# Insurance Loss Analytics Report

By Group 23:
Gihyeon Kwon (8766526948), Gray Underhill (7582020489),
Heng-Yao Cheng (8985680635), Morgan Budidjaja (7748057449),
Raymond Moy (6106900274), Xingjian Shao (2451945705)

Contact Person: Morgan Budidjaja
budidjaj@usc.edu

DSO530: Applied Modern Statistical Learning Methods
Section: 20251_16219
Professor Paromita Dubey
May 8, 2025

# Executive Summary

Setting accurate premiums is a critical challenge in the insurance industry. Mispricing can lead to adverse selection, where profitable, lower-risk policyholders leave for better-priced competitors while higher-risk individuals remain. This imbalance threatens financial performance and portfolio stability.

On the other hand, mispricing the premium too low will result in less profitability of the company as well as losing the financial buffer to the risk of high impact accidents. An accurate prediction of the claim losses ensures competitive pricing following National Association of Insurance Commissioners (NAIC)'s regulation on Risk-Based Capital (RBC).

This project aims to build data-driven predictive models to support more accurate and risk-sensitive pricing in auto insurance. Specifically, we forecast claim losses through two target variables: Loss Cost per Exposure Unit (LC) and Historically Adjusted Loss Cost (HALC). These continuous outcomes are evaluated using Mean Squared Error (MSE) to assess the model's performance in estimating expected claim severity.

In addition, we develop a binary classification model to predict the likelihood that a policyholder will file a claim. This probability-based risk assessment enables targeted pricing strategies and better customer segmentation. The classification performance is measured using the Receiver Operating Characteristic Area Under the Curve (ROC-AUC), reflecting the model's ability to distinguish between claim and non-claim cases.

Initially attempting machine learning models like multiple linear regression and decision trees, we soon realized limitations with these simple models. We decided to use a strong boosting model, XGBoost, which performed the best compared to other ML models including deep learning models.

For Task 2, resulting in a ROC-AUC of 0.85 for classification. The end result was after the hyperparameter tuning, ensuring the model would perform better on unseen test data. The result of Task 1 is a MSE of 410K for LC and a MSE of 1.37M for HALC on our dataset. Using a two-step approach, first classifying the profile into CS = 1 and 0, we ran a separate XGBoost Regressor model that only trained on profiles with a 1 claim status.

In future processing, we aim to improve our model's ability to generalize on unseen test data. Adding interaction terms, more advanced feature engineering depending on SHAP analysis and two step modeling for the tail risk are some of the next steps for this project.

These predictive models help insurers segment policyholders more effectively, identify high-risk profiles early, and comply with solvency standards such as the NAIC's RBC guidelines. Ultimately, this improves premium adequacy, minimizes underwriting losses, and strengthens financial resilience in an increasingly competitive market.

## Data Pre-Processing Steps [1]

1. Compute the variables ‘LC’, ‘HALC’, and ‘CS’ which will be used as the response variables in different models.
    a. LC = Total Cost of Claims, Current Year / Total number of claims, current Year
    b. HALC = LC / Ratio of the number of claims filed to the total year
    c. CS = 1 if  Total number of claims, current Year > 0 else 0
2. Deal with null or missing values using either 0’s, the mean, or the mode as appropriate according to the nature of the column.
3. Added dummy variables into the dataset for the relevant categorical features using one-hot encoding.
4. Transform date columns into new columns with more useful information: policy duration in days, time since last renewal in days, age of the driver, and age of the driver’s license.
5. Drop the necessary columns for the datasets that will be used to predict ‘LC’, ‘HALC’, and ‘CS’.

### Task 1: LC & HALC [2]

*Linear Regression*

We began our modeling process with multiple linear regression to establish a performance baseline. This approach offers straightforward implementation and produces easily interpretable coefficients. However, the model exhibited poor performance on our dataset. Diagnostic checks revealed several violations of linear regression assumptions, including non-normal distribution of residuals, multicollinearity among predictors, and heteroscedasticity (non-constant variance of errors). Linear regression also assumes a linear relationship that is often unrealistic in the context of insurance claim data which is inherently skewed and influenced by complex, nonlinear interactions. Consequently, while useful for interpretation, the linear model lacked the flexibility needed to produce accurate predictions.

*Tweedie XGBoost with Cross-Validation and Hyperparameter Tuning*

To model the highly skewed and zero-inflated insurance claims data, we implemented a Tweedie XGBoost model, well-suited for targets like Loss Cost (LC) and HALC. Starting with a base configuration (variance power = 1.5), we performed 5-fold cross-validation and achieved a cross-validated RMSE of 742.46 at the 115th boosting round for LC.

We then conducted a randomized hyperparameter search over 150 combinations, tuning parameters such as learning rate, tree depth, regularization terms, and Tweedie variance power. The best configuration (variance power = 1.1, depth = 6, learning rate = 0.05) achieved a comparable RMSE of 754.68, confirming the model's consistency and resilience across settings.

The same process was applied to HALC, where the base model yielded an RMSE of 1446.20, and the tuned model achieved 1511.43. Although $R^2$ remained low due to high variance in the targets, the model consistently captured the core signal better than simpler baselines. Tweedie XGBoost struck a strong balance between predictive power and interpretability, aided by SHAP-based feature explanations.

*LightGBM Tweedie with Cross-Validation and Hyperparameter Tuning*

We also explored LightGBM, favored for its speed, efficiency, and scalability on structured datasets. Using a similar Tweedie objective and hyperparameter set (variance power = 1.1, learning rate = 0.04), we ran 5-fold cross-validation after standardizing features.

For LC, LightGBM reached a cross-validated RMSE of 743.13 at boosting round 128—nearly identical to XGBoost but with faster training and lower computational cost. For HALC, the model achieved a best RMSE of 1448.04, again closely mirroring XGBoost's performance. While it didn't outperform XGBoost in accuracy, LightGBM's training speed and ease of deployment make it an appealing choice for production-scale modeling.

*Outlier Removal Trial*

To assess model robustness, we experimented with removing outliers using a ±2.5 z-score threshold on LC (Figures 1) and HALC (Figures 2), applying this filter only to the training and validation sets. This preprocessing led to improved validation performance across models: linear regression reached 158.08 (LC) and 314.20 (HALC), random forest achieved 154.08 (LC) and 308.93 (HALC), and Tweedie XGBoost obtained 157.03 (LC) and 312.35 (HALC).

However, when tested on the full, unfiltered dataset, all models experienced significant performance drops. For example, Tweedie XGBoost's RMSE rose to 882.37 for LC and 1484.26 for HALC, revealing that trimming outliers reduced generalizability. Given the importance of capturing rare, high-loss events in insurance modeling, we ultimately trained our final models on the complete dataset to ensure real-world applicability.

**Task 2: CS**

*Logistic Regression*

For predicting the occurrence of a claim, we applied logistic regression as a baseline model due to its ease of interpretation. Running this basic model with no tuning, we obtained a 73% prediction accuracy and ROC-AUC value of 0.79 on our test dataset. This left room for plenty of improvement with more complex models and serves as a helpful point of comparison.

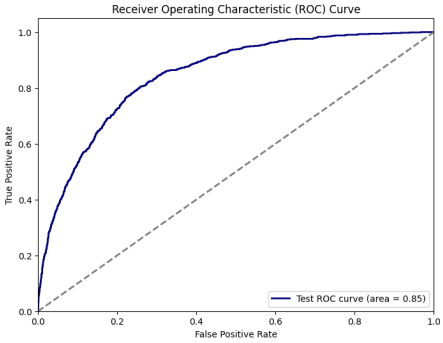
*Neural Network Classification*

After testing a basic logistic regression model, we moved on to a neural network classifier with two hidden layers of 128 and 64 neurons, using a sigmoid activation for the output layer. The model was trained with a binary cross-entropy loss function, and we applied dropout along with early stopping callbacks to prevent overfitting. Logistic regression provided a solid baseline, but given the complexity and non-linearity present in the data, it struggled to capture nuanced patterns. In contrast, the neural network was able to model more complex interactions between features, which led to a significant improvement. Our best ROC-AUC score increased to 0.835, making it a more effective tool for classifying claim status in this context.

## Final Model for Task 2 [3]

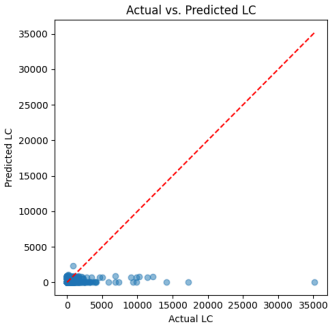
We applied XGBoost for the classification task and conducted extensive hyperparameter tuning to improve the model's predictive performance. Specifically, we tuned parameters such as learning rate, maximum tree depth, number of estimators, subsample ratio, and column

subsampling. These parameters were chosen because they directly impact the model's ability to balance bias and variance. Learning rate and tree depth help control how aggressively the model learns, while subsample and colsample_bytree introduce regularization to reduce overfitting and improve generalization.

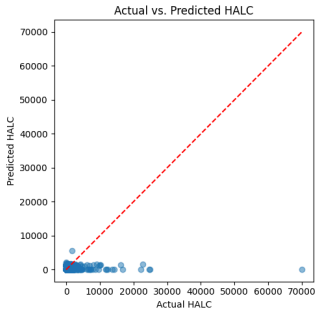



It handles imbalanced data well, supports missing values, and can capture complex nonlinear relationships with minimal preprocessing. These strengths align well with the nature of insurance data, which often includes a mix of categorical and continuous variables, missing values, and skewed distributions. For insurance companies, XGBoost offers a powerful yet practical tool for identifying policyholders who are likely to make a claim. With a final ROC-AUC score of 0.849, it outperformed all other classification models in our approaches.

## Final Creative Approach for Task 1 [4]




The distributions of LC and HALC are highly skewed, with most values being zero. To address this, we implemented a **creative two-step approach.** First, we used the CS classification model to estimate the probability of a claim. Then, we trained the XGBoost regressor from Task 1 using only the subset of training data where CS equals 1, focusing on rows where a claim has occurred. After applying the classification model to the entire dataset, we used a threshold on the predicted probabilities to decide which rows should receive LC and HALC predictions. For rows below the threshold, we assumed both LC and HALC would be zero. By tuning this threshold (Figure 3), we achieved an LC RMSE of 640.87, a HALC RMSE of 1172.55, and a CS ROC-AUC of 0.849. This method avoids applying a single model across a skewed dataset dominated by zeros and outliers. Instead, it separates classification from regression, reducing error and improving the accuracy of claim-based cost predictions for insurance companies.




## Additional Notes

The final CS model was tuned based on ROC-AUC, while the regressors were evaluated using MSE. Since the regression models were trained only on rows with CS = 1, zero-inflation was no longer a concern, allowing us to switch the loss function from Tweedie to squared error for better fit. While our classification model could have been further optimized for recall to capture more claim cases, we focused on ROC-AUC and MSE to align with the evaluation criteria. Initially, we also considered using clustering methods to identify groups with similar claim behavior, but opted for a supervised classification approach due to its clearer alignment with predictive

accuracy and business interpretability. This trade-off balanced classification precision with regression accuracy for overall performance in real-world insurance decision-making.

## Model Interpretation [5]

*Task 1 & 2 SHAP*

The SHAP analyses (Figure 4) for the LC, HALC, and CS models reveal consistent patterns in the features most predictive of loss severity and claim likelihood, offering meaningful insights for underwriting and pricing decisions. Shorter time to next renewal and higher net premium were strong indicators of increased risk across all models. Features like total policies, contract age, and vehicle value helped differentiate between more stable, long-term policyholders and those with newer or higher-value contracts. The HALC model emphasized behavioral and demographic signals such as nonpayment history and age of insured, reflecting its focus on historical claim frequency. The CS model highlighted similar trends, reinforcing the role of customer behavior in predicting claims. These findings support a more personalized pricing strategy that reduces adverse selection and improves reserve planning by aligning premiums more closely with individual policyholder risk.

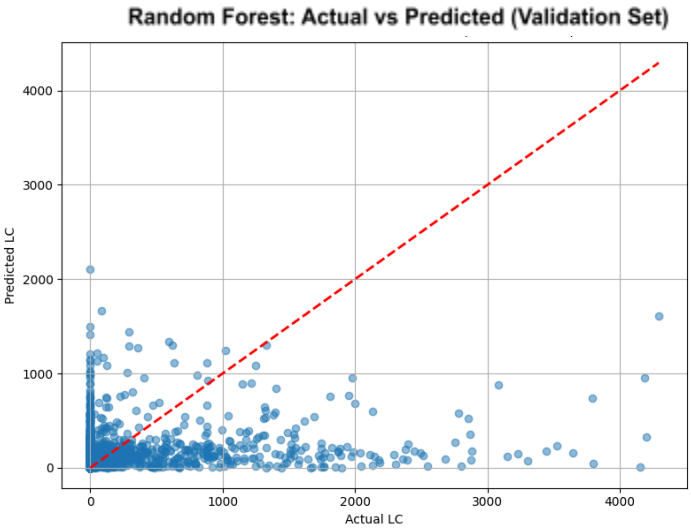
### Business Implications

In the insurance industry, machine learning is used as a complementary tool to estimate the financial capital needed to reserve for covering risk. The company sums up the total estimated HALC for the upcoming year, takes the 99% quantile upper extreme values for the population, and reserves capital to cover any disastrous accidents. The Law of Large Numbers (LLM) is the core to making the prediction. Each policyholder faces random risk (accident, death, loss, etc), but aggregating the risk for a large number of sample policyholders, the total average will be closer to the real population average. However, we need to verify our model performance on the test data, to check the generalization ability of the model on unforeseen data, so we can create real-life impact with our machine learning model.

## Conclusion

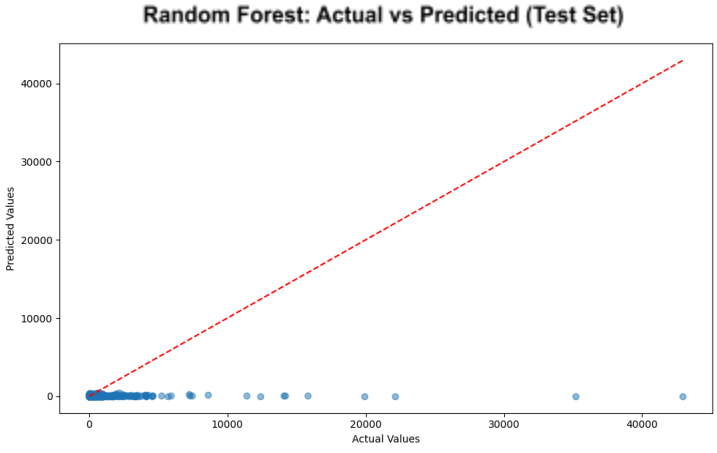
Accurately forecasting insurance claims is vital for maintaining pricing competitiveness, managing risk exposure, and complying with regulatory capital requirements. We began with baseline models such as linear regression, which offered interpretability but were limited by its inability to capture complex, nonlinear patterns from the insurance data. After transitioning to more advanced methods, Tweedie XGBoost with Cross-Validation and Hyperparameter Tuning and a classification-regression two-step model, we significantly improved predictive performance. Our final models demonstrated strong predictive power for claim likelihood and reducing loss prediction error through targeted modeling of claim-positive cases. SHAP analysis further enabled transparency, highlighting actionable drivers like renewal timing, policyholder behavior, and premium value. These insights could directly inform underwriting decisions, refine pricing strategies, and support any capital allocations. The modeling framework is scalable and adaptable, offering insurers a practical toolset for improving profitability and portfolio stability in a volatile market.

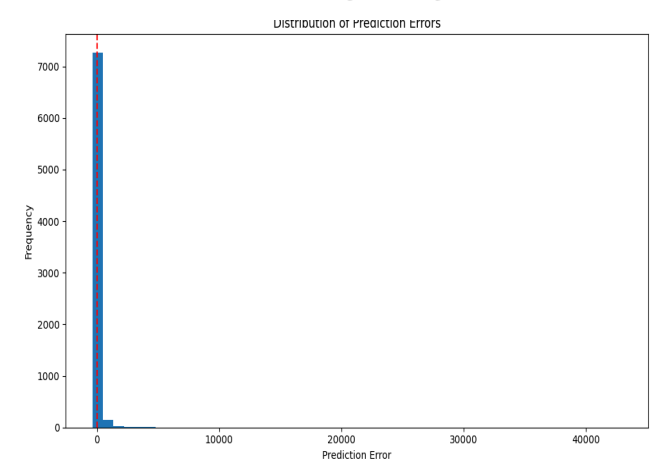
## Appendix
## Outlier Removal Approach (LC) (Figures 1)




On the filtered validation set, the model tends to underpredict high LC values and clusters heavily around low to mid-range predictions. While performance appears relatively controlled within the expected LC range (due to outlier removal), the scatter remains wide, suggesting variability in prediction precision even within the trimmed data.

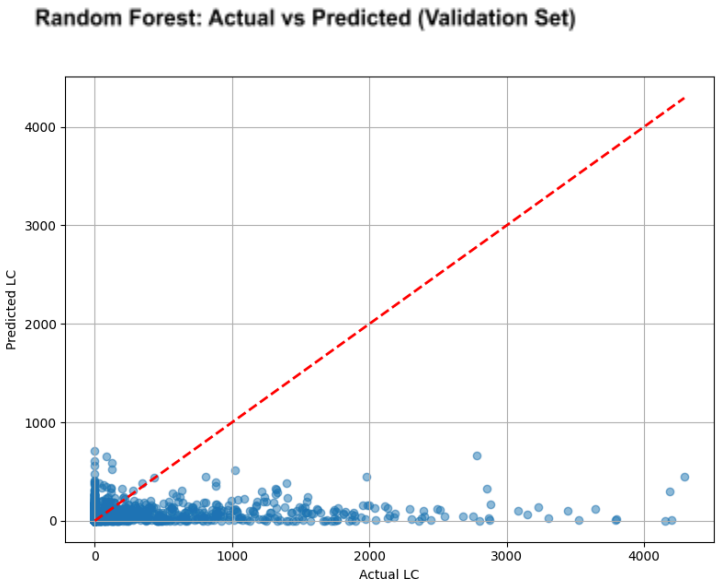
When the same model is applied to the untouched test set—which includes the full range of LC values including extreme cases—the predictive weakness becomes clear. The model fails to capture the magnitude of high LC values, with most predictions collapsing toward the lower end. This confirms the issue of model fragility when trained only on filtered data, especially in domains like insurance where outliers are common and materially important.



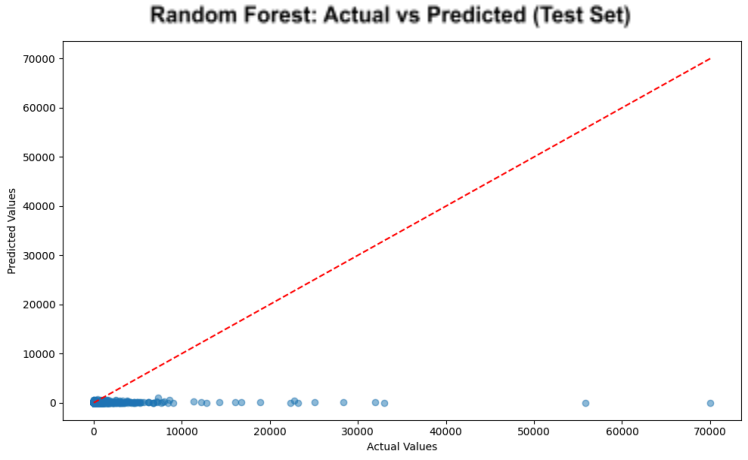

The histogram of prediction errors on the test set shows a heavy right skew. The vast majority of predictions incur small errors, but there are notable spikes in large errors—reflecting the model's inability to predict extreme losses. These few but significant underpredictions could have serious financial consequences in real-world insurance applications.

## Outlier Removal Approach (HALC) (Figures 2)




On the trimmed validation set, HALC predictions mirror those of LC in their tendency to underestimate higher values, with dense clustering around lower predicted values. However, HALC exhibits even less vertical spread, suggesting the model struggles more with precision, likely because it's capturing both severity and frequency effects without fully modeling their interaction.

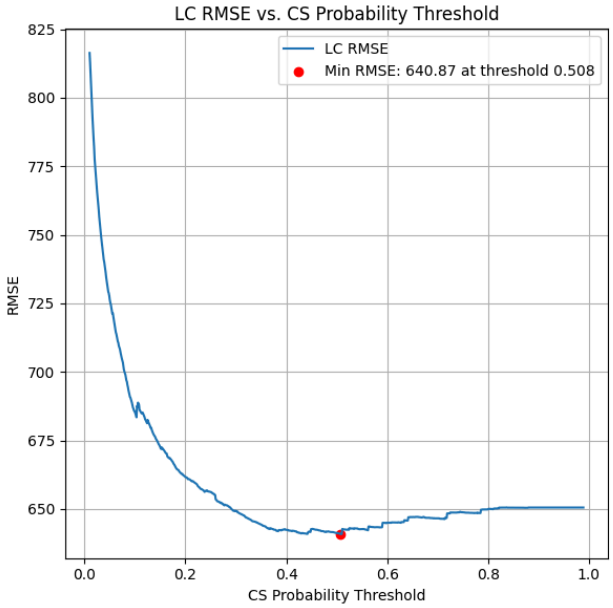
When the model is exposed to the full HALC distribution, prediction quality deteriorates notably. Many extreme actual values receive overly compressed predictions, with a visible gap between predicted and actual values for the most expensive cases. This indicates limited generalization capacity when faced with out-of-distribution or high-leverage samples.

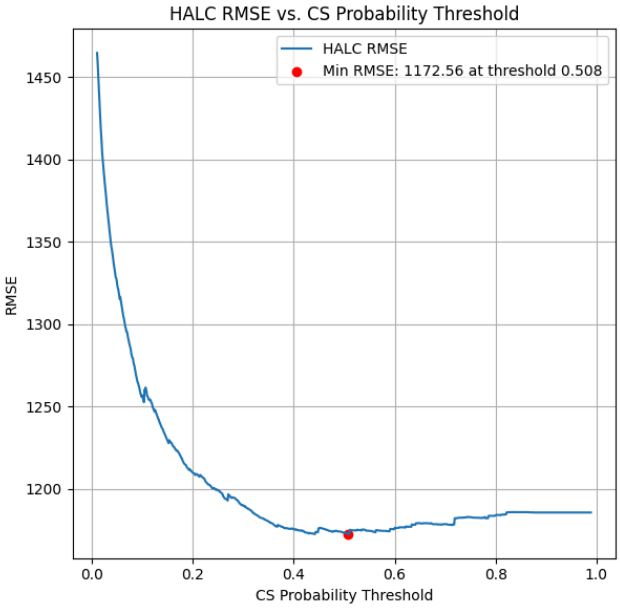




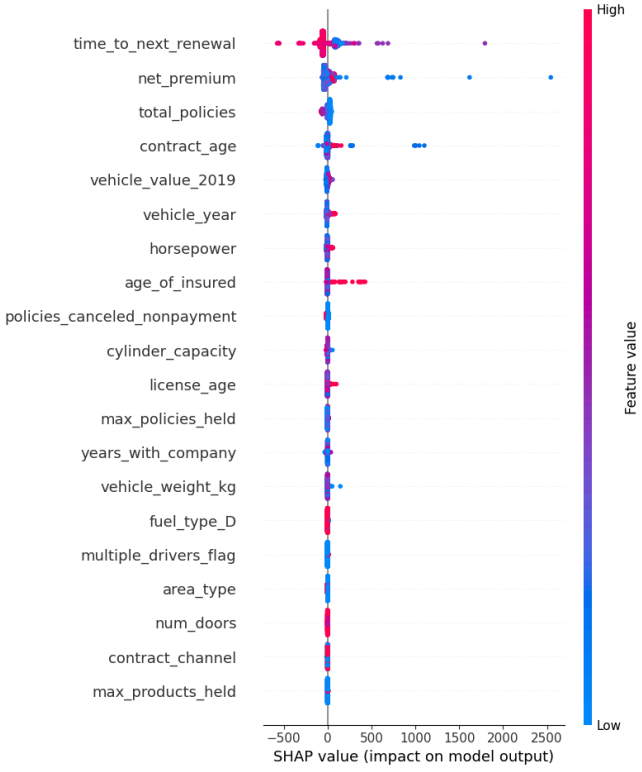
The histogram again reveals an extremely right-skewed distribution of prediction errors, though the spikes are more severe in HALC than LC. This reinforces the idea that the model is disproportionately misestimating the highest-risk policyholders, which poses a substantial challenge for pricing or reserving in actuarial contexts.

## LC/HALC Threshold Tuning (Figure 3)

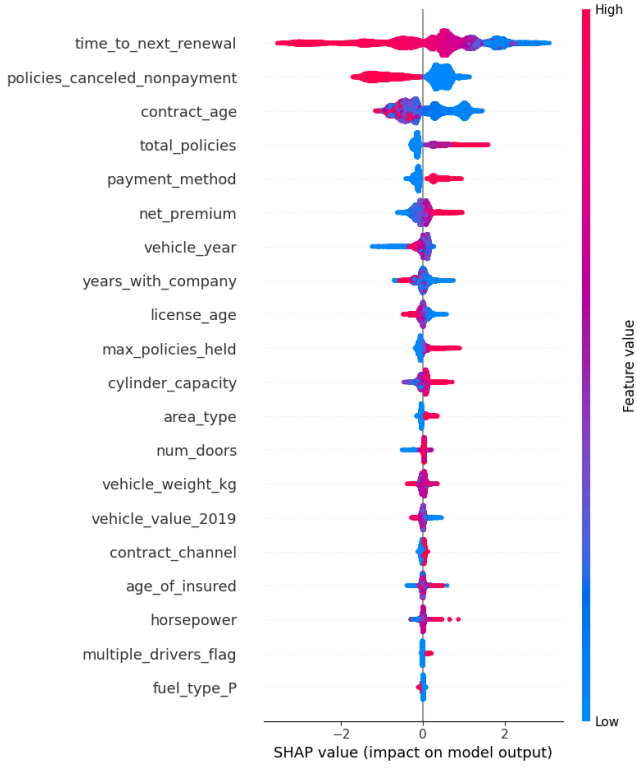



## SHAP Task 1 (Figure 4)




## SHAP Task 2



# Insurance Loss Analytics

Author: Group 23

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

In [2]:
```
import numpy as np
import pandas as pd
import os
import math
import statsmodels.api as sm
from datetime import datetime
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
pd.set_option("display.max_columns", None)

from sklearn.metrics import r2_score, mean_squared_error

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import train_test_split
os.chdir('/content/drive/MyDrive/530 Project/data')
os.listdir()
```

Out[2]:
```
['insurance_train.csv',
 'insurance_test.csv',
 'checkpoints',
 'group_x_prediction1.csv',
 'checkpoints_cs',
 'best_model_cs.keras',
 'group_x_prediction.csv',
 'group_23_prediction.csv',
 'LC_train.csv',
 'HALC_train.csv',
 'insurance_pre_train.csv',
 'train_preprocess_gk.csv']
```

## [1] Preprocessing

In [29]:
```
df = pd.read_csv("insurance_train.csv")

df["LC"] = np.where(df['X.16'] != 0, df['X.15']/df['X.16'], 0)
df["HALC"] = np.where(df['X.16'] != 0, (df['X.15']/df['X.16']) * df['X.18'], 0)
df["CS"] = np.where(df['X.16'] != 0, 1, 0)

column_rename_dict = {
    "X.1": "contract_id",
    "X.2": "contract_start_date",
    "X.3": "last_renewal_date",
    "X.4": "next_renewal_date",
    "X.5": "insured_birth_date",
    "X.6": "license_issue_date",
    "X.7": "contract_channel",
    "X.8": "years_with_company",
    "X.9": "total_policies",
    "X.10": "max_policies_held",
    "X.11": "max_products_held",
    "X.12": "policies_canceled_nonpayment",
    "X.13": "payment_method",
    "X.14": "net_premium",
    "X.15": "claims_cost_current_year",
    "X.16": "claims_count_current_year",
    "X.17": "claims_count_total",
    "X.18": "claims_per_year_ratio",
    "X.19": "vehicle_type",
    "X.20": "area_type",
    "X.21": "multiple_drivers_flag",
    "X.22": "vehicle_year",
    "X.23": "horsepower",
    "X.24": "cylinder_capacity",
    "X.25": "vehicle_value_2019",
    "X.26": "num_doors",
    "X.27": "fuel_type",
    "X.28": "vehicle_weight_kg"
}
df = df.rename(columns=column_rename_dict)
```

```
df['fuel_type'] = df['fuel_type'].fillna(df['fuel_type'].mode()[0])

date_columns = ['contract_start_date', 'last_renewal_date', 'next_renewal_date',
                'insured_birth_date', 'license_issue_date']

for col in date_columns:
    df[col] = pd.to_datetime(df[col], format='%d/%m/%Y', errors='coerce')

df['contract_age'] = (datetime.now() - df['contract_start_date']).dt.days / 365
df['age_of_insured'] = (datetime.now() - df['insured_birth_date']).dt.days / 365
df['license_age'] = (datetime.now() - df['license_issue_date']).dt.days / 365
df['time_to_next_renewal'] = (df['next_renewal_date'] - datetime.now()).dt.days / 365

vehicle_type_dummies = pd.get_dummies(df['vehicle_type'], prefix='vehicle_type')
fuel_type_dummies = pd.get_dummies(df['fuel_type'], prefix='fuel_type')

df = pd.concat([df, vehicle_type_dummies], axis=1)
df = pd.concat([df, fuel_type_dummies], axis=1)

df.drop(columns=date_columns + ['contract_id', 'vehicle_type', 'fuel_type',
                                'claims_cost_current_year','claims_count_current_year',
                                'claims_count_total', 'claims_per_year_ratio'], inplace=True)
```

In [30]:
```
# df.to_csv("train_preprocess_gk.csv")
```

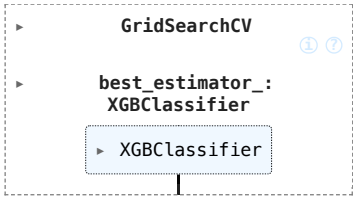
## [3] Task 2: CS

In [31]:
```
X = df.drop(columns=["LC", "HALC", "CS"])
y_cs = df["CS"]
y_lc = df["LC"]
y_halc = df["HALC"]

# Unified split across all targets
X_train_CS, X_test_CS, y_train_CS, y_test_CS, y_LC_train, y_LC_test, y_HALC_train, y_HALC_test = train_test_split(
    X, y_cs, y_lc, y_halc,
    test_size=0.2,
    stratify=y_cs,
    random_state=42
)
```

In [32]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

xgb_model = XGBClassifier(
    objective='binary:logistic',
    use_label_encoder=False,
    eval_metric='auc',
    random_state=42,
    # scale_pos_weight=(len(y_train_CS[y_train_CS == 0]) / len(y_train_CS[y_train_CS == 1]))
)

param_grid = {
    "learning_rate": [0.01, 0.1],
    "n_estimators": [100, 200],
    "max_depth": [3, 5],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.5, 0.8]
}

grid = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=3,
    verbose=1,
    n_jobs=-1
)

grid.fit(X_train_CS, y_train_CS)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [22:54:05] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

Out[32]:




In [33]:
```
from sklearn.metrics import roc_auc_score

# Best model from GridSearchCV
best_model = grid.best_estimator_

# Evaluate on final test set
test_probs = best_model.predict_proba(X_test_CS)[:, 1]
test_auc = roc_auc_score(y_test_CS, test_probs)
print("Test ROC-AUC:", test_auc)

# If you want classification threshold results too:
from sklearn.metrics import classification_report
print(classification_report(y_test_CS, best_model.predict(X_test_CS)))
```

```
Test ROC-AUC: 0.8493163767515642
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      6661
           1       0.62      0.17      0.26       830

    accuracy                           0.90      7491
   macro avg       0.76      0.58      0.60      7491
weighted avg       0.87      0.90      0.87      7491
```

## [2] TASK 1 model

In [34]:
```
X_train_lc_halc = X_train_CS[y_train_CS == 1]
y_LC_train_cs1 = y_LC_train[y_train_CS == 1]
y_HALC_train_cs1 = y_HALC_train[y_train_CS == 1]
```

In [35]:
```
#LC

import xgboost as xgb
from sklearn.model_selection import GridSearchCV


param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'subsample': [0.8],
    'colsample_bytree': [0.7],
    'reg_alpha': [0, 10],
    'reg_lambda': [1, 10]
}
# xg_reg = xgb.XGBRegressor(
#     objective='reg:tweedie',
#     tweedie_variance_power=1.5,  # fixed
#     random_state=42
# )
xg_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    random_state=42
)


grid_cv_lc = GridSearchCV(
    estimator=xg_reg,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    verbose=1,
    n_jobs=-1
)

import time

start_time = time.time()
grid_cv_lc.fit(X_train_lc_halc, y_LC_train_cs1)
end_time = time.time()

print(f"GridSearchCV took {(end_time - start_time)/60:.2f} minutes")

# Output the best found hyperparameters and the best score
```

```
print("Best hyperparameters:", grid_cv_lc.best_params_)
print("Best cross-validation score (negative MSE):", grid_cv_lc.best_score_)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
GridSearchCV took 0.56 minutes
Best hyperparameters: {'colsample_bytree': 0.7, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100, 'reg_alpha': 10,
'reg_lambda': 10, 'subsample': 0.8}
Best cross-validation score (negative MSE): -6648627.848680933
```

In [36]:
```
#HALC
import xgboost as xgb
from sklearn.model_selection import GridSearchCV


param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'subsample': [0.8],
    'colsample_bytree': [0.7],
    'reg_alpha': [0, 10],
    'reg_lambda': [1, 10]
}


xg_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    random_state=42
)

grid_cv_halc = GridSearchCV(
    estimator=xg_reg,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    verbose=1,
    n_jobs=-1
)

# Fit GridSearchCV to the training data
import time

start_time = time.time()
grid_cv_halc.fit(X_train_lc_halc, y_HALC_train_cs1)
end_time = time.time()

print(f"GridSearchCV took {(end_time - start_time)/60:.2f} minutes")

# Output the best found hyperparameters and the best score
print("Best hyperparameters:", grid_cv_halc.best_params_)
print("Best cross-validation score (negative MSE):", grid_cv_halc.best_score_)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
GridSearchCV took 0.46 minutes
Best hyperparameters: {'colsample_bytree': 0.7, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100, 'reg_alpha': 0, 'r
eg_lambda': 10, 'subsample': 0.8}
Best cross-validation score (negative MSE): -26881530.289080054
```

## [4] Use CS to predict LC and HALC

In [37]:
```
import math
def two_step_model(model_lc, model_halc, model_cs, cs_prob_threshold):
    # Predict CS (claim status)
    cs_probs = model_cs.predict_proba(X_test_CS)[:, 1]
    cs_preds = (cs_probs >= cs_prob_threshold).astype(int)

    # Initialize LC and HALC predictions as zeros
    lc_preds = np.zeros(len(X_test_CS))
    halc_preds = np.zeros(len(X_test_CS))

    # Indices where CS == 1
    cs_1_indices = np.where(cs_preds == 1)[0]

    # Apply LC & HALC models only to those
    lc_preds[cs_1_indices] = model_lc.predict(X_test_CS.iloc[cs_1_indices])
    halc_preds[cs_1_indices] = model_halc.predict(X_test_CS.iloc[cs_1_indices])

    final_preds = pd.DataFrame({
        "Predicted_CS": cs_preds,
        "Predicted_LC": lc_preds,
        "Predicted_HALC": halc_preds
    })

    # Calculate metrics
    LC_RMSE = math.sqrt(mean_squared_error(y_LC_test, final_preds["Predicted_LC"]))
    HALC_RMSE = math.sqrt(mean_squared_error(y_HALC_test, final_preds["Predicted_HALC"]))
```

```
    ROC_AUC = roc_auc_score(y_test_CS, cs_probs)

    return LC_RMSE, HALC_RMSE, ROC_AUC
```

In [38]:
```
from tqdm import tqdm
model_lc = grid_cv_lc.best_estimator_
model_halc = grid_cv_halc.best_estimator_

model_cs = grid.best_estimator_

out_dict = {}
for t in tqdm(np.arange(0.01, 0.99, 0.001)):
  LC_RMSE, HALC_RMSE, ROC_AUC = two_step_model(model_lc, model_halc, model_cs, t)

  out_dict[t] = [LC_RMSE, HALC_RMSE, ROC_AUC]
```

```
100%|██████████| 980/980 [01:04<00:00, 15.14it/s]
```

In [39]:
```
pd.DataFrame(out_dict)
```

Out[39]:

| | 0.010 | 0.011 | 0.012 | 0.013 | 0.014 | 0.015 | 0.016 | 0.017 | 0.018 | 0.019 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 816.293006 | 812.771640 | 808.378259 | 803.145734 | 799.131961 | 795.522572 | 791.731014 | 787.933156 | 784.771311 | 781.695129 | 777 |
| **1** | 1464.681672 | 1459.690427 | 1452.929369 | 1445.208356 | 1438.826927 | 1433.402655 | 1426.709138 | 1420.630910 | 1415.909411 | 1410.619778 | 1404 |
| **2** | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0.849316 | 0 |

In [40]:
```
# Convert the dictionary to a DataFrame for easier manipulation
df_out_dict = pd.DataFrame(out_dict).T.reset_index()
df_out_dict.columns = ['threshold', 'LC_RMSE', 'HALC_RMSE', 'ROC_AUC']


# Find the keys (thresholds) corresponding to the lowest LC_RMSE and HALC_RMSE
lowest_LC_RMSE_key = df_out_dict.loc[df_out_dict['LC_RMSE'].idxmin(), 'threshold']
lowest_HALC_RMSE_key = df_out_dict.loc[df_out_dict['HALC_RMSE'].idxmin(), 'threshold']

print(f"Threshold for lowest LC_RMSE: {lowest_LC_RMSE_key}")
print(f"Threshold for lowest HALC_RMSE: {lowest_HALC_RMSE_key}")
```

```
Threshold for lowest LC_RMSE: 0.5079999999999996
Threshold for lowest HALC_RMSE: 0.5079999999999996
```

In [41]:
```
print("Best RMSE for LC: ", out_dict[lowest_LC_RMSE_key][0])
```

```
Best RMSE for LC:  640.8708297279916
```

In [42]:
```
print("Best RMSE for HALC: ", out_dict[lowest_HALC_RMSE_key][1])
```

```
Best RMSE for HALC:  1172.5550081529175
```

## Results

## FINAL Prediction

In [43]:
```
df_test = pd.read_csv('insurance_test.csv')
df_test.head()
```

Out[43]:

| | X.2 | X.3 | X.4 | X.5 | X.6 | X.7 | X.8 | X.9 | X.10 | X.11 | X.12 | X.13 | X.14 | X.19 | X.20 | X.21 | X.22 | X.23 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 23/06/2017 | 23/06/2018 | 23/06/2019 | 13/09/1982 | 03/02/2011 | 0 | 2 | 2 | 2 | 1 | 1 | 0 | 240.76 | 3 | 1 | 0 | 2003 | 115 | |
| **1** | 29/06/2015 | 29/06/2016 | 29/06/2017 | 07/07/1946 | 12/08/1966 | 0 | 2 | 1 | 1 | 1 | 1 | 1 | 367.97 | 3 | 0 | 0 | 1999 | 90 | |
| **2** | 14/03/2018 | 14/03/2018 | 14/03/2019 | 26/12/1957 | 02/09/1977 | 0 | 1 | 4 | 4 | 2 | 0 | 0 | 291.90 | 3 | 0 | 0 | 2003 | 143 | |
| **3** | 16/10/2014 | 16/10/2018 | 16/10/2019 | 27/02/1961 | 29/10/1980 | 1 | 5 | 1 | 1 | 1 | 0 | 0 | 303.28 | 2 | 1 | 0 | 1998 | 60 | |
| **4** | 01/07/2015 | 01/07/2017 | 01/07/2018 | 03/07/1986 | 02/08/2006 | 0 | 3 | 1 | 1 | 1 | 1 | 0 | 333.30 | 3 | 0 | 0 | 2015 | 66 | |

In [44]:
```
df_test.shape
```

Out[44]:
```
(15787, 23)
```

In [45]:
```
column_rename_dict = {
    "X.1": "contract_id",
    "X.2": "contract_start_date",
    "X.3": "last_renewal_date",
    "X.4": "next_renewal_date",
    "X.5": "insured_birth_date",
    "X.6": "license_issue_date",
    "X.7": "contract_channel",
```

```
    "X.8": "years_with_company",
    "X.9": "total_policies",
    "X.10": "max_policies_held",
    "X.11": "max_products_held",
    "X.12": "policies_canceled_nonpayment",
    "X.13": "payment_method",
    "X.14": "net_premium",
    "X.15": "claims_cost_current_year",
    "X.16": "claims_count_current_year",
    "X.17": "claims_count_total",
    "X.18": "claims_per_year_ratio",
    "X.19": "vehicle_type",
    "X.20": "area_type",
    "X.21": "multiple_drivers_flag",
    "X.22": "vehicle_year",
    "X.23": "horsepower",
    "X.24": "cylinder_capacity",
    "X.25": "vehicle_value_2019",
    "X.26": "num_doors",
    "X.27": "fuel_type",
    "X.28": "vehicle_weight_kg"
}
df_test = df_test.rename(columns=column_rename_dict)
```

In [46]:
```
df_test['fuel_type'] = df_test['fuel_type'].fillna(df_test['fuel_type'].mode()[0])

date_columns = ['contract_start_date', 'last_renewal_date', 'next_renewal_date',
                'insured_birth_date', 'license_issue_date']

for col in date_columns:
    df_test[col] = pd.to_datetime(df_test[col], format='%d/%m/%Y', errors='coerce')

df_test['contract_age'] = (datetime.now() - df_test['contract_start_date']).dt.days / 365
df_test['age_of_insured'] = (datetime.now() - df_test['insured_birth_date']).dt.days / 365
df_test['license_age'] = (datetime.now() - df_test['license_issue_date']).dt.days / 365
df_test['time_to_next_renewal'] = (df_test['next_renewal_date'] - datetime.now()).dt.days / 365

vehicle_type_dummies = pd.get_dummies(df_test['vehicle_type'], prefix='vehicle_type')
fuel_type_dummies = pd.get_dummies(df_test['fuel_type'], prefix='fuel_type')

df_test = pd.concat([df_test, vehicle_type_dummies], axis=1)
df_test = pd.concat([df_test, fuel_type_dummies], axis=1)

df_test.drop(columns=date_columns + [ 'vehicle_type', 'fuel_type'], inplace=True)
```

In [47]:
```
cs_probs = model_cs.predict_proba(df_test)[:, 1]
cs_preds_lc = (cs_probs >= lowest_LC_RMSE_key).astype(int)
cs_preds_halc = (cs_probs >= lowest_HALC_RMSE_key).astype(int)

# Initialize LC and HALC predictions as zeros
lc_preds = np.zeros(len(df_test))
halc_preds = np.zeros(len(df_test))

# Indices where CS == 1
cs_1_indices_lc = np.where(cs_preds_lc == 1)[0]
cs_1_indices_halc = np.where(cs_preds_halc == 1)[0]

# Apply LC & HALC models only to those
lc_preds[cs_1_indices_lc] = model_lc.predict(df_test.iloc[cs_1_indices_lc])
halc_preds[cs_1_indices_halc] = model_halc.predict(df_test.iloc[cs_1_indices_halc])
```

In [48]:
```
FINAL_OUT = pd.DataFrame({"LC": lc_preds, "HALC": halc_preds, 'CS': cs_probs})
FINAL_OUT.head()
```

Out[48]:

| | LC | HALC | CS |
|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.048428 |
| **1** | 0.0 | 0.0 | 0.079023 |
| **2** | 0.0 | 0.0 | 0.280138 |
| **3** | 0.0 | 0.0 | 0.027279 |
| **4** | 0.0 | 0.0 | 0.008097 |

In [49]:
```
FINAL_OUT.shape
```

Out[49]:
```
(15787, 3)
```

In [50]:
```
FINAL_OUT
```

Out[50]:

| | LC | HALC | CS |
|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.048428 |
| **1** | 0.0 | 0.0 | 0.079023 |
| **2** | 0.0 | 0.0 | 0.280138 |
| **3** | 0.0 | 0.0 | 0.027279 |
| **4** | 0.0 | 0.0 | 0.008097 |
| **...** | ... | ... | ... |
| **15782** | 0.0 | 0.0 | 0.014070 |
| **15783** | 0.0 | 0.0 | 0.457804 |
| **15784** | 0.0 | 0.0 | 0.011223 |
| **15785** | 0.0 | 0.0 | 0.075776 |
| **15786** | 0.0 | 0.0 | 0.017924 |

15787 rows × 3 columns

In [51]:
```
FINAL_OUT.to_csv("group_23_prediction.csv", index=False)
```

In [52]:
```
FINAL_OUT.describe()
```

Out[52]:

| | LC | HALC | CS |
|---|---|---|---|
| **count** | 15787.000000 | 15787.000000 | 15787.000000 |
| **mean** | 20.914880 | 39.133386 | 0.112193 |
| **std** | 119.949442 | 226.500239 | 0.145274 |
| **min** | 0.000000 | 0.000000 | 0.000231 |
| **25%** | 0.000000 | 0.000000 | 0.018005 |
| **50%** | 0.000000 | 0.000000 | 0.053335 |
| **75%** | 0.000000 | 0.000000 | 0.143749 |
| **max** | 2354.228516 | 6027.395996 | 0.899403 |

In [53]:
```
df[['LC', 'HALC', 'CS']].describe()
```

Out[53]:

| | LC | HALC | CS |
|---|---|---|---|
| **count** | 37451.000000 | 37451.000000 | 37451.000000 |
| **mean** | 69.810230 | 128.147849 | 0.110838 |
| **std** | 838.664672 | 1665.727208 | 0.313936 |
| **min** | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 0.000000 | 0.000000 | 0.000000 |
| **max** | 118142.590000 | 236285.180000 | 1.000000 |

In [54]:
```
df[df['CS'] == 0][['LC', 'HALC', 'CS']].describe()
```

Out[54]:

| | LC | HALC | CS |
|---|---|---|---|
| **count** | 33300.0 | 33300.0 | 33300.0 |
| **mean** | 0.0 | 0.0 | 0.0 |
| **std** | 0.0 | 0.0 | 0.0 |
| **min** | 0.0 | 0.0 | 0.0 |
| **25%** | 0.0 | 0.0 | 0.0 |
| **50%** | 0.0 | 0.0 | 0.0 |
| **75%** | 0.0 | 0.0 | 0.0 |
| **max** | 0.0 | 0.0 | 0.0 |

In [55]:
```
print(f"Number of rows:&amp;nbsp;{len(FINAL_OUT)}")
print(f"Number of columns:&amp;nbsp;{len(FINAL_OUT.columns)}")
```

```
print(f"Column names: {list(FINAL_OUT.columns)}")
print(FINAL_OUT.dtypes)
```

```
Number of rows: 15787
Number of columns: 3
Column names: ['LC', 'HALC', 'CS']
LC      float64
HALC    float64
CS      float32
dtype: object
```

## [5] SHAP

In [ ]:
```
import shap
X_train_CS = X_train_CS.astype(float)

explainer_cs = shap.Explainer(model_cs, X_train_CS)
shap_values_cs = explainer_cs(X_train_CS)

# Global feature impact (summary plot)
shap.summary_plot(shap_values_cs, X_train_CS)
```

In [ ]:
```
X_train_lc_halc = X_train_lc_halc.astype(float)

explainer_lc = shap.Explainer(model_lc, X_train_lc_halc)
shap_values_lc = explainer_lc(X_train_lc_halc)

shap.summary_plot(shap_values_lc, X_train_lc_halc)
shap.summary_plot(shap_values_lc, X_train_lc_halc, plot_type="bar")
```

In [ ]:
```
explainer_halc = shap.Explainer(model_halc, X_train_lc_halc)
shap_values_halc = explainer_halc(X_train_lc_halc)

shap.summary_plot(shap_values_halc, X_train_lc_halc)
shap.summary_plot(shap_values_halc, X_train_lc_halc, plot_type="bar")
```