Report for Advanced Analytics in Business

Ana Maria Giraldo Vargas, r0822450

David Badajkov, r0604517

Marcela Lopez Viveros, r0773141

Sonia Rocio Socadagui Casas, r0823960

Wai Chun Cheung, r0817438

May 3, 2021

Contents

Assignment 1

Feature engineering

The (training) dataset for Assignment 1 consists of 55,463 observations and 78 features. The number of features. It is easy to observe that there are a lot of missing values and categorical or date features in the dataset. In this section, we would discuss the strategies used in handling such problems.

Missing values

As shown in figure 1, there are 55 feature which contain missing values, and 24 out of them contain more than 80% of missing values. For those features with a high proportion of missing values (more than 50%), missing values are treated as an extra category. By treating missing values as an extra category called *unknown*, the information of the non-missing entries of those features can be retained and learnt by the model, whereas removal of those features may lead to a loss in information or pattern. For those features with a lower proportion of missing values (less than 50%), some imputation techniques can be applied to estimate their possible values. Another kind of missing values visualization using `missingno` is presented in figure 2.

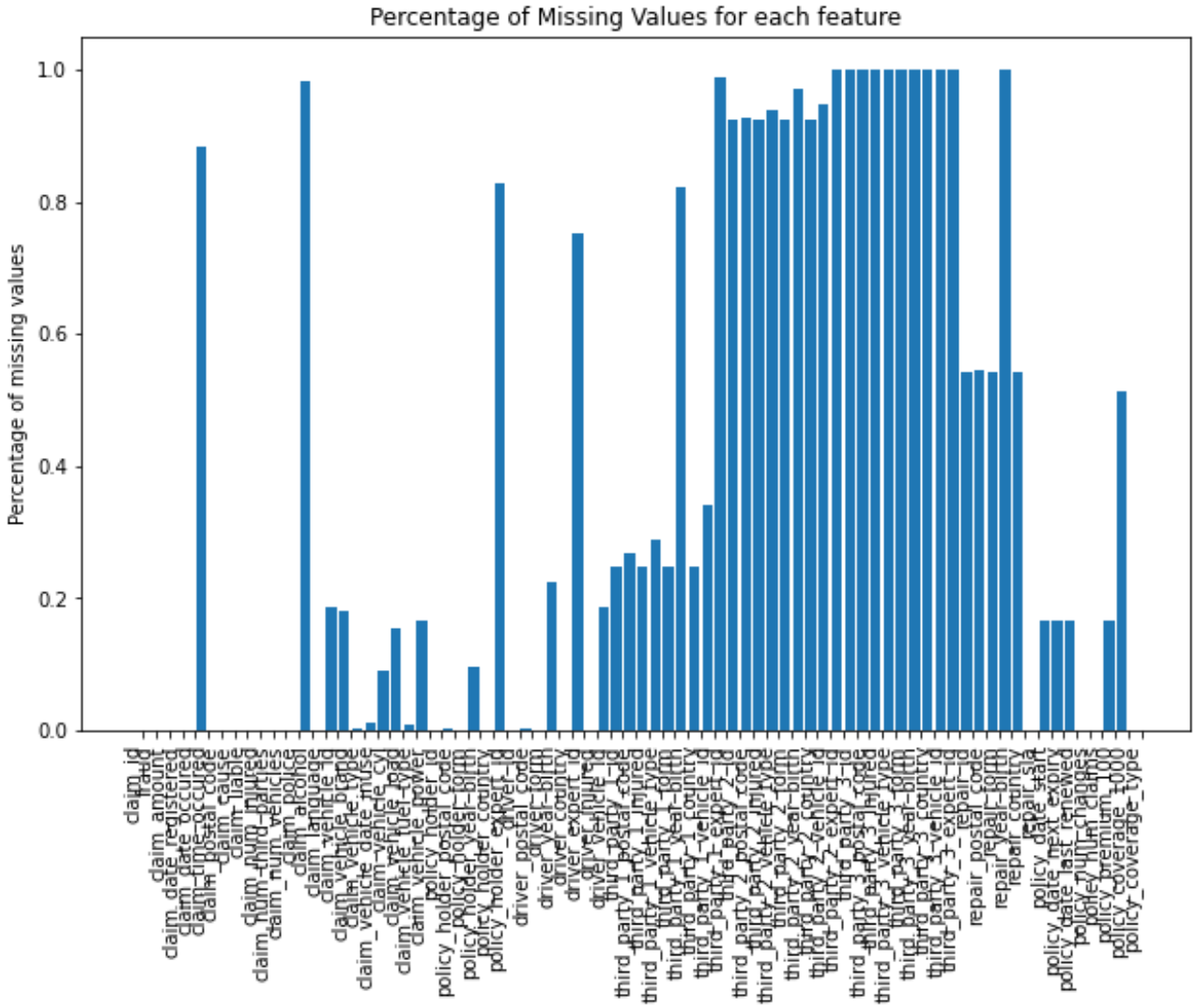


Figure 1: Proportions of missing values for each feature

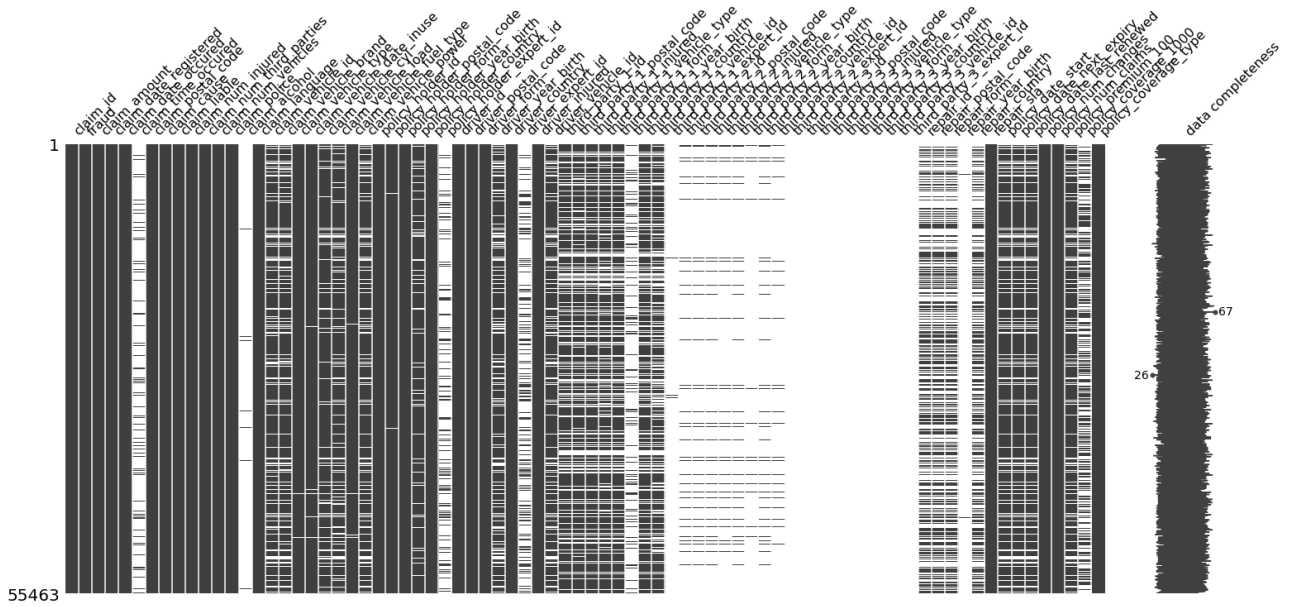


Figure 2: Visualization of missing values with `missingno` python library

Date features

Some features are available in form of date. In the dataset, the following features are in terms of date: `claim_date_registered`, `claim_date_occured`, `claim_vehicle_date_inuse`, `policy_date_start`, `policy_date_next_expiry` and `policy_date_last_renewed`. However, date itself is not suitable to serve as an input for some machine learning models. Hence, as summarized in table 1, we construct another set of features which are more interpretable and meaningful based on these date features.

Apart from the aforementioned features, features related to birth year are also considered as date features. They are `policy_holder_year_birth`, `driver_year_birth`, `third_party_1_year_birth`, `third_party_2_year_birth`, `third_party_3_year_birth` and `repair_year_birth` in the dataset. By subtracting them from the year of `claim_date_occured`, we obtain age-related features.

Constructed features	Descriptions
`days_before_registered`	The number of days between `claim_date_registered` and `claim_date_occured`.
`days_before_occured`	The number of days between `claim_date_occured` and `claim_vehicle_date_inuse`.
`policy_length`	The number of days between `policy_date_last_renewed` and `policy_date_start`.
`policy_claim_length`	The number of days between `policy_date_next_expiry` and `claim_date_occured`.
various age-related features	The number of years between the year of `claim_date_occured` and birth year.

Table 1: Featurization of date features

Data quality issue

During data cleaning process, we discovered some problems with regards to data quality. For example, the instance with claim id 62780 has an invalid value for the year of `claim_vehicle_date_inuse`. Such observations may be due to input error in manual data entering process. In this

case, we may manually correct the year from 2705 to 2005. However, they can be hard and time-consuming to observe in general. A separate data validation process should be carried out before data analysis.

```
[15]: # Input error in year of claim_vehicle_date_inuse
      data[data['claim_vehicle_date_inuse'] > 202012]
```

[15]:

n_vehicle_id	claim_vehicle_brand	claim_vehicle_type	claim_vehicle_date_inuse	claim_ve
J5ZWJmMjQ	CITROEN	car	270505.0	

Figure 3: Invalid value of `claim_vehicle_date_inuse` with claim id 62780

Data binning

Binning continuous features can help incorporating missing values and extreme values in a more natural way as they can be reformulated as categorical features. It is useful for those continuous features with a high proportion of missing values or highly right-skewed. For age-related features, they are binned by age groups with equal intervals. In our case, age-related features are encoded into the following categories: $(0, 20], (20, 40], (40, 60], (60, 80], (80, \infty]$ and *unknown*. Equal intervals of 20 years seem to be a reasonable choice by looking at their kernel density estimates of histograms. For driver age and policy holder age, the density looks like bimodal with a local minima around 40 years. Moreover, extreme ages, which are below 20 and above 80, can be handled properly under such binning.

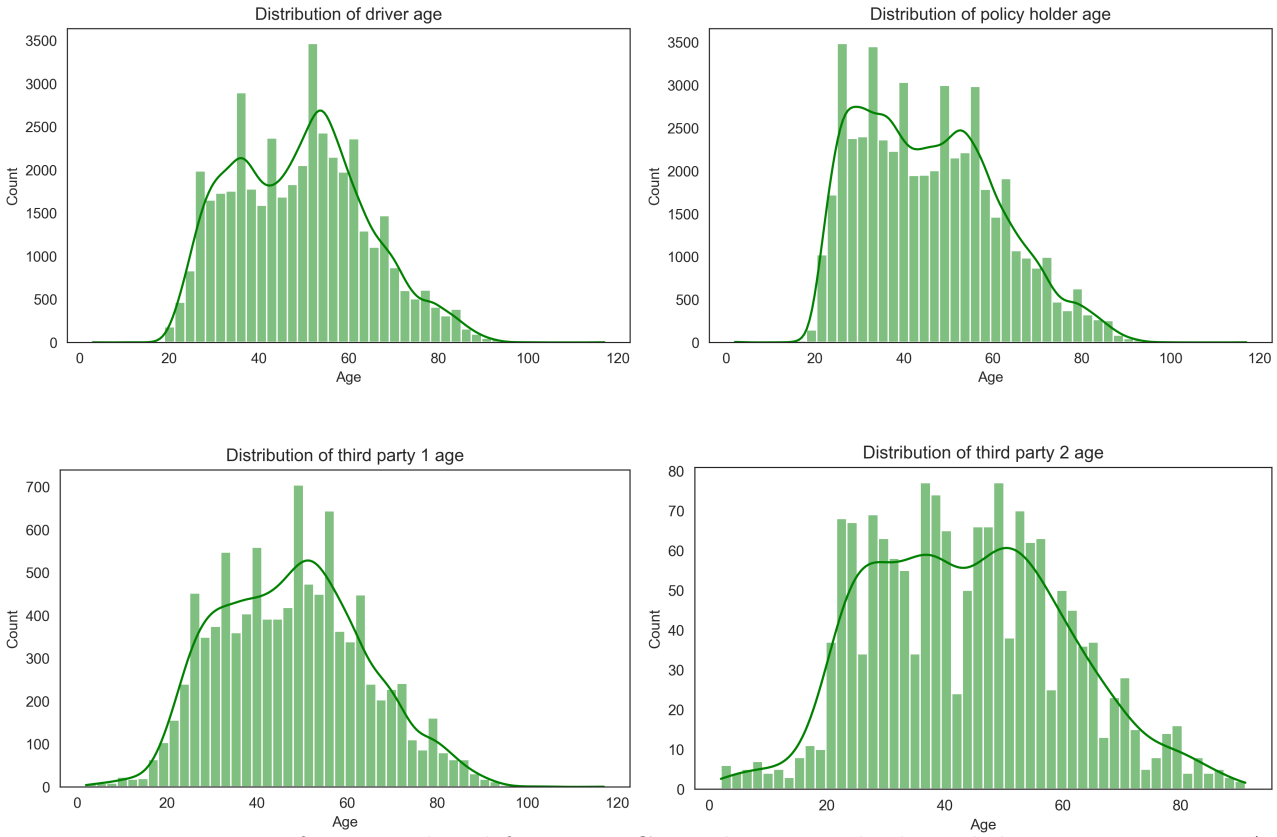


Figure 4: Histograms for age-related features. Green lines are the kernel density estimates. As third party 3 contains only few non-missing values, its histogram is not shown here.

An alternative to kernel density estimate is one-dimensional k-means algorithm. However, k-means algorithm partitions features based on mean and the specified number of clusters k. Due to skewness in age-related features, the bin for high age group may be too wide. As shown in figure 5, the fourth group has the range $[63, \infty)$ for `driver_age` and `policy_holder_age`. Therefore, we do not apply one dimensional k-means algorithm in this case.

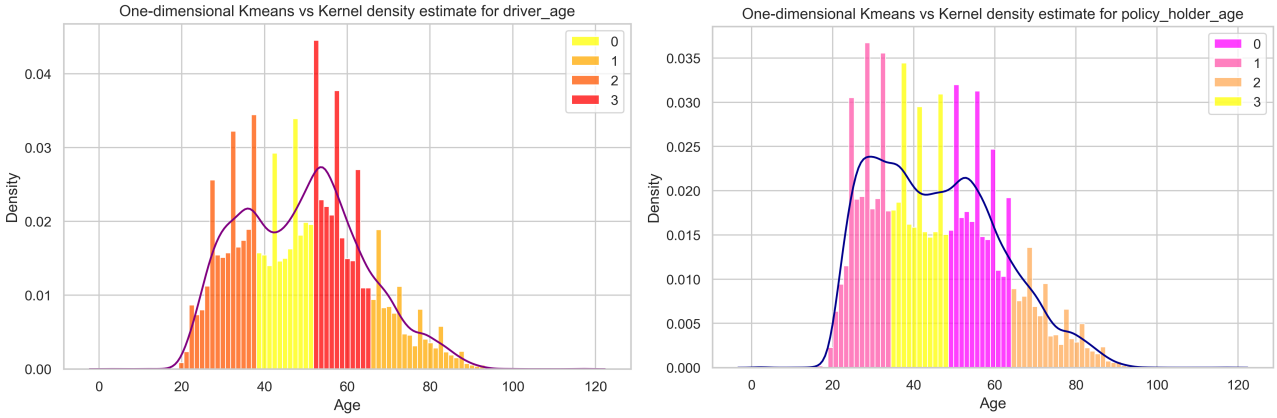


Figure 5: Compare data binning results of age-related features using One-dimensional Kmeans with 4 clusters and kernel density estimates.

For time features i.e. `claim_time_occured`, it contains about 88.4% of missing values and has a format of `HH:MM`. Therefore, `claim_time_occured` is binned in terms of hours, 00:00-01:00, 01:00-02:00,..., 23:00-24:00, and missing values are encoded as *unknown*. Under this categorization, non-missing time can be kept as a feature. Other binning schemes are possible but it is natural to bin time with `HH:MM` format into hours.

The dataset contains features related to the postal code, including `claim_postal_code`, `policy_holder_postal_code`, `driver_postal_code`, `third_party_1_postal_code`, `third_party_2_postal_code`, `third_party_3_postal_code` and `repair_postal_code`. However, these features have more than 1,000 unique postal code values. Without data binning, one hot encoding will create sparse data matrix and greatly increase the dimension of dataset. To avoid a huge search space, we keep the postal codes by their information values, calculated from their weights of evidence. For imbalanced data, we considered a lower threshold for information value, and the postal codes are preserved when their information values are greater than 0.01. Otherwise, they will be grouped under the same level, namely, *other* because they do not contribute much to detect fraudulent cases. A small threshold of 0.01 is used because there are many factor levels. The results are summarized in exploratory data analysis section. For `third_party_3_postal_code`, no non-missing fraudulent cases are found, and hence we simply encode it as a dummy variable with levels *known* and *unknown*.

Postal code-related features	Number of distinct values
`claim_postal_code`	1060
`policy_holder_postal_code`	1082
`driver_postal_code`	1077
`third_party_1_postal_code`	1088
`third_party_2_postal_code`	750
`third_party_3_postal_code`	27
`repair_postal_code`	754

Table 2: Total number of distinct postal codes for each related feature in the dataset

Apart from postal code, `policy_coverage_type` also have many factor levels. So, weight of evidence encoding is applied for the similar reason. For other techniques, the odds-based grouping also provides similar results whereas one-dimensional k-means is only applicable to continuous features. Grouping postal code by frequency seems a less plausible method for imbalanced dataset because the grouping may be biased to the majority class. Hence, weight of evidence encoding is a good method in our case.

There are some id-related features in the dataset, for instance, `claim_vehicle_id`, `policy_holder_expert_id`, `third_party_1_id`, `third_party_1_vehicle_id` and `third_party_1_expert_id`. As every id-related feature is anonymized, one hot encoding may also lead to a huge increase in dimension. Dummy encoding of id-related features, with known id labelled as 1 and missing id labelled as 0, gives meaningful interpretation to them.

Exploratory data analysis

In the exploratory analysis, most graphs are visualized using `seaborn` and `matplotlib` in python. Treemaps are visualized using `squarify`.

Class imbalance problem

Class imbalance problem makes the classification task harder. In the dataset, 55,155 cases are non-fraudulent, and 308 cases are fraudulent which accounts for 0.56% of the total sample, as summarized in figure 6. There are very few fraudulent cases in the dataset, and hence the dataset is imbalanced. It suggests some sampling techniques e.g. oversampling or smart sampling as well as metrics (other than accuracy) are needed in the training process so as to overcome the class imbalance problem.

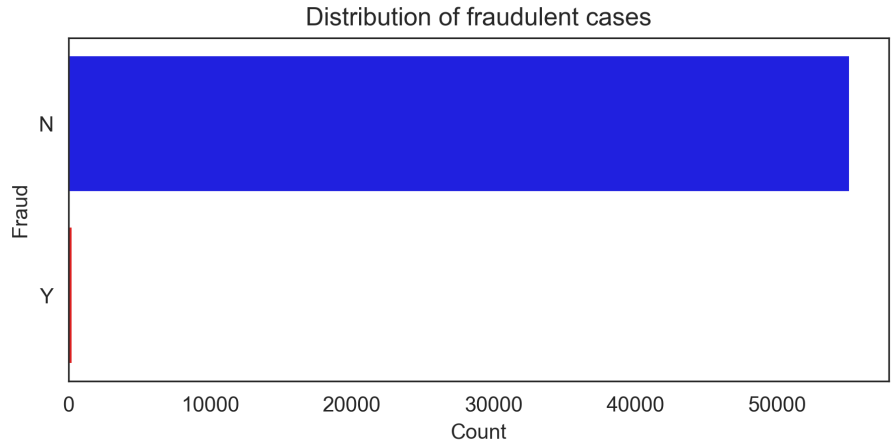


Figure 6: Barplot for `fraud`

Descriptive statistics

To develop a basic understanding of the training data, some features are covered in this section. The feature `claim_amount` is only provided in the training data which serves as a weighting variable in scoring because the goal of the classification task is to predict the probabilities of top 100 fraudulent cases in terms of their insurance claim amounts. In figure 7, we observe the distribution for `claim_amount` is highly right-skewed. Some observations have very high claiming amounts. In general, fraudulent cases tend to have higher claim amounts as their median and the upper quartile are much higher than those of non-fraudulent cases. Still, there are some non-fraudulent cases having high claim amounts.

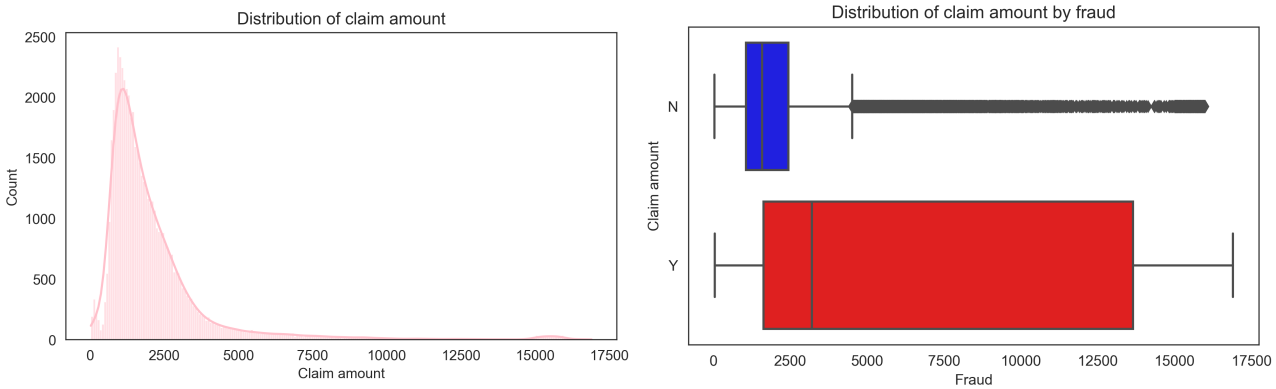


Figure 7: Distribution of `claim_amount`

A brief summary of categorical features is presented in figure 8 as barplots. For continuous features, their boxplots are presented in figure 9.

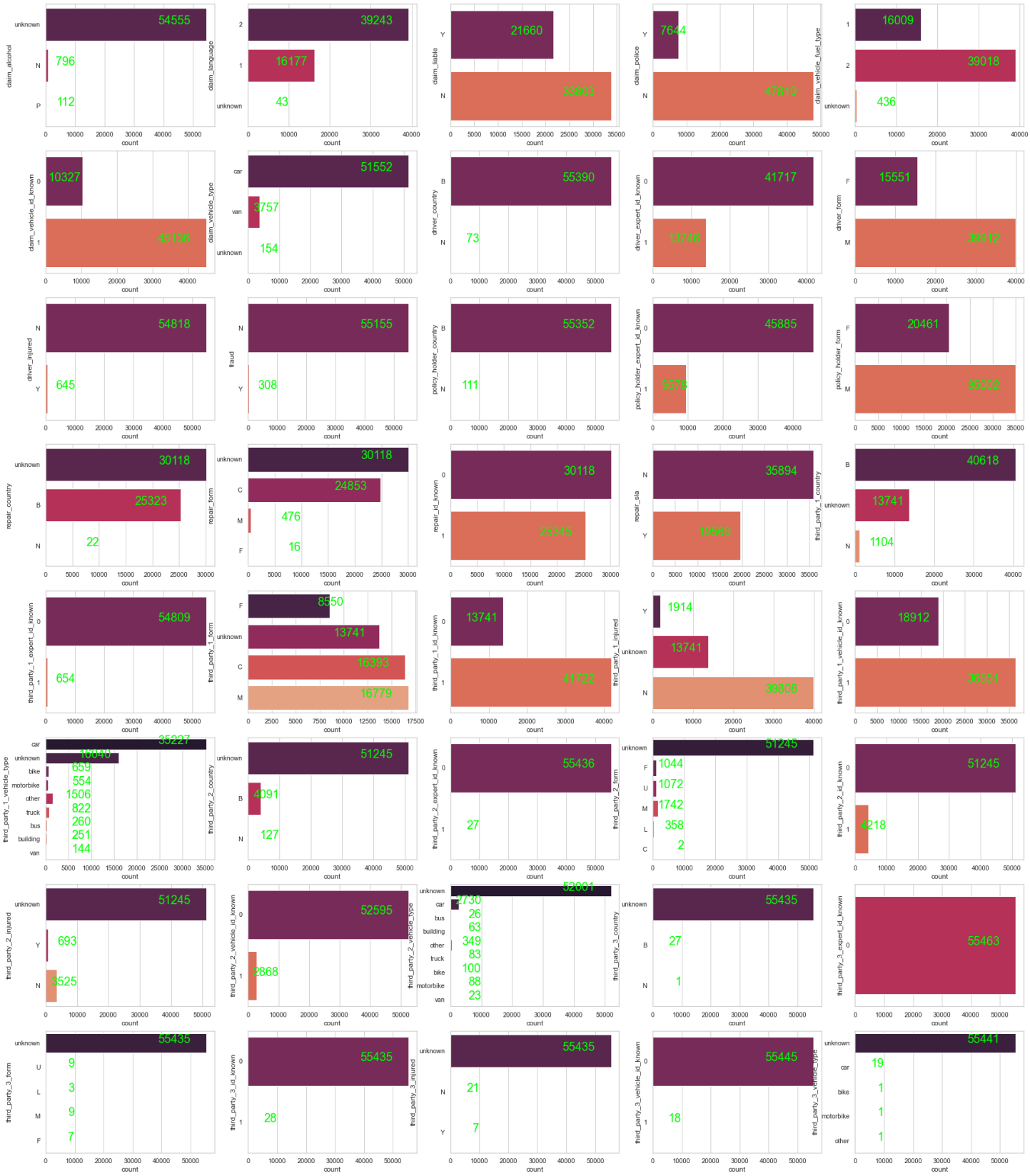


Figure 8: Summary of categorical features. Features with too many factor levels e.g. postal-code or vehicle brands are not included in this plot

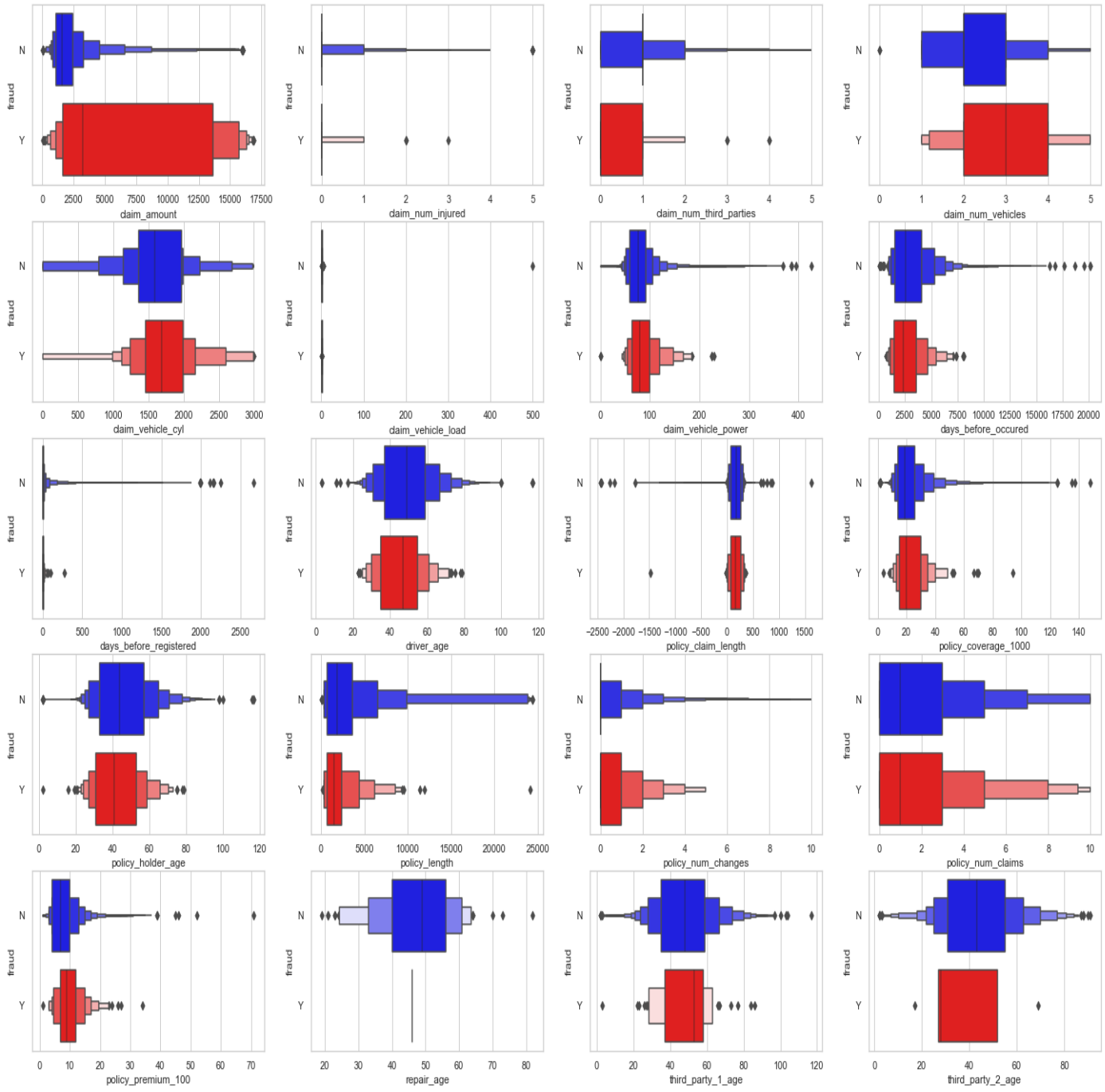


Figure 9: Summary of continuous features grouped by fraudulent status

The dataset contains some features related to the third parties, and `claim_num_third_party` indicates the number of third parties involved. There are about 69% of cases involved exactly one third party. About 24.7% of cases do not involve any third party, and only 6.3% of cases involve more than one third party. The average claim amounts and frequency of the number of third party conditioned on fraudulent status are summarized as heatmaps presented in figure 10. For fraudulent cases, about 58% of observations do not have third party, and 39% of them have one third party. Unlike non-fraudulent cases, about 69% of observations have one third party and only 25% of them have no third party. However, in terms of average claim amounts, fraudulent cases have high values for instances with more than one third party. It means features related to the second and the third third parties may be useful in classification.

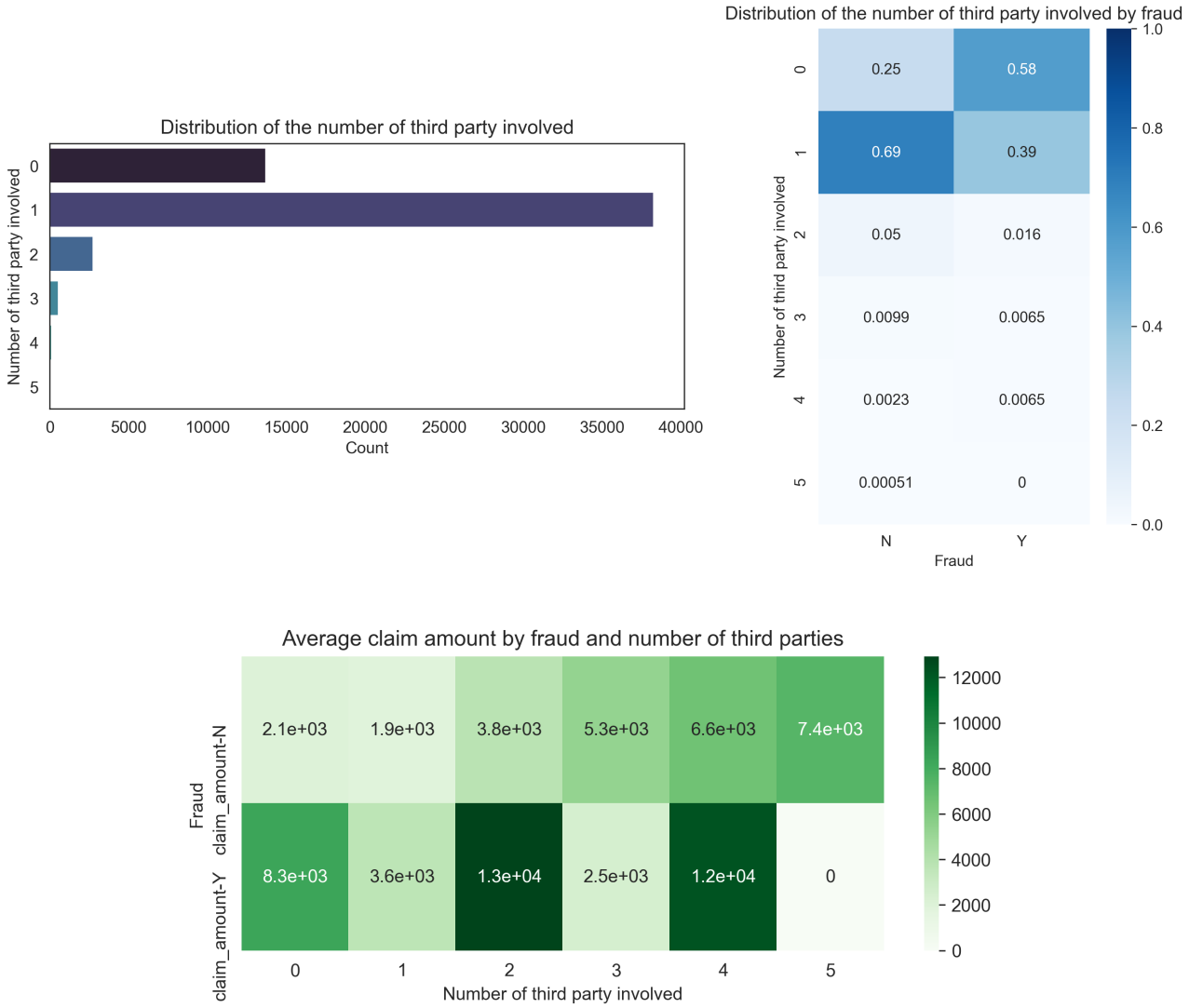


Figure 10: Distributions of `claim_num_third_parties` with barplot and heatmaps

To know whether the occurrence of fraudulent cases and their claim amounts vary across vehicle brands, we visualize `claim_vehicle_brand` using treemaps from `squarify` library in python. In figure 11, it is clear that vehicles from Porsche, Land Rover and Toyota have the highest average claiming amounts for fraudulent cases. The average claiming amounts are $15,861 for Porsche, $15,328 for Land Rover and $11,640 for Toyota. However, most of the fraudulent cases do not have vehicle brands recorded, which are labelled as *unknown* in figure 12. Apart from missing cases, BMW, Volkswagen and Citroen are found to be most frequent fraudulent cases.

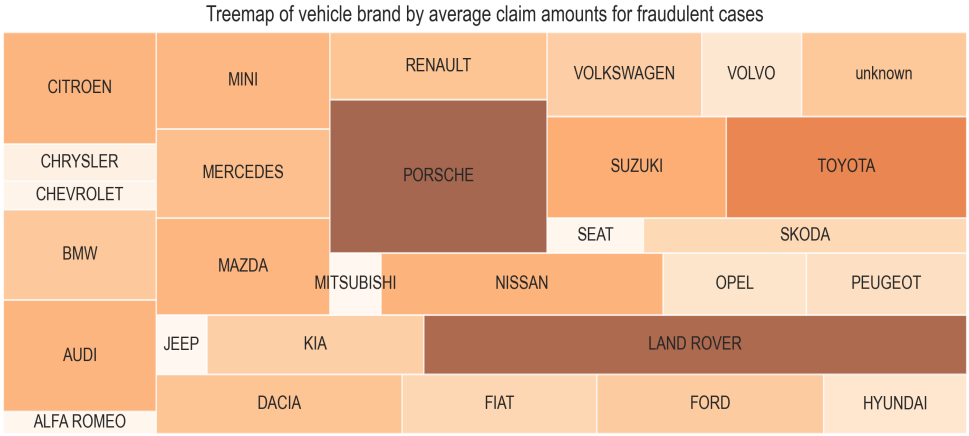


Figure 11: Average claim amounts of fraudulent cases for different vehicle brands

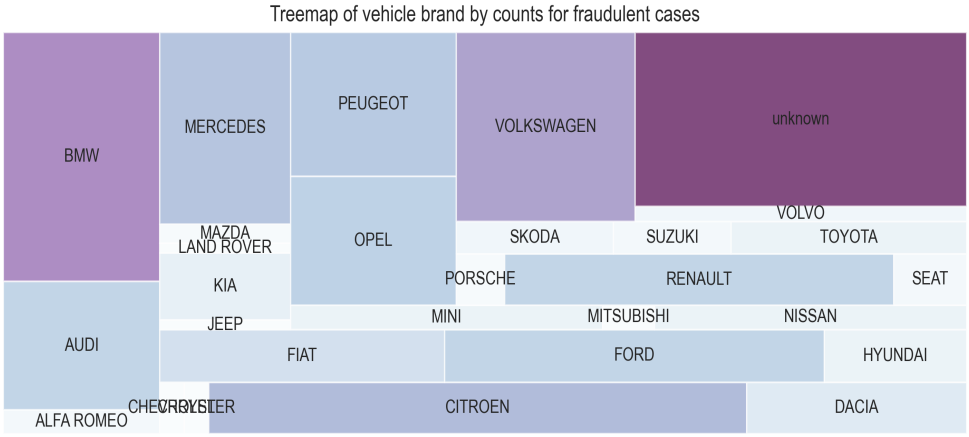


Figure 12: Counts of fraudulent cases for different vehicle brands

Among various claim causes, we found that *fire* and *theft* are associated with higher claim amounts. The most frequent causes for fraudulent cases are *traffic accident*, *theft* and *other*. In the training data, there are only few observations for *windows*, *weather* and *vandalism*. The violinplot in figure 13 summarizes the findings.

`claim_cause`	animal	fire	other	theft	traffic accident	vandalism	weather	windows
`fraud` = N	334	29	7379	513	43366	198	744	2592
`fraud` = Y	10	10	56	97	130	0	4	1

Table 3: Counts by `fraud` and `claim_cause`

As discussed in previous section, `claim_time_occured` is binned into 24 hours. Therefore, we can analyze the frequency of claim causes in 24 hours in an exploratory way. A heatmap on the frequency of claim causes across time for non-missing data is presented in figure 14. We observe that the claims due to *animal* usually occur at 06:00 - 07:00 and 22:00 - 23:00. The claims due to *fire* occur most frequently at the afternoon (from 16:00 to 18:00). There is no special time pattern for the claims due to *other*. The claims due to *theft* occur most frequently at 03:00-04:00, and the claims due to *traffic accident* occur at the working time period (07:00-21:00). The claims due to *vandalism* usually occur at 10:00-11:00 and 15:00-16:00. The claims due to *weather* usually occur at 16:00-18:00, and the claims due to *windows* usually occur at 12:00-13:00.

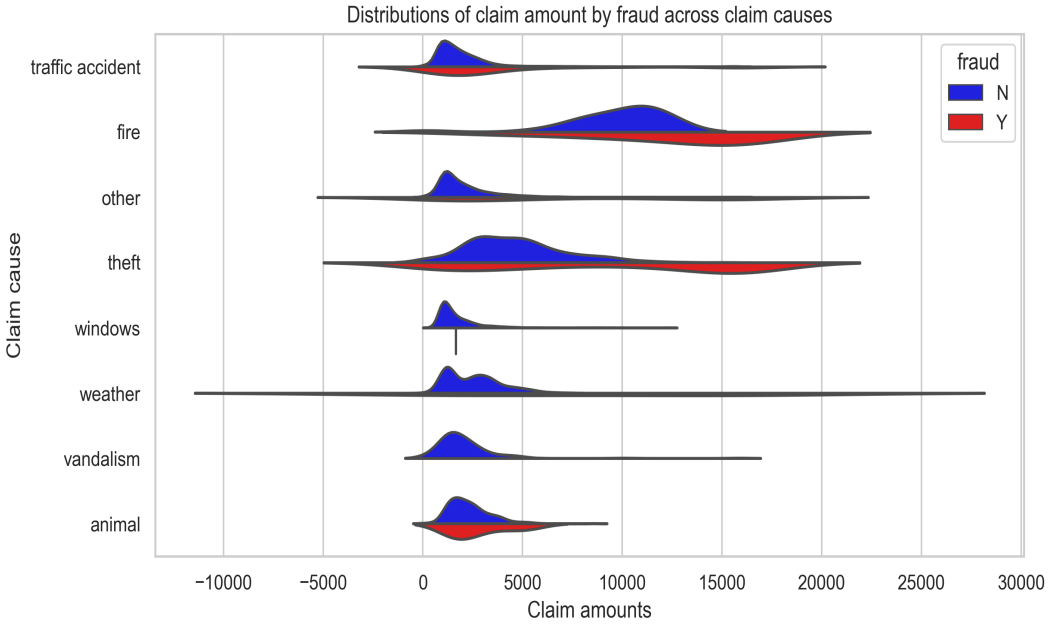


Figure 13: Distributions of claim amounts by fraudulent status and claim causes

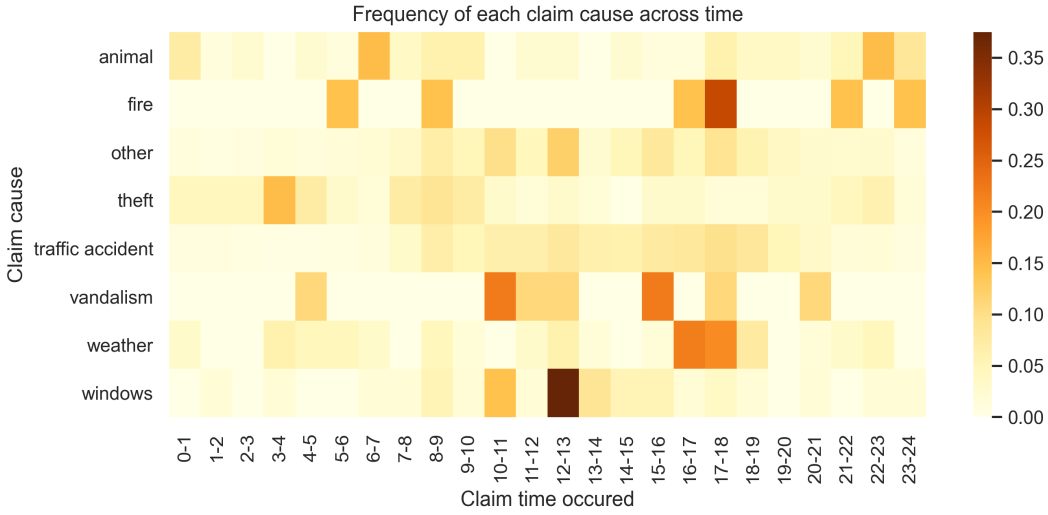
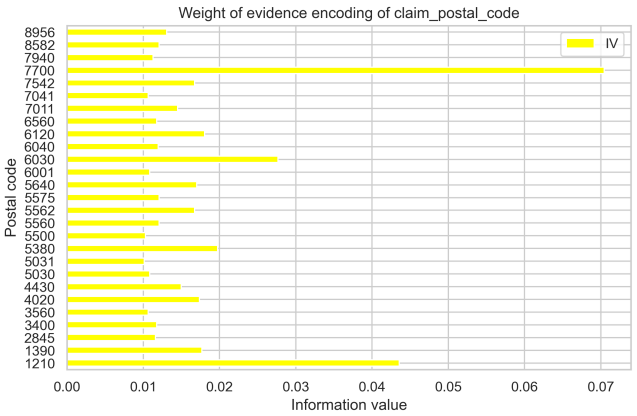
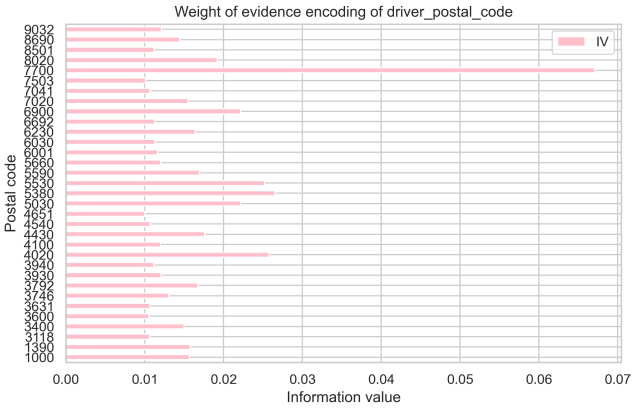


Figure 14: Frequency of claim causes across time, with missing data excluded

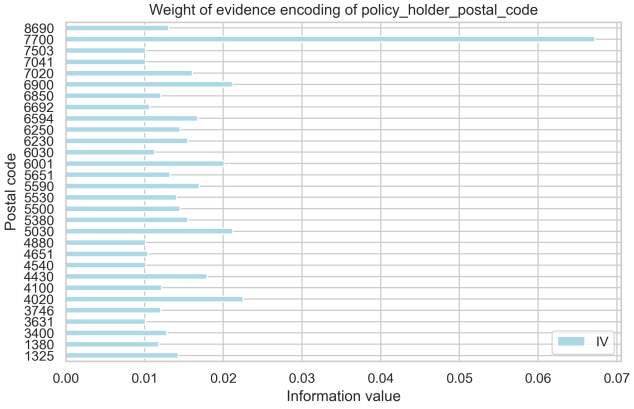
For postal code-related features, as discussed in previous section, postal codes are grouped in terms of their information values from the weight of evidence encoding. The corresponding information values are summarized in figure 15. For `claim_postal_code`, postal codes 7700, 1210 and 6030 have relatively high information values. For `driver_postal_code` and `policy_holder_postal_code`, postal code 7700 also has the highest information values. It seems that postal code 7700 has some evidence for fraudulent cases. For `repair_postal_code`, postal code 1731 has the highest information value. For `third_party_1_postal_code`, *unknown* has the highest information value because fraudulent cases mostly have missing values. For `third_party_2_postal_code`, postal code 7141 and 3510 have relatively high information values. For `third_party_3_postal_code`, there are no fraudulent cases with non-missing values. Therefore, it is not presented here.



(a) `claim_postal_code`

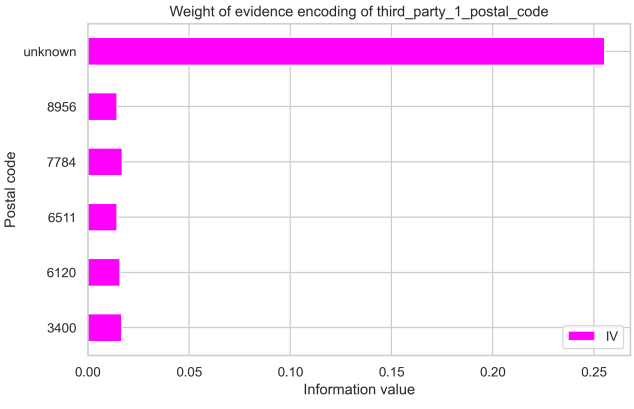


(b) `driver_postal_code`

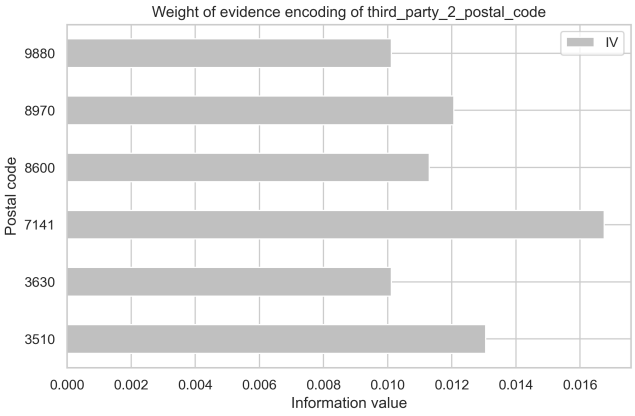


(c) `policy_holder_postal_code`

(d) `repair_postal_code`



(e) `third_party_1_postal_code`



(f) `third_party_2_postal_code`

Figure 15: Information values for postal code-related features

There are 73 different policy coverage types in total. After weight of evidence encoding, coverage types #000110000, #111110001 and #000110100 show relatively high information values as shown in figure 16. These coverage types may be informative in identifying fraudulent cases.

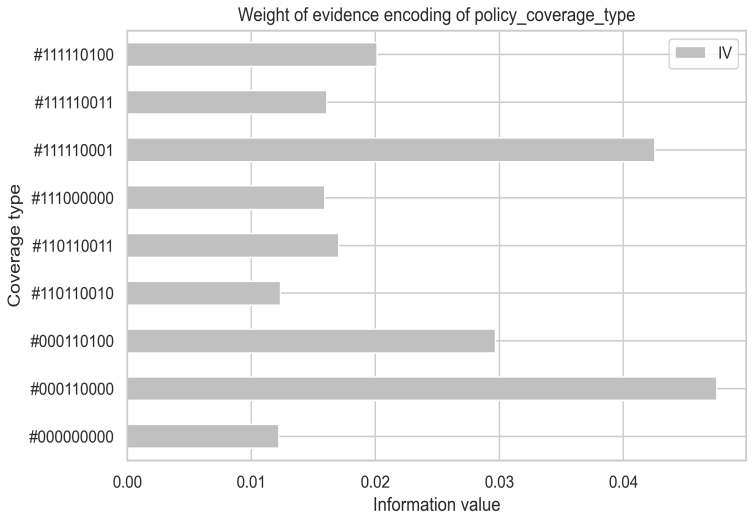


Figure 16: Information values for policy coverage types

Correlation plot

High correlation among features may indicate the existence of some redundant features. Therefore, a correlation plot is presented in figure 17, and we observe that the correlation among `policy_coverage_1000` and `claim_vehicle_cyl` and `claim_vehicle_power` are strong. And the correlation between `driver_age` and `policy_holder_age` are also strong. There may be some common factors behind correlated features. Although the correlations are strong. Instead of dropping them, we can discretize the features to reduce the correlation and preserve information as they are not 100% correlated. As discussed in previous section, age-related features are discretized. Moreover, we discretize `policy_coverage_1000` into equal intervals because it has more than 50% of missing values and a strong correlation.

Dimension reduction

For exploratory analysis, we visualize the training data using dimension reduction techniques. Uniform manifold approximation and projection (Umap) is a dimension reduction technique using manifold learning, which efficiently embeds or represents the topology of higher dimension space with a lower dimension space through approximation. Of course, there are many other popular dimension reduction techniques available like t-distributed stochastic neighbor embedding (tSNE) and principal component analysis (PCA) and its variants e.g. kernel PCA. However, it is hard to incorporate class label using tSNE, whereas Umap supports supervised dimension reduction. Moreover, Umap tends to give a more stable results and preserves more global structure compared to tSNE.

For PCA which reduces dimensions in a linear fashion, the correlation plot in mentioned in the previous subsection shows most of the features do not share a high linear correlation. Thus, eigendecomposition on covariance matrix would not reduce much dimensions as expected. In this sense, Umap is an appealing tool for dimension reduction. In this section, we use `umap-learn` python library to perform Umap.

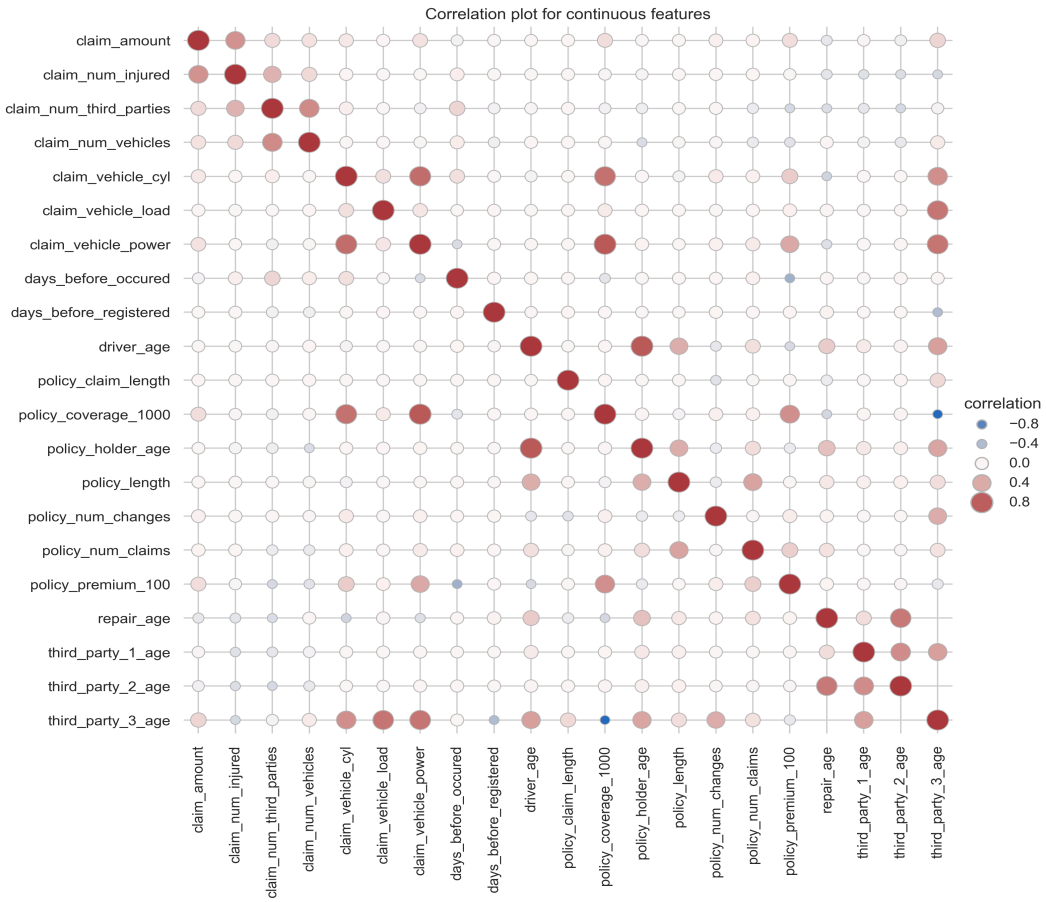


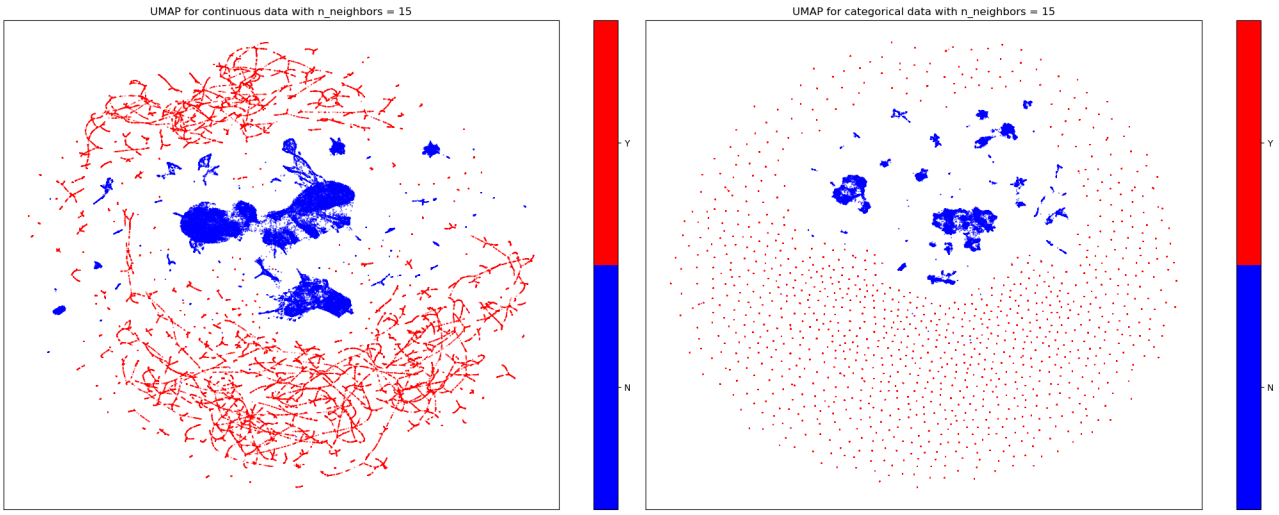
Figure 17: Correlation plot for continuous features

There are some hyperparameters in Umap, and the most important hyperparameters are `n_components`, `n_neighbors`, `min_dist` and `metric`. We notice that `n_components` represents the output dimension, usually between 2 and 3 for visualization purpose, while `n_neighbors` represents the number of neighbourhood instances used in manifold approximation. The minimum distance among points in the output dimension is controlled by `min_dist`. A larger `min_dist` would give a more disperse representation in the embedded space. The distance metric defined between two points in original dimensional space is controlled by `metric` for which the Euclidean distance is applied to continuous features and the Jaccard distance is applied to categorical features after encoding into dummy variables.

Outline of the steps in performing dimension reduction:

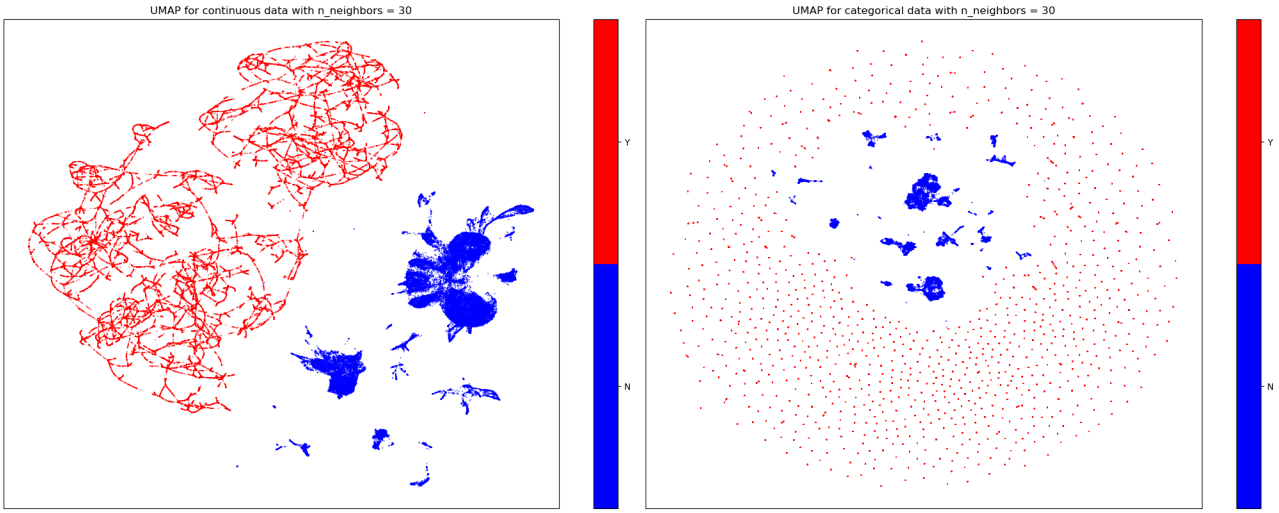
1. Apply featurization and imputation to the training data.
2. Standardize continuous features.
3. Oversampling with SMOTE (due to imbalanced class) from `imblearn`.
4. Apply Umap to continuous features with the Euclidean distance and to categorical features with the Hamming distance after encoding into dummy variables.
5. Train Umap with different `n_neighbors` and `min_dist`.

The results of Umap with difference `n_neighbors` are shown in figure 18 and 19. The embedded results for continuous features changes when `n_neighbors` increases from 15 to 30, and not much change in the results for categorical features.



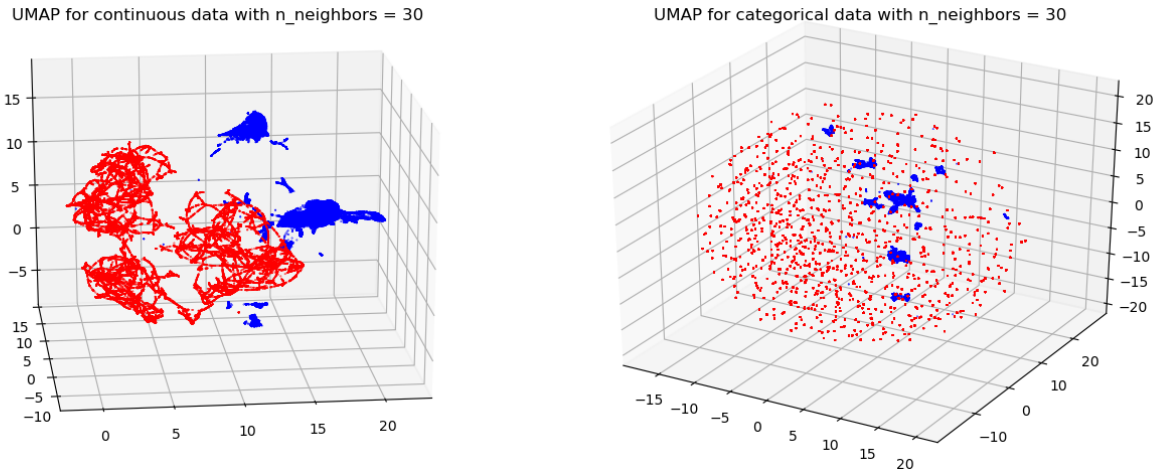
(a) Continuous features (b) Categorical features

Figure 18: Visualization using supervised Umap, `n_neighbors` = 15, `min_dist` = 0.1, `n_components` = 2



(a) Continuous features (b) Categorical features

Figure 19: Visualization using supervised Umap, `n_neighbors` = 30, `min_dist` = 0.1, `n_components` = 2

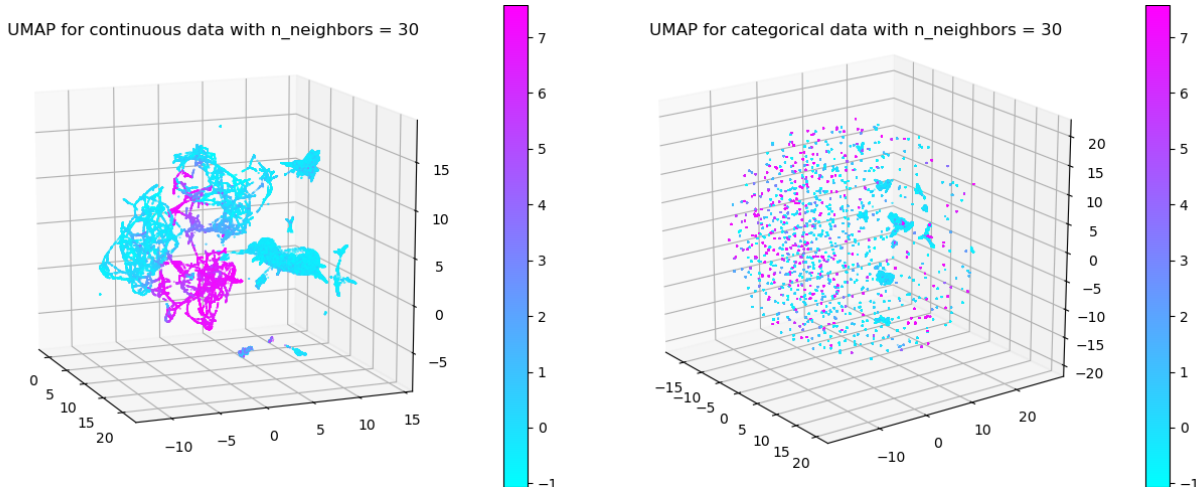


(a) Continuous features (b) Categorical features

Figure 20: Visualization using supervised Umap, `n_neighbors` = 30, `min_dist` = 0.1, `n_components` = 3

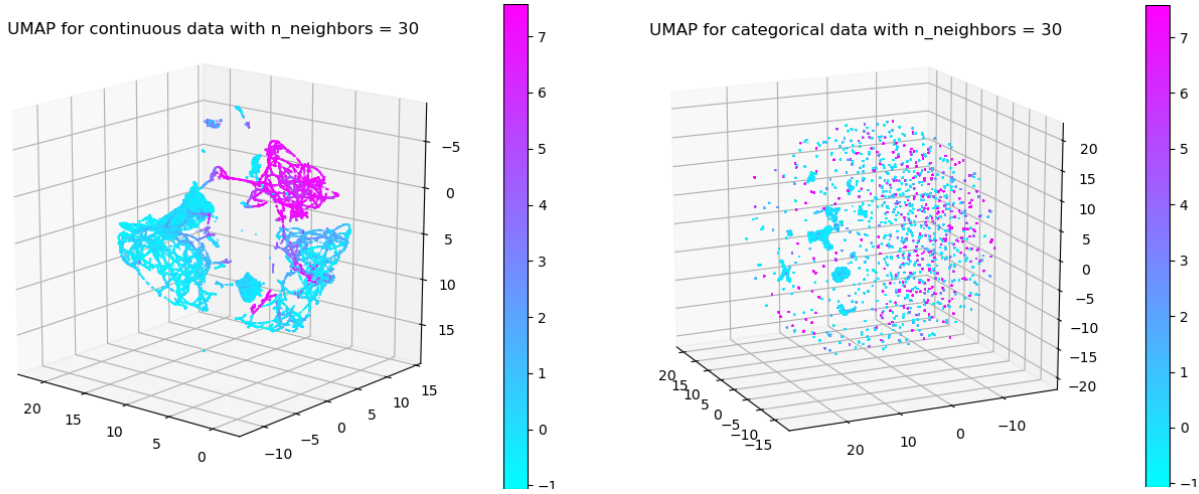
When the number of neighbourhood increases from 15 to 30, fraudulent cases are more connected to each other in the embedded representation for continuous features. So, it preserves more global structure of the training data. If we colour the output from Umap with the claim amounts, a weighting variable which is not included in training the Umap, we observe fraudulent cases with high claiming amounts are staying closer to each other in Figure 22. Thus, Umap seems able to capture the topological structure of fraudulent cases with high claiming amount. This pattern is less clear for categorical features but it also depends on the distance metric for categorical features.

Ideally, we can construct a weighted Umap model combining both Umap models for both continuous and categorical features. However, it may introduce extra hyperparameters to tune which make the training process much complicated. For example, we may need to tune the weighting parameter which assign weights to both models. Moreover, it will make the training process much time-consuming because the combined model also requires optimization. However, it can give us a more holistic view on the training data.



(a) Continuous features (b) Categorical features

Figure 21: Visualization of Umap output in figure 20 coloured by `claim_amount`



(a) Continuous features (b) Categorical features

Figure 22: Rotated version of figure 21

Outlier detection

In this section, we apply outlier detection algorithm to the training dataset so as to find out extreme observations. In particular, we are also interested if extreme observations are associated with higher claiming amounts and fraudulent claims. It is possible to perform outlier detection using boxplot for each continuous features. However, it does not necessarily reflect an outlier in a higher dimensional space. In contrast to other popular outlier detection algorithms, isolation forest has much fewer hyperparameters to tuned when compared to one-class support vector machine and does not require distance metric like local outlier factor method. As scikit learn does not support Gower's distance in local outlier factor method, isolation forest becomes a more appealing tool. We use isolation forest from `sklearn.ensemble`, and it can assign anomaly scores based on the number of partitions needed to isolate an observation. Anomaly scores are further analyzed with other features, for example, `claim_amount` and `fraud`.

In figure 23, we observe that fraudulent cases do not have anomaly scores higher than non-fraudulent cases. There is no obvious pattern between the anomaly scores and their claim amounts as well. So, observations with higher anomaly scores do not necessarily have higher claim amounts. In fact, the median of anomaly scores for fraudulent cases is lower than that for non-fraudulent cases.

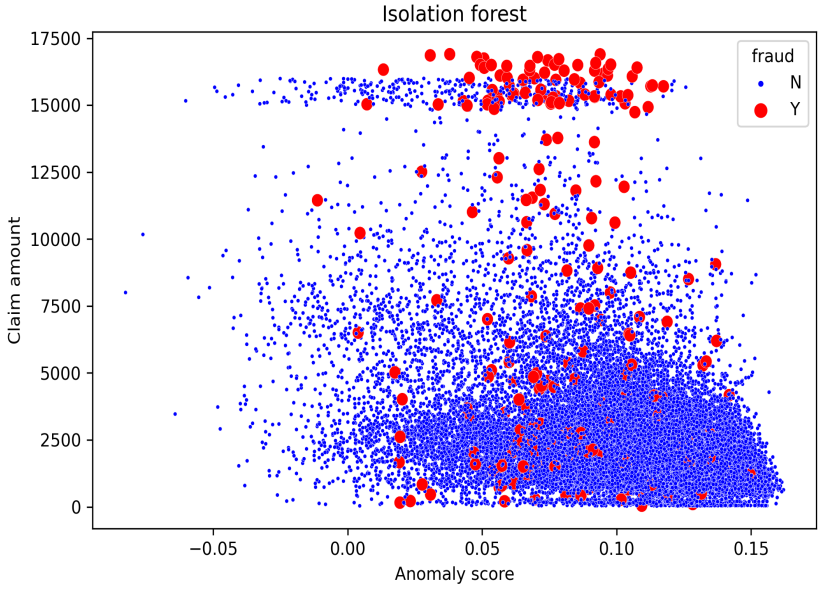


Figure 23: Joint distribution of anomaly scores estimated by isolation forest and their claim amounts

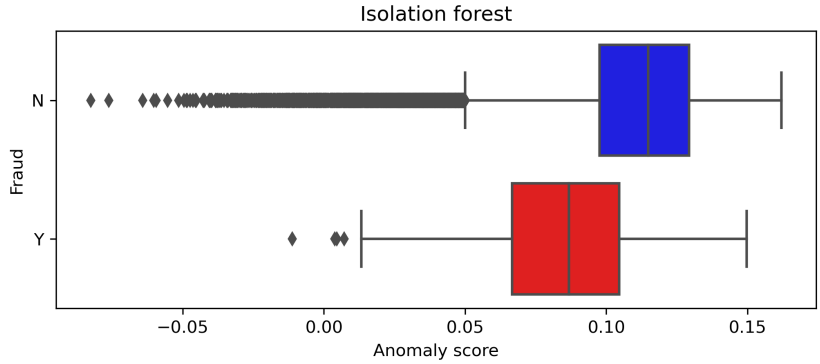


Figure 24: Distribution of anomaly scores by fraudulent status

Model building

The goal of this task is to predict top 100 fraudulent claims in terms of their claim amounts. In this section, we outline our model building process and justify how our choices are made.

Algorithms

In this subsection, we summarize the motivations, packages and hyperparameters of the algorithms in the model building process.

As our training data contains missing values for many features. Unlike categorical features, missing values in continuous features require imputation because it cannot be grouped as an extra factor level. There are several popular imputation techniques, including mean, median, mode imputations in `SimpleImputer` and kNN imputation in `KNNImputer` from `sklearn`. Among mean, median and mode imputations, median imputation is the best choice because mean imputation reduces feature variances and mode imputation may not be suitable for continuous features which usually has distinct values. Compared to median imputation, kNN imputer is a better choice because different values are assigned based on its neighbourhood. During the training process, we discovered kNN imputer lengthens the training time by a lot in cross validation for grid search and probability calibration. Taking training time into account, we apply median imputer in the model.

As discussed in the section of exploratory data analysis, the dataset is highly imbalanced which requires sampling so that the classification becomes feasible. Otherwise, the classifier tends to lend the features of the majority class. As there are only 308 fraudulent cases, undersampling would lead to insufficient sample size in the training process. For oversampling, replication of observations form the minority class may encourage overfitting of the algorithm because the same patterns appear many time. Therefore, we need smart sampling technique which introduces some variability during sampling and reduce the risk of overfitting. For the task, we apply synthetic minority oversampling technique (SMOTE) which is available in `imblearn` library.

SMOTE is an algorithm which combines both oversampling and k nearest neighbourhood (kNN). For the minority class, the features of new sample are synthesized based on the k nearest neighbourhood of the observations from the minority class. Therefore, we need to tune the parameter k for the number of nearest neighbourhood in kNN. We consider k from 3 to 9.

For classifier, we focus on ensemble methods, and we use random forest, which is a classical and well-known ensemble method, as our baseline classifier, implemented in `sklearn` as `RandomForestClassifier`. Random forest requires only very few hyperparameters and is more robust to overfitting because the final prediction is calculated by the majority voting of individual decision trees. Some important hyperparameters in random forest are the maximum number of features used in constructing individual trees and the total number of such individual trees. We consider the maximum number of features from 10 to 70. For the total number of individual trees, we just need to ensure that it is sufficient for averaging effect to take place (and hence reduce the risk of overfitting).

Apart from random forest, we consider one of the latest ensemble technique, light gradient boosting machine (LightGBM), which is released 5 years ago. It is an efficient algorithm but comes with more than 100 hyperparameters, available in python as `lightgbm`. The details of all available hyperparameters can be found at https://lightgbm.readthedocs.io/en/latest/

Parameters.html. LightGBM trains weak learners and grows the tree in a leaf-wise manner. LightGBM has a higher performance and efficiency but can be prone to overfitting. We consider the following hyperparameters in training LightGBM, learning rate in gradient boosting and the number of leaves for a weak learner. These hyperparameters are essential in balancing overfitting and predictive performance. We consider learning rate from 0.05 to 0.2 and the number of leaves from 10 to 30. We would compare its performance with a well-tuned random forest classifier, and further increase or reduce the search space of hyperparameters.

Grid Search and Metrics

To tune the hyperparameters mentioned in the previous section, we need to perform a grid search over the hyperparameter space by cross validation. We prefer cross validation over train-validation set approach because using a single validation set can yield unstable performance metrics due to the randomness in splitting. Cross validation reduces the randomness by splitting the dataset into k folds, and takes the average performance metrics.

Probability Calibration

Data pipeline

Model Comparison

Interpretation

Reflection

Assignment 2

Assignment 3

Assignment 4