings, so we defined CatBoost config 1, XGBoost config 1, decision tree and default parameters. The final estimator that we used to apply the Stacking Classifier was Logistic Regression. The performance that we have got was an average score of 0.31 for train F1 and 0.30 for test F1.

3.2.6 Voting Ensemble

The voting ensemble aggregates predictions from multiple models through majority voting or weighted averaging. It makes predictions with several independent models and decides the final result based on a vote (for classifiers) or average (for regressions), combining the strength of the models to improve accuracy. The models that we combined were NaiveBayes, CatBoost, XGBoost and Decision Tree. For each model, we have to adjust the settings, so we defined CatBoost config 2, XGBoost config 2, decision tree and default parameters. Then we defined the VotingClassifier with the parameters estimators=base_models, voting='soft' and weights=[1.0, 2.0, 2.0, 1.0]. The performance that we have got was an average score of 0.65 for train F1 and 0.41 for test F1.

3.2.7 Final Model

For the final predictive model we used the previously mentioned configuration for XGBoost with all features, we implemented a pipeline that used a Stratified K-fold with 6 folds to achieve better results. The model is trained for each fold and the predictions are then averaged by the number of total folds and the final prediction is the class with the highest averaged probability. The performance that we got was an average score of 0.75 for train F1 and 0.44 for test F1.

3.2.8 Comparing models & GridSearch

In order to optimize the models we made use of Random Search and Search Space based gridsearch. We performed random search on the 4 basic models and due to the increased complexity of CatBoost and XGBoost we created a search space based on the results of the random search to iterate over both models and have a more precise optimization. For this process we used the library Ray to mitigate the intensity of the process by better allocating the computer resources and running more than one model at the same time, a total of 24 cpus ran 4 trails at same time each with 6 cpus. What most affects the models performance is how good the EDA and preprocessing was, as such we saw slight increases in the score 25. We decided that the best model would be XGBoost, despite looking the most overfitted (because uses decision trees as base learners) it had the same avg val score as CatBoost, but it was faster and it alows us to implement the XGBoost with kfold, that showed overfitting but by far the greatest results.

4 Open-Ended Section

Our final step was to create an application, named GrantApp, that enables the prediction of the 'Claim Injury Type' when new inputs are given. The inputs can be written

directly in the app, comma separated, or they can be imported as a csv. The new data will go through the data pipeline that we created where the transformations applied to the test dataset apply to this new input. After the new data goes through the pipeline, the features used to train the model are selected and the target is predicted using our final model. The predictions appear in the GrantApp.

5 Conclusion

The main objective of this project was to develop a classification model capable of accurately predicting the type of WCB decision in relation to the Claim Injury Type variable. During the process, we explored the data, identified problems such as missing values, inconsistencies and outliers, and implemented various attribute engineering and variable selection techniques to maximize model performance. Finally, we developed an application, GrantApp, which applies the predictive model to new input data, optimizing the WCB's decision-making process.

The results obtained, especially with the XGBoost model optimized with K-fold validation, met initial expectations of identifying an efficient and accurate model. However, we observed that although XGBoost showed the best performance (F1-score of 0.44 for the validation), it showed signs of overfitting. This reinforces the need for additional adjustments to improve the model's generalizability.

In the future, techniques such as oversampling (SMOTE) or undersampling could be implemented to improve the representativeness of minority classes.

In summary, this project has demonstrated how machine learning techniques can be used to automate and optimize complex decision-making processes. The results achieved are promising and lay a solid foundation for future developments.

A Annexes A

Figure 1: Dendrogram of Missing Values

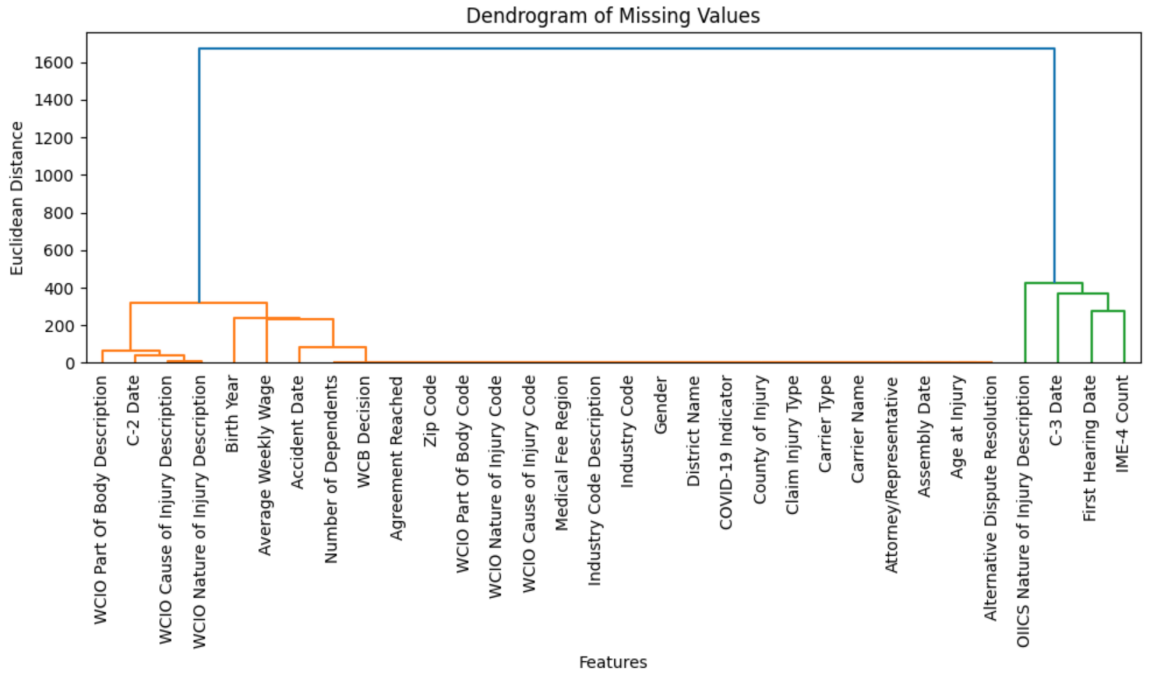

Figure 2: Heatmap of Missing values

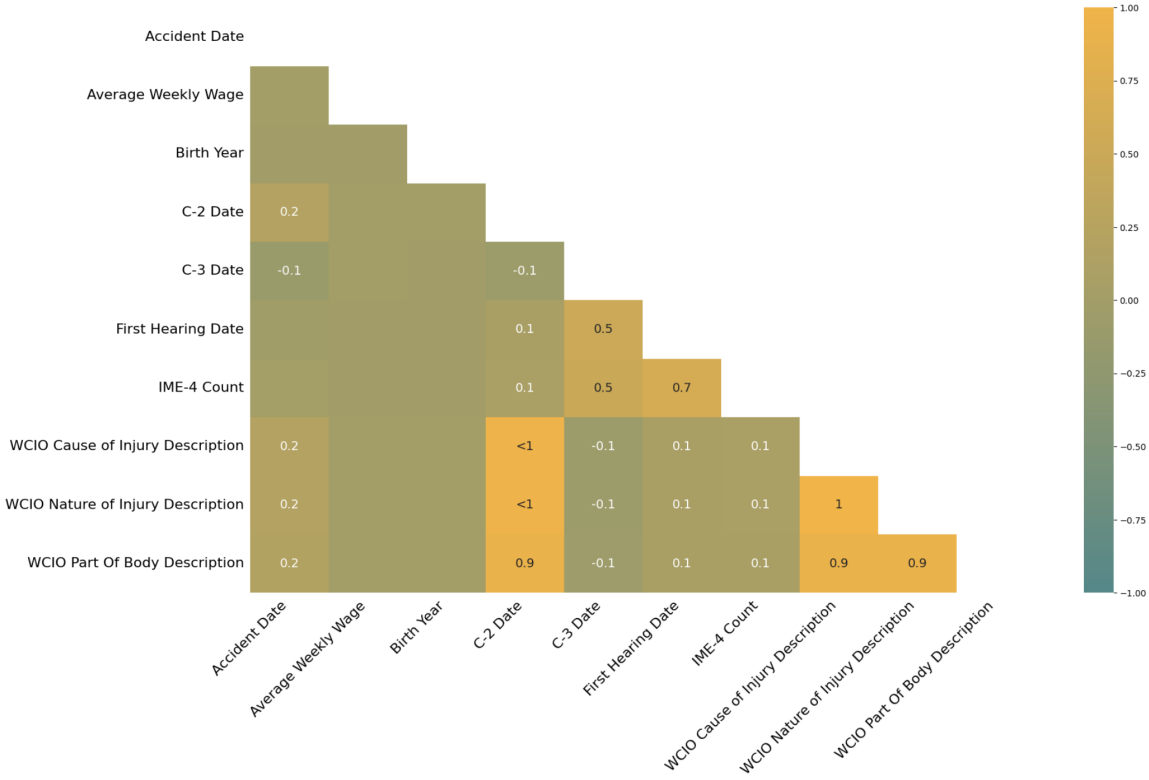

Figure 3: Frequency of Claim Injury Type

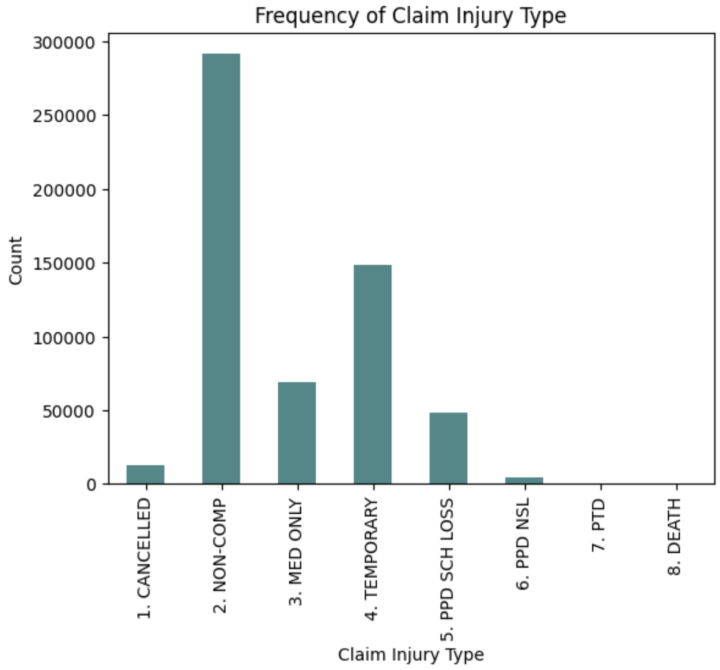

Figure 4: Distribution of Age at Injury

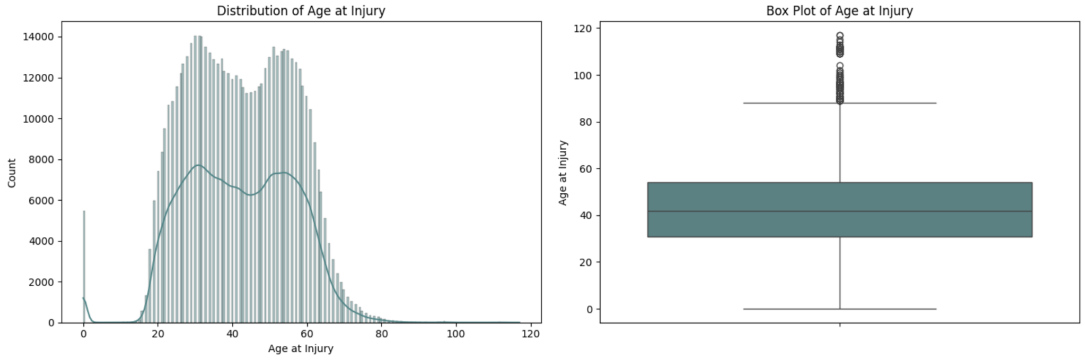

Figure 5: Distribution of Average Weekly Wage

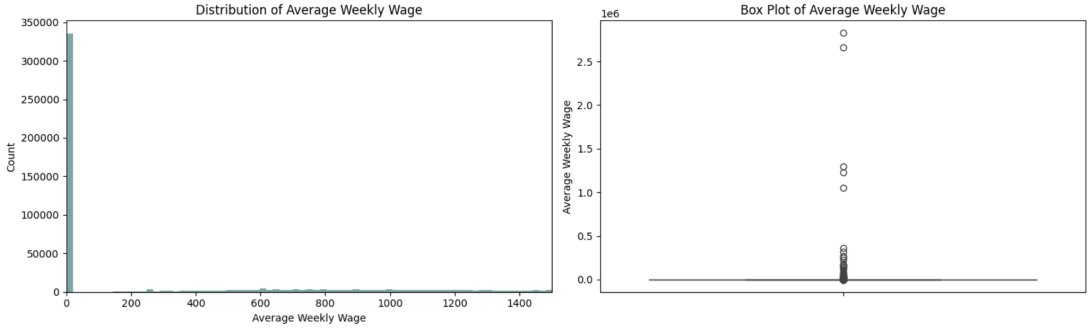

Figure 6: Distribution of Birth Year

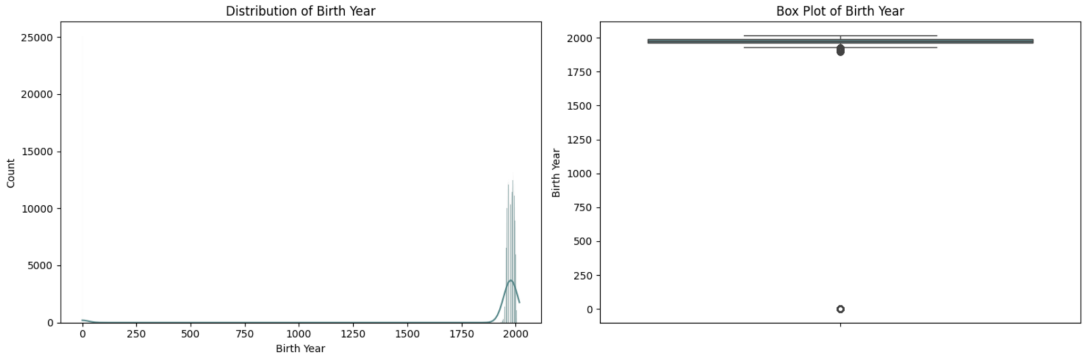

Figure 7: Distribution of IME-4 Count

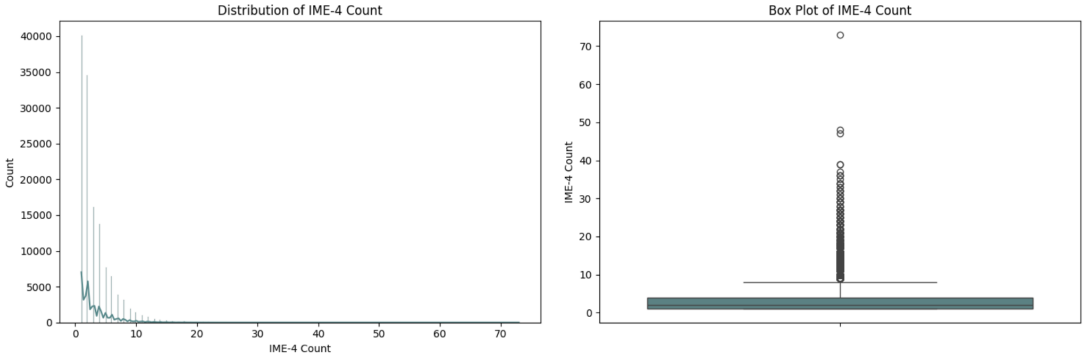

Figure 8: Distribution of Number of Dependencies

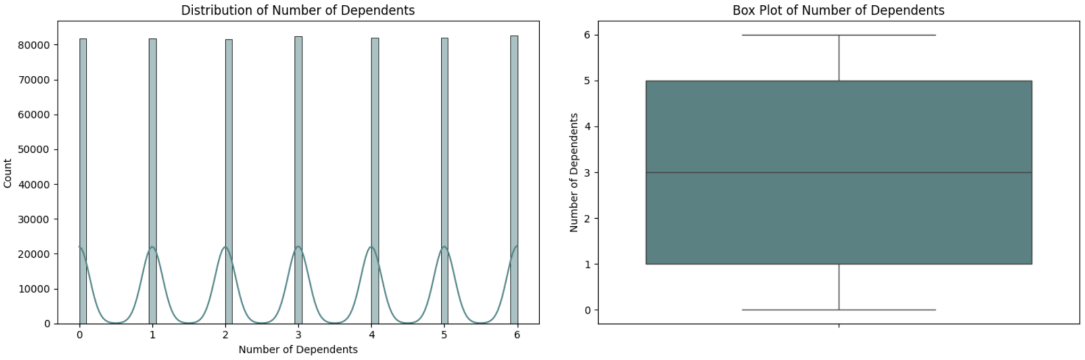

Figure 9: Key Features Relationships

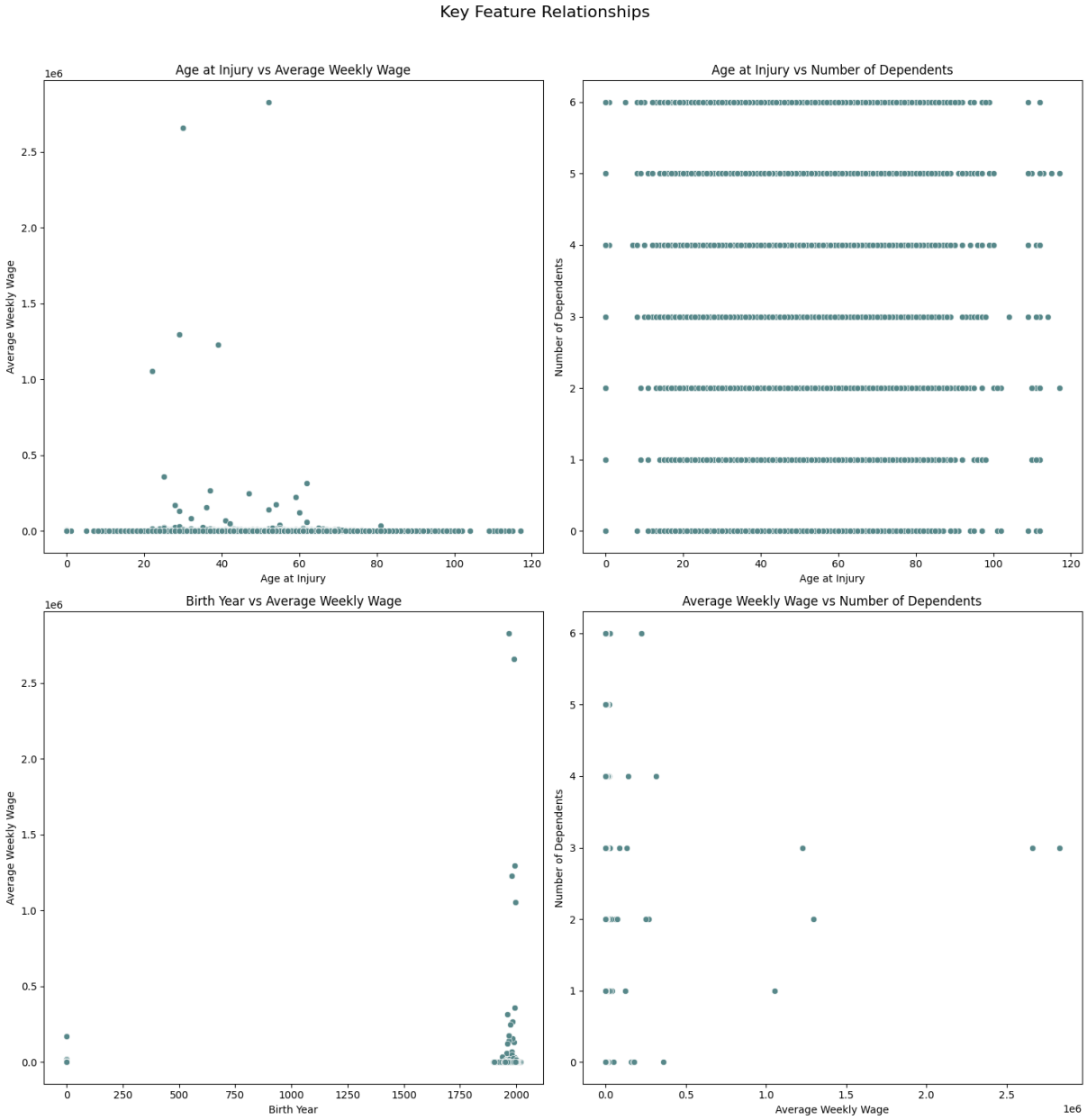

Figure 10: Frequency of Alternative Dispute Resolution

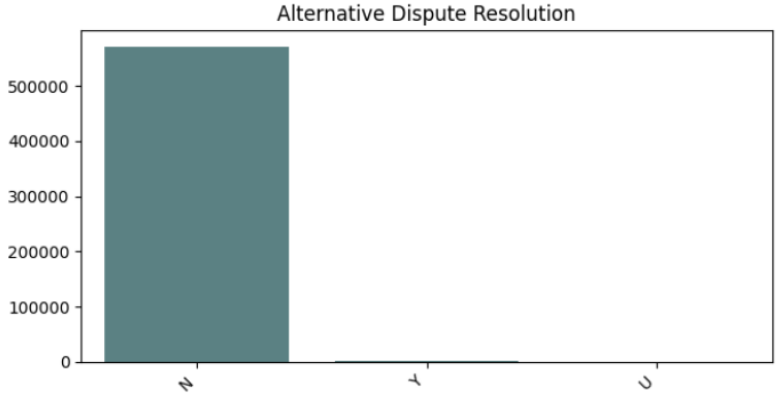

Figure 11: Frequency of Carrier Name

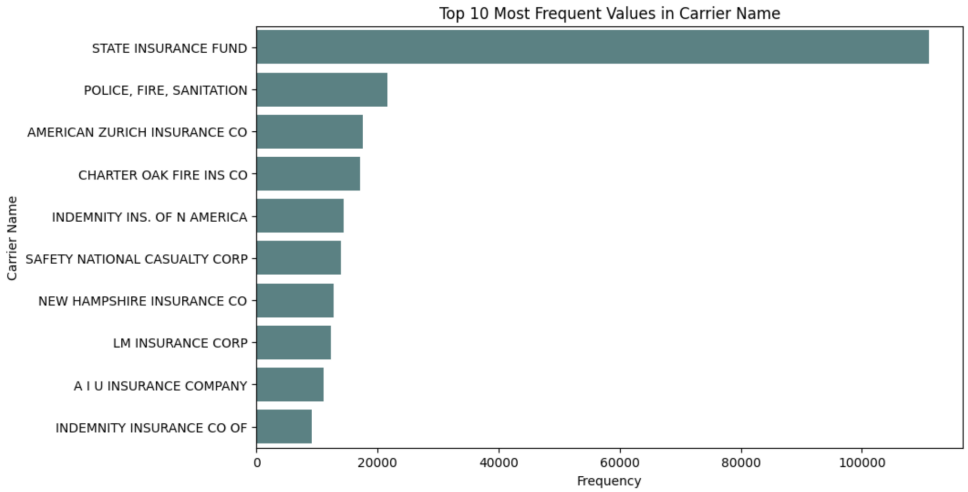

Figure 12: Frequency of Attoney/Representative

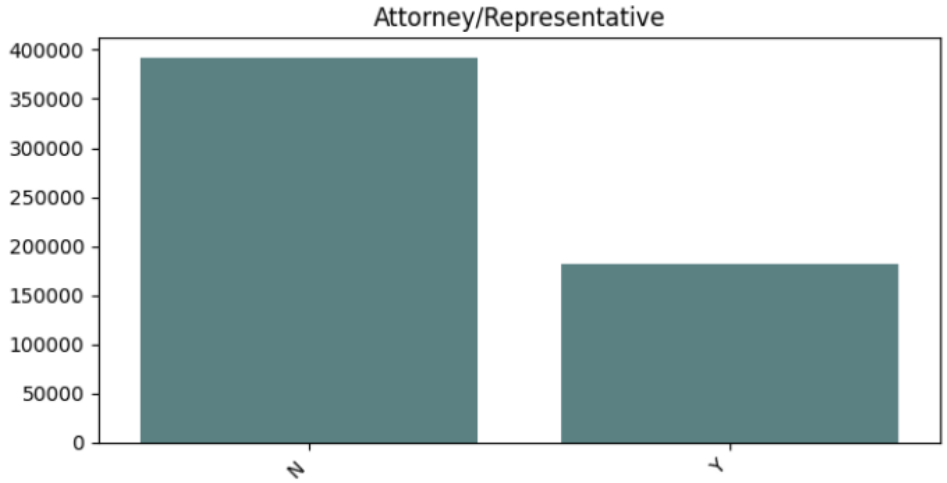

Figure 13: Frequency of Gender

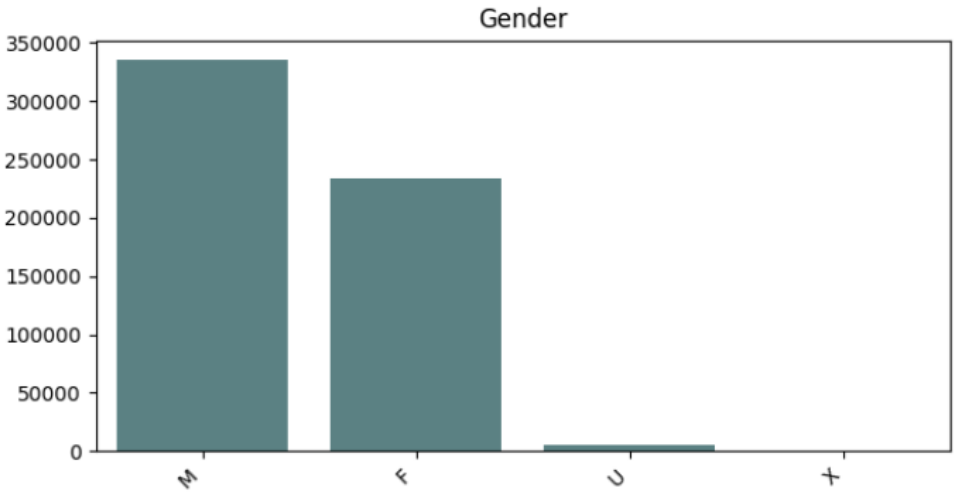

Figure 14: Number of cases by country

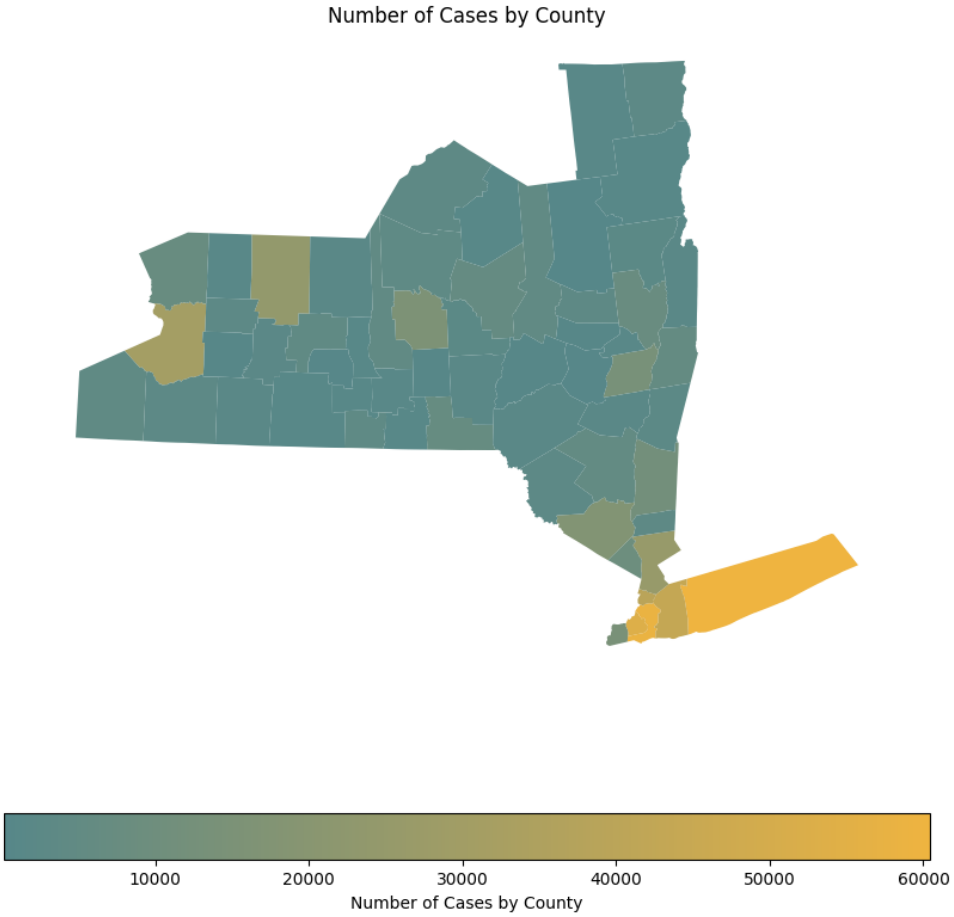

Figure 15: Claims count by year and month

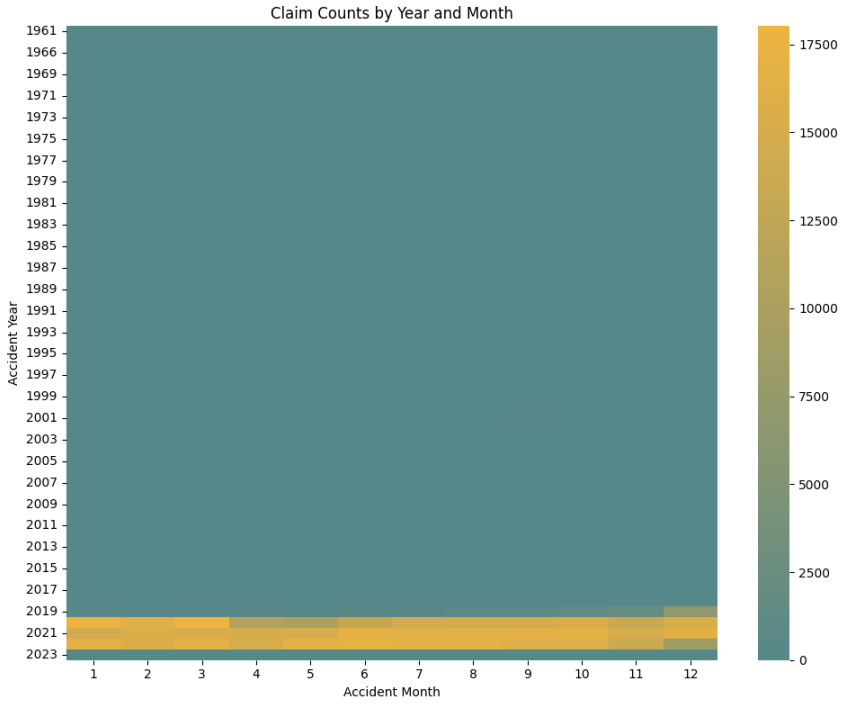

Figure 16: New Features

New Feature	how were created	description	why they were created
Kown_{Accident Date' + 'Assembly Date' + 'C-2 Date' + 'C-3 Date' + 'First Hearing Date' + 'Age at Injury' + 'Birth Year'}	Extracting data components	Create columns for year, month, day, and day of the week	Due to the high amount of missing values in these columns (sometimes because the event did not happen yet), we decided to create some variables that inform us if the date exists or not
Kown_{'Days_to_First_Hearing' + 'Days_to_C2' + 'Days_to_C3'}	Calculate the time difference between relevant dates	new columns to capture the elapsed time between key dates	can help understand delays or the timeline of events
'Accident_Season'	assigns a season based on the exact date	column that indicates the season in which the accident happened.	In order to know in which season the accident happen
'Weekend_Accident'	Add holidays for the current year to the list	Create columns that indicate if the accident happened on a weekend or a holiday	To know if the accidents happend on a weekend or a holiday
'Risk_Level'	create a dictionary with the correspondent risk	Create a column that indicates how dangerous the industry of the worker is	To see how danger is working in each industry
'Relative_Wage'	Calculate whether the injured worker's wage is above or below the median wage for the dataset	Relative Wage Compared to Median Wage	it's potentially reflecting job type or socioeconomic factors
'Financial Impact Category'	calculated as the Average Weekly Wage divided by the adjusted number of dependents	inancial impact per dependent based on the weekly wage	categorizes this financial burden into predefined groups for easier analysis
'Age_Group'	converts the categorized values into numeric codes	Group ages into categories like "young" or "senior"	capture different risk profiles

Figure 17: Transforming Features

Trasform Existing Feature	how were created	description	why they were created
'Carrier Type'	join all of the special funds in the unkown special funds	Type of primary insurance provider responsible for providing workers' compensation coverage	there are three values that are similar which represent different types of special funds
'Gender'	Replace 'U' for 'Unknown' and 'X' for 'Unknown'	The reported gender of the injured worker	making it easier to analyze and interpret

Figure 18: Distributions of numerical features after modifications

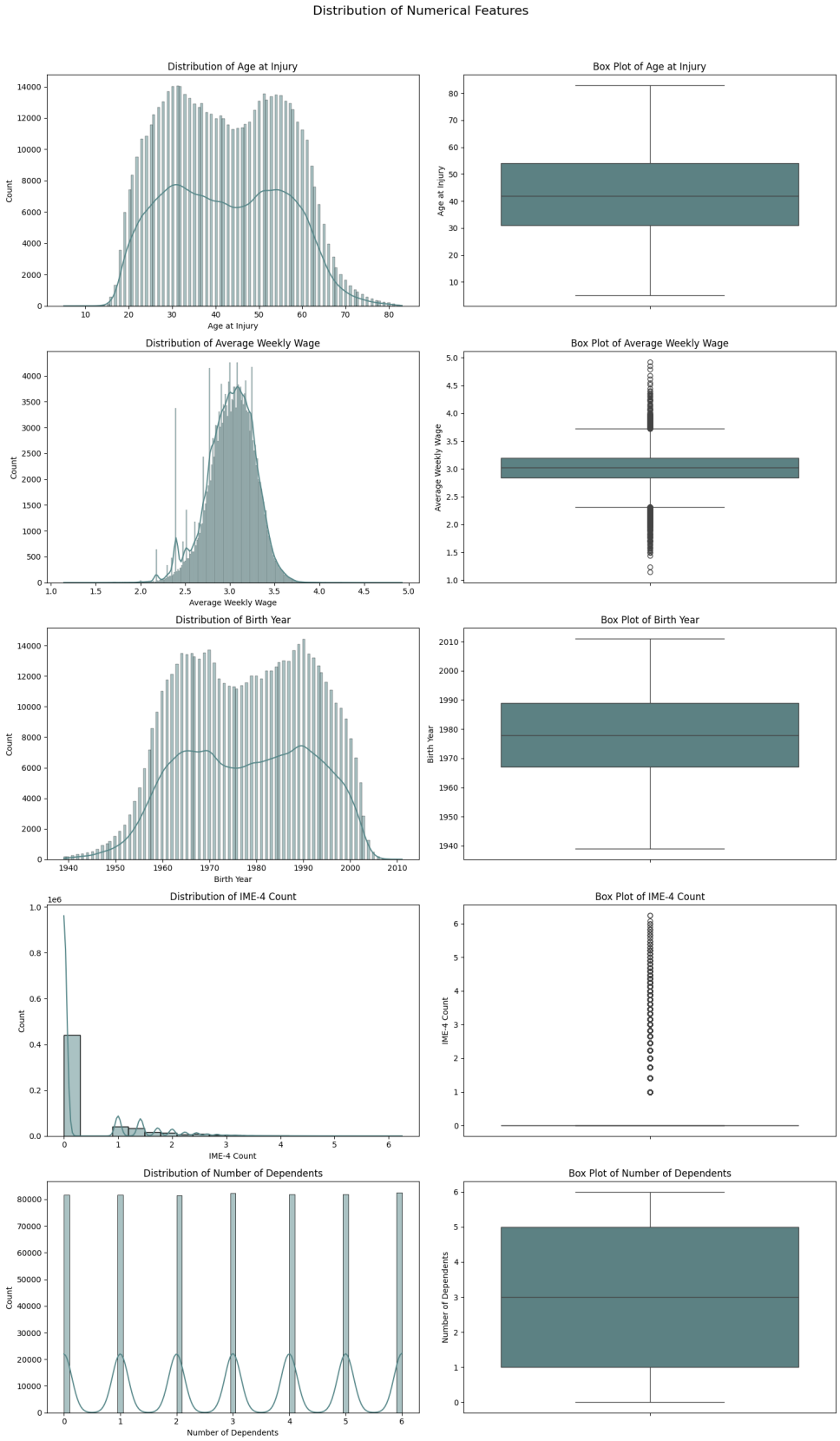

Figure 19: Correlation Heatmap of Numerical Features after modifications

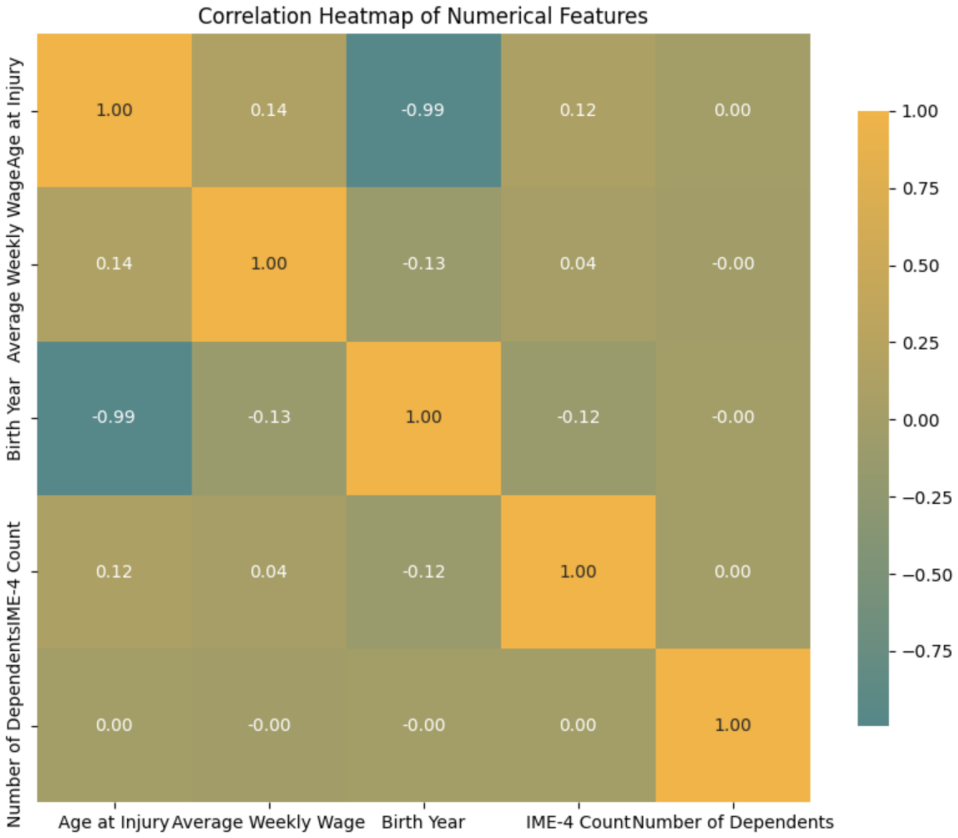

Figure 20: Multivariate Exploration to numerical features in order to 'Claim Injury Type'

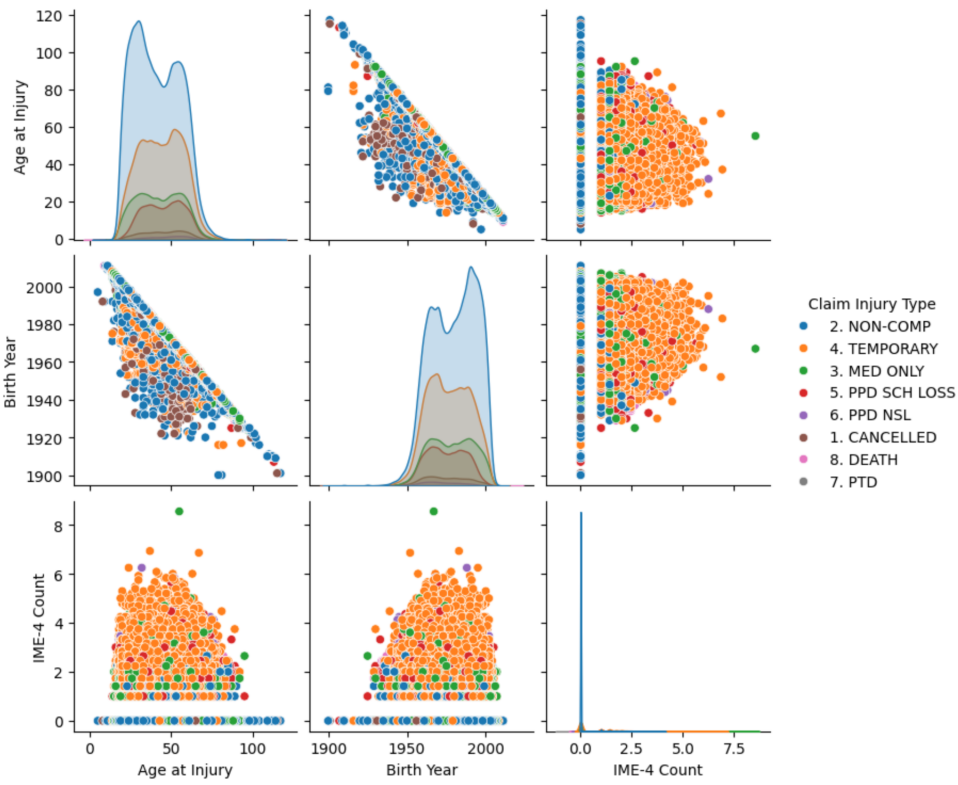

Figure 21: Multivariate Exploration to categorical features in order to 'Claim Injury Type'

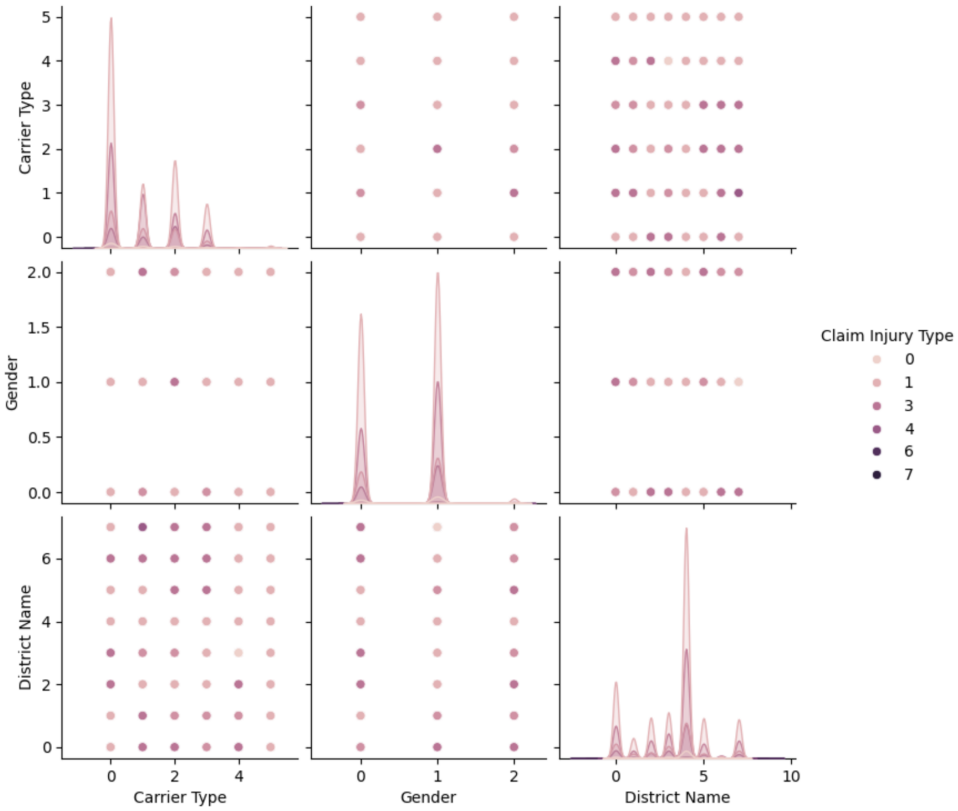

Figure 22: Scores of old models

Model	Average Train Macro F1 Score	Average Validation Macro F1-Score
Neural Networks with Keras	1.0	0.35
Decision Trees	1.00	0.35
Logistic Regression	0.27	0.27
Instance based	0.35	0.32
Random Forest	0.39	0.35
Neural Networks	0.54	0.38

Figure 23: Feature Selection on numerical features

Predictor	Spearman	Feature Importance XGB	Feature Importance LGB	Feature Importance DT	Lasso	RFE Logistic Regression	What to do? (One possible way to "solve")
Age at Injury	Keep	Drop	Keep	Keep	Drop	Keep	Try with and without
Average Weekly Wage	Keep	Keep	Keep	Keep	Keep	Keep	Include in the model
Birth Year	Drop	Drop	Keep	Drop	Drop	Keep	Discard
IME-4 Count	Keep	Keep	Keep	Keep	Keep	Keep	Include in the model
Number of Dependents	Keep	Drop	Keep	Drop	Drop	Drop	Discard
Days_to_C2	Keep	Drop	Keep	Drop	Keep	Keep	Try with and without
Days_to_C3	Keep	Drop	Keep	Drop	Keep	Keep	Try with and without
Days_to_First_Hearing	Keep	Drop	Keep	Keep	Keep	Keep	Include in the model
C-2 Date_Day	Keep	Drop	Keep	Keep	Drop	Keep	Try with and without
C-2 Date_Month	Keep	Drop	Keep	Drop	Drop	Drop	Discard
C-2 Date_Year	Keep	Drop	Keep	Keep	Keep	Keep	Include in the model
C-2 Date_DayOfWeek	Keep	Drop	Keep	Drop	Drop	Keep	Try with and without
C-3 Date_Day	Keep	Drop	Keep	Keep	Drop	Keep	Try with and without
C-3 Date_Month	Keep	Drop	Keep	Drop	Drop	Keep	Try with and without
C-3 Date_Year	Drop	Drop	Keep	Drop	Keep	Keep	Try with and without
First Hearing Date_Day	Keep	Drop	Keep	Drop	Drop	Keep	Try with and without
First Hearing Date_Month	Keep	Drop	Keep	Drop	Drop	Keep	Try with and without
First Hearing Date_Year	Drop	Drop	Keep	Drop	Keep	Keep	Try with and without
First Hearing Date_DayOfWeek	Keep	Drop	Keep	Drop	Drop	Keep	Try with and without
Accident Date_Day	Keep	Drop	Keep	Keep	Drop	Keep	Discard
Accident Date_Month	Keep	Drop	Keep	Drop	Keep	Keep	Try with and without
Accident Date_Year	Keep	Drop	Keep	Drop	Keep	Keep	Include in the model
Assembly Date_Day	Keep	Drop	Keep	Drop	Drop	Drop	Discard
Assembly Date_Month	Drop	Drop	Keep	Drop	Drop	Keep	Discard
Assembly Date_Year	Keep	Drop	Keep	Drop	Drop	Keep	Include in the model
Assembly Date_DayOfWeek	Keep	Drop	Keep	Drop	Drop	Drop	Discard

Figure 24: Feature Selection on categorical features

Predictor	Mutual Info Classif	Feature Importance XGB	Feature Importance LGB	Feature Importance DT	Lasso	RFE Logistic Regression	Chi-Squared Test	What to do? (One possible way to "solve")
Alternative Dispute Resolution_U	Drop	Drop	Drop	Drop	Drop	Drop	Drop	Discard
Alternative Dispute Resolution_Y	Drop	Keep	Drop	Drop	Keep	Keep	Keep	Try with and without
Attorney/Representative_Y	Keep	Keep	Keep	Keep	Keep	Keep	Keep	Include in the model
Carrier Type_2A. SIF	Keep	Drop	Keep	Drop	Keep	Keep	Keep	Try with and without
Carrier Type_3A. SELF PUBLIC	Keep	Drop	Keep	Drop	Drop	Keep	Keep	Try with and without
Carrier Type_4A. SELF PRIVATE	Drop	Drop	Keep	Drop	Keep	Keep	Keep	Try with and without
Carrier Type_5D. SPECIAL FUND - UNKNOWN	Drop	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Carrier Type_UNKNOWN	Drop	Keep	Drop	Drop	Drop	Keep	Keep	Discard
COVID-19 Indicator_Y	Drop	Keep	Keep	Drop	Keep	Keep	Keep	Try with and without
Enc County of Injury	Keep	Drop	Keep	Drop	Drop	Keep	Keep	Try with and without
Enc District Name	Keep	Drop	Keep	Drop	Keep	Keep	Keep	Try with and without
Enc Industry Code	Keep	Drop	Keep	Drop	Keep	Keep	Keep	Try with and without
Enc WCIO Cause of Injury Code	Keep	Drop	Keep	Drop	Drop	Keep	Keep	Try with and without
Enc WCIO Nature of Injury Code	Keep	Drop	Keep	Keep	Keep	Keep	Keep	Include in the model
Enc WCIO Part Of Body Code	Keep	Drop	Keep	Keep	Drop	Keep	Keep	Try with and without
Enc Zip Code	Drop	Drop	Keep	Drop	Drop	Drop	Keep	Discard
Financial Impact Category	Keep	Drop	Drop	Drop	Drop	Keep	Drop	Discard
Gender_M	Keep	Drop	Keep	Drop	Keep	Keep	Keep	Try with and without
Gender_Unknown	Drop	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Holiday_Accident	Drop	Drop	Drop	Drop	Drop	Drop	Drop	Discard
Known Accident Date	Keep	Drop	Drop	Drop	Drop	Keep	Drop	Discard
Known Assembly Date	Keep	Drop	Drop	Drop	Drop	Keep	Drop	Discard
Known C-2 Date	Keep	Keep	Drop	Drop	Drop	Keep	Keep	Try with and without
Known C-3 Date	Keep	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Known First Hearing Date	Keep	Keep	Drop	Keep	Drop	Keep	Keep	Try with and without
Known Age at Injury	Keep	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Known Birth Year	Keep	Drop	Drop	Drop	Drop	Keep	Drop	Discard
Medical Fee Region_II	Drop	Drop	Drop	Drop	Drop	Keep	Drop	Discard
Medical Fee Region_III	Drop	Drop	Drop	Drop	Keep	Keep	Drop	Discard
Medical Fee Region_IV	Keep	Drop	Drop	Drop	Keep	Keep	Drop	Discard
Medical Fee Region_UK	Drop	Drop	Drop	Drop	Keep	Keep	Drop	Discard
Relative_Wage	Keep	Keep	Keep	Keep	Keep	Keep	Keep	Include in the model
Risk_Level	Keep	Drop	Keep	Drop	Drop	Keep	Keep	Try with and without
Weekend_Accident	Drop	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Age_Group	Keep	Drop	Drop	Drop	Drop	Keep	Keep	Discard
Accident_Season_Sin	Drop	Drop	Keep	Drop	Drop	Keep	Keep	Discard
Accident_Season_Cos	Drop	Drop	Drop	Drop	Drop	Drop	Keep	Discard

Figure 25: Comparing models

Model	Best Parameters	Feature Selection	Average Train Macro F1 Score	Average Validation Macro F1-Score
CatBoostClassifier	`{"iterations": 1000, "learning_rate": 0.11, "depth": 6, "l2_leaf_reg": 5, bagging_temperature": 0.4}`	None	0.46	0.42
XGBoostClassifier	`{"n_estimators": 200, "learning_rate": 0.2, "max_depth": 7, "subsample": 0.9, "colsample_bytree": 0.9, "gamma": 0.3}`	None	0.65	0.42
Decision Trees	`{"min_samples_split": 10, "min_samples_leaf": 4, "max_depth": 20, "criterion": "entropy"}`	Essential Features	0.43	0.31
Naive Bayes	Default Parameters	Essential Features	0.25	0.24
StackEnsemble	CatBoost Config 1, XGBoost Config 1, Decision Trees, Default Parameters	Essential Features	0.31	0.30
VotingEnsemble	CatBoost Config 2, XGBoost Config 2, Decision Trees, Default Parameters	None	0.65	0.41
XGBoostClassifier With kfold	`{"n_estimators": 200, "learning_rate": 0.2, "max_depth": 7, "subsample": 0.9, "colsample_bytree": 0.9, "gamma": 0.3}`	None	0.75	0.45

A.1 CatBoost

To improve the results of our model, we decided to explore algorithms beyond the scope of the course. One of the models we explored was CatBoost, a gradient boosting algorithm designed to efficiently handle categorical data without the need for heavy preprocessing, such as one-hot encoding or label encoding following the article by Prokhorenkova et al. (2018)[2]. The CatBoost algorithm introduces two key techniques: ordered boosting and ordered boosting with categorical features [2].

First, CatBoost addresses the problem of prediction shift in traditional gradient boosting. Prediction shift occurs when a model overfits to the training data during the boosting process. CatBoost mitigates this issue by employing ordered boosting. Instead of using the entire training dataset for each gradient calculation, CatBoost introduces a random order to the data. When computing the gradient for a particular data point, only the data points preceding it in this random order are used. This randomization helps reduce the risk of overfitting, as it breaks the direct dependence on the entire training set and improves the model's generalization [2].

Second, CatBoost handles categorical features by converting them into numerical values known as Target Statistics. These values represent the expected target value for instances belonging to a specific category. To prevent target leakage, which occurs when the encoding of a category is influenced by its own target value, CatBoost calculates these target statistics in an ordered fashion. It only uses the target values of the preceding examples in a random order to compute the target statistics for any given example. This ordered approach eliminates the risk of data leakage and improves the model's ability to generalize to new data [2].

While both CatBoost and traditional Gradient Boosting involve building sequential trees to correct the errors of previous models, CatBoost introduces enhancements that make it more effective in specific scenarios. Unlike Gradient Boosting, which typically requires preprocessing for categorical features, CatBoost directly processes these features using Target Statistics [2]. Moreover, while both methods aim to reduce overfitting, CatBoost's ordered boosting technique helps mitigate this issue by randomizing the order of data, preventing the model from memorizing patterns in the training set. These innovations make CatBoost a more efficient and robust solution, especially when dealing with categorical data [2].

A.2 XGBoost

To enhance the performance of our predictive model, we explored XGBoost, a state-of-the-art gradient boosting algorithm that builds upon the principles of traditional gradient boosting with innovative enhancements. Widely used for classification and regression tasks, XGBoost has established itself as a high-performance tool due to its scalability, accuracy, and flexibility [1][3].

XGBoost operates by combining predictions from multiple decision trees in an ensemble framework. Each tree is sequentially trained to minimize the errors of the previous ones, reducing the overall prediction error. A key advantage of XGBoost lies in its ability to optimize a regularized objective function, which balances model complexity and performance. This approach incorporates both L1 (Lasso) and L2 (Ridge) regularization, helping to prevent overfitting while improving the model's ability to generalize to unseen data [1].

Additionally, XGBoost introduces several technical innovations that set it apart from traditional gradient boosting. First, it leverages advanced sparsity-aware techniques to handle missing or sparse data efficiently. By learning default directions for missing values

during tree construction, XGBoost avoids common pitfalls associated with incomplete datasets. Second, the algorithm uses a weighted quantile sketch to approximate splits in large datasets, ensuring computational efficiency even with billions of data points [1].

The computational efficiency of XGBoost is further enhanced by parallelization and distributed computing capabilities. By utilizing multithreading and optimizing cache access patterns, XGBoost achieves faster training times without sacrificing accuracy. Features such as column subsampling and shrinkage (a learning rate applied to each tree's contribution) further reduce the risk of overfitting and improve scalability[1][3].

In comparison to traditional gradient boosting, XGBoost delivers superior performance in terms of speed and predictive accuracy. Its ability to directly handle missing values and sparse data, coupled with advanced regularization and optimization techniques, makes it a robust choice for structured data tasks. XGBoost has consistently demonstrated state-of-the-art results across various domains, including fraud detection, risk assessment, and ranking problems[1][3].

References

[1] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD'16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[2] Liudmila Prokhorenkova et al. "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

[3] S. Ramraj et al. "Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets". In: *International Journal of Control Theory and Applications* 9.40 (2016), pp. 651–662. ISSN: 0974-5572. URL: https://www.researchgate.net/publication/318132203_Experimenting_XGBoost_Algorithm_for_Prediction_and_Classification_of_Different_Datasets.