Auto Insurance Loss Analytics | Group 30

Jui Mathuria

Rohit Kumar

Prarthana Kakulte

Vibhav Bhat

Nandeesh H.S.

Executive Summary

In the competitive and risk-sensitive landscape of the insurance industry, accurately pricing premiums is essential to mitigate adverse selection and ensure profitability. This project addresses that challenge by developing robust predictive models to estimate three key insurance metrics:

- **Loss Cost (LC):** Expected claim cost per exposure unit
- **Historically Adjusted Loss Cost (HALC):** LC adjusted for historical claim behavior
- **Claim Status (CS):** Binary classification predicting whether a policyholder will file a claim

The training dataset comprised 37,451 policyholder records and the test set included 15,787 records, each with 28 features covering demographic, behavioral, insurance, and vehicle-related variables. To address data complexities, including a high proportion of zero claims and highly skewed loss distributions, targeted modeling strategies were applied to effectively capture both claim frequency and severity.

The modeling strategy began with the creation of a **stratification column** by combining information from Claim Status (CS), Loss Cost (LC), and Historically Adjusted Loss Cost (HALC). This composite variable enabled a **stratified sampling approach** that preserved the distributional characteristics of all three targets, ensuring consistent representation across training and validation sets.

Multiple modeling approaches were explored, including:

- **Individual Model Approach:** Separate models for CS (classification) and LC/HALC (regression) [2.1]
- **Two-Step Approach:** CS prediction followed by LC/HALC prediction only for likely claimants [2.2]
- **Three-Headed Neural Network:** Shared architecture with task-specific outputs for CS, LC & HALC [2.3]
- **Final Approach:** A two-step pipeline where out-of-fold CS probabilities (p_cs_1) were used as inputs to LC and HALC models, enabling them to learn loss patterns based on predicted claim likelihood [2.4]

Each modeling approach involved experimenting with a range of machine learning algorithms, including gradient boosting methods (XGBoost, LightGBM, CatBoost), logistic regression, and neural networks. For each model, feature selection was conducted using Recursive Feature Elimination with Cross-Validation (RFECV), and hyperparameter optimization was performed using Optuna, enabling efficient exploration of the search space. The best-performing models were selected based on cross-validated performance metrics tailored to each task.

The final approach adopted was a refined two-step pipeline with a key shift: instead of using binary CS predictions, the predicted probability of CS = 1 (denoted as p_cs_1) was introduced as an input feature in the regression models. This allowed LC and HALC models to learn that lower claim probabilities are generally associated with lower expected losses. To avoid information leakage, p_cs_1 was generated using out-of-fold predictions via cross-validation. This framework used a tuned CatBoost classifier to estimate claim probabilities and a LightGBM regressor with Tweedie loss to predict LC and HALC, offering strong predictive performance while maintaining robustness to skewed and zero-inflated distributions.

Model Performance (Selected Metrics):

- **Claim Status (CS): CatBoost Classifier - AUC** ≈ 0.822
- **Loss Cost (LC): Light GBM Regressor with Tweedie - RMSE** ≈ 835.63
- **HALC RMSE: Light GBM Regressor with Tweedie** ≈ 1659.27

These results demonstrate the effectiveness of a data-driven framework for insurance risk modeling, enabling accurate and equitable premium pricing. The selected models were compatible with SHAP-based interpretability, allowing for transparent analysis of key predictors. For Claim Status (CS), the most influential features included the policy tenure and contract start month while for LC & HALC, key drivers were probability of CS, years with the company, Cylinder Capacity. This combination of strong predictive performance and interpretability equips insurers with actionable insights for strategic pricing and risk assessment.

Review of Approaches

Data Stratification Methodology [1]

We implemented a stratification approach to ensure representative data partitioning. Our methodology involved creating a composite 'Strata' column by combining Claim Status (CS) with percentile-based binning of Loss Cost (LC) and Historically Adjusted Loss Cost (HALC) variables into four tiers each. This multidimensional stratification preserved the distribution of critical target variables across partitions. This column was used for data splitting and cross-validation splits.

Feature Selection Methodology

We implemented Recursive Feature Elimination with Cross-Validation (RFECV) to systematically identify the optimal feature subset for our models. This technique iteratively eliminated less impactful variables while evaluating performance through cross-validation, ensuring robust selection criteria. The resulting optimized feature set significantly reduced model complexity while maintaining predictive power, enhancing both computational efficiency and model interpretability. Our RFECV implementation provided a data-driven approach to dimensionality reduction that directly supported our modelling objectives.

Feature Engineering Implementation

Our feature engineering focused on creating predictive variables to boost model performance and interpretability. We included features like age_at_last_renewal, driving_experience, and vehicle_age to reflect demographic risk, road proficiency, and vehicle depreciation. Policy_tenure and first_policy_age captured customer loyalty and early insurance behavior, while claim_frequency_total normalized historical claims across time. Together, these features offered a well-rounded view of insurance risk while ensuring efficiency and business relevance.

Modelling Flow 1: Individual Model Approach [2.1]

In our first modelling approach, we adopted a modular strategy by building separate models for classification (Claim Status) and regression (Loss Cost and Historically Adjusted Loss Cost). After performing a combined stratified split (70% train, 30% test), each model underwent Recursive Feature Elimination with Cross-Validation (RFECV), hyperparameter tuning, and model training using cross-validation to select the optimal algorithm. These models were then evaluated on the hold-out test set using appropriate metrics—AUC for classification and MSE/RMSE for regression. Once performance was satisfactory, final predictions were generated using the unseen test.csv file. This method, implemented using XGBoost with a Tweedie distribution for regression, offered simplicity and clear task specialization. However, a key limitation was the lack of information sharing between tasks, potentially leading to inconsistent outputs across the models—an issue addressed in our subsequent integrated modelling approach.

Modelling Flow: 2-Step Model Approach [2.2]

The 2-Step model is like the above except here we first predict CS, then predict LC and HALC only for predicted claims (CS=1). This sequential approach first uses a classification model to determine whether a policy will have a claim (CS=1) or not (CS=0). Then, regression models for Loss Cost (LC) and Historically Adjusted Loss Cost (HALC) are trained and applied only to those policies predicted to have claims. This method focuses the regression task exclusively on actual claim events, reducing noise from no-claim policies and potentially improving prediction accuracy by addressing the zero-inflated nature of insurance claim data.

3-Headed Neural Network – Innovative Method [2.3]

The third model used a shared neural network with three output heads—CS (binary cross-entropy), LC, and HALC (custom Tweedie loss). While this design theoretically offers enhanced feature sharing, stronger generalization capabilities, and improved interpretability through its unified structure, the observed performance metrics did not meet our expectations. Despite the elegant architectural approach and potential parameter efficiency, experimental results revealed that the multi-head model underperformed compared to our separate model implementations. The additional complexity in hyperparameter tuning and balancing multiple loss functions likely contributed to these suboptimal outcomes, suggesting that simpler, dedicated models may be more effective for our specific prediction requirements.

Final Approach

Two-Stage Modeling Architecture [2.4]

To effectively model insurance claim outcomes and associated losses, we implemented a structured two-stage modeling pipeline that reflects the real-world underwriting process—first assessing claim risk, then estimating financial exposure.

In the first stage, a binary classification model was trained to predict CS (Claim Status), indicating whether a policyholder would file a claim. The model produced a probability score, *p_cs_1,* representing the likelihood of a claim. We utilized a CatBoostClassifier due to its strong handling of categorical features and robustness to class imbalance, which is critical given that only **~11%** of records correspond to claims.

In the second stage, the predicted p_cs_1 was used as an input feature in two LightGBM regression models:

- One to predict Loss Cost (LC), and
- Estimate Historically Adjusted Loss Cost (HALC), defined as $LC \times X.18$.

Incorporating *p_cs_1* enables the regression models to condition loss estimates on the probability of a claim, allowing for a more risk-sensitive and business-aligned prediction strategy.

Model Training and Validation

To ensure generalizability, we applied **3-fold cross-validation** during model development. For the classification model, we evaluated **AUC** to measure discriminatory performance. For both regression models, we used RMSE on out-of-fold predictions to quantify prediction errors.

Further, we conducted **feature selection and hyperparameter optimization** using **Optuna**, a Bayesian optimization framework known for its ability to efficiently navigate large search spaces. This approach enabled systematic tuning of model parameters, maximizing performance while controlling overfitting.

Model Performance

The finalized two-stage pipeline achieved the following performance on held-out validation data:

- **AUC (Claim Status)** ≈ 82.22
- **RMSE (Loss Cost)** ≈ 835.63
- **RMSE (HALC)** ≈ 1659.27

These results indicate strong classification performance and reliable loss estimation, especially considering the skewed and zero-inflated nature of the target variables.

Advantages, Limitations, and Robustness Analysis

Advantages:

- Interpretability: Aligns closely with the domain logic of insurance—first assess risk, then estimate cost.
- Modularity: Allows independent tuning and improvement of classification and regression components.
- Risk-Aware Prediction: Incorporates claim probability (p_cs_1) directly into cost estimation, improving realism.

Limitations:

- Error Propagation: Inaccuracies in p_cs_1 may carry forward into LC and HALC predictions.
- Added Complexity: The two-stage pipeline introduces inter-model dependencies and requires careful orchestration.

To assess the robustness of our approach, we varied the threshold applied to p_cs_1 and measured the resulting impact on HALC RMSE. As shown in the figure below, RMSE remained largely stable across a wide range of thresholds. This indicates that the regression models are not overly sensitive to the precise calibration of the classifier and validates the reliability of our staged modeling framework.

SHAP & Interpretability

For Claim status, SHAP analysis identified policy_tenure, contract_start_month, and years_with_company as the top predictors [6.1]. Longer tenure and customer loyalty consistently reduced claim likelihood, as shown by their negative SHAP values. Policies starting later in the year also correlated with lower claim risk. The beeswarm plot [6.2] confirms that higher values of these features (in red) shift predictions left, indicating reduced probability of filing a claim. These insights align with business intuition and support data-driven premium adjustments.

For the LC model, SHAP analysis showed predicted claim status (p_cs_1) as the top driver, with high values significantly increasing loss cost. Years_with_company lowered predicted loss costs slightly, while vehicle age increased it [7.3]. The beeswarm plot confirms that predicted claims and older vehicles raise risk, emphasizing the value of claim likelihood and vehicle condition in cost estimation [7.4].

SHAP analysis for HALC identified predicted claim status (p_cs_1) as the most influential factor, with higher values strongly increasing HALC. Vehicle-related attributes like cylinder capacity, vehicle power, and vehicle weight followed, where higher values generally led to higher cost predictions [8.4]. The beeswarm plot confirms that these features have a strong positive impact when their values are high (in red), reinforcing the link between vehicle specifications and historical risk exposure [8.5].

Innovation: Autoencoder-Augmented Multi-Task Modeling [3]

To explore advanced modeling strategies, we implemented a three-part innovation framework combining representation learning, feature augmentation, and custom loss optimization. These efforts aimed to improve predictive accuracy while addressing the complex, imbalanced, and nonlinear nature of the insurance dataset.

Outlier-Aware Autoencoder Training

We trained an unsupervised autoencoder to learn compressed representations of the original input features. After training, we computed the reconstruction loss for each record and used it as a proxy for structural uniqueness or outlier behavior. Records with higher reconstruction loss were assigned greater sample weights in downstream models, effectively encouraging the supervised learning process to focus on harder-to-represent, potentially high-risk observations.

Bottleneck Feature Augmentation

The 16-dimensional output of the autoencoder's bottleneck layer was extracted and concatenated with the original 30 input features to enrich the model's input space. These latent features captured nonlinear relationships and interactions that may not be explicitly represented in the raw data. Including both raw and learned features allowed downstream models to leverage low-level and abstracted representations simultaneously.

Three-Headed Neural Network with Custom Loss Functions

We developed a multi-task neural network with three heads: one for binary classification of Claim Status (CS), and two for regression tasks predicting Loss Cost (LC) and Historically Adjusted Loss Cost (HALC). The CS head used binary cross-entropy loss, while LC and HALC were trained using a custom Tweedie loss function suited for zero-inflated, skewed distributions. To balance these objectives, we applied weighted loss functions and experimented with different combinations to optimize overall model performance.

Despite the theoretical strength of this approach, the results were suboptimal due to high computational demands, limited tuning, and reduced interpretability of the learned features. We ultimately discontinued this direction in favor of more transparent and scalable methods, but the process offered valuable insights for future deep learning-based actuarial modeling.

Conclusion

Model Performance Summary

Our modeling efforts have produced strong results across all tasks. CatBoost with Auto Encoder, RFECV and optuna achieved an RMSE of 835.63 for Loss Cost (LC) and 1659.27 for Historically Adjusted Loss Cost (HALC). For Claim Status classification, we got an AUC of 0.822 with high accuracy and precision.

Business Insights

- **Accuracy vs. Interpretability:** In insurance, interpretability outweighs raw accuracy. Explaining premium calculations is crucial for regulatory compliance, customer trust and informed business decisions.
- **Key Challenges:** We addressed imbalanced data (few policies with claims), mixed data types requiring specialized preprocessing and zero-inflation through Tweedie models designed for insurance loss distributions.

Model Applicability Limitations

This model cannot be directly applied to a case like life insurance, which fundamentally differs in its risk assessment and predictive modeling approach. Life insurance follows age-dependent mortality tables with fixed payouts, in contrast to our auto insurance model's approach of handling random, skewed losses with dynamic probabilistic distributions.

The fundamental divergence lies in the underlying risk characteristics and predictive methodologies. Life insurance fundamentally relies on comprehensive health history, genetic predispositions, and lifestyle factors that are entirely unrelated to our vehicle-based predictors. While our model excels in capturing the complex, short-term risk landscape of auto insurance, characterized by frequent, variable claim patterns, life insurance modeling requires a fundamentally different analytical framework.

Strategic Features

We prioritized business-relevant predictors: Years with Company, Total Policies, Max Policies, Non-Pay Cancellations and Net Premium.

The philosophical underpinning of our approach is simple yet powerful: true predictive effectiveness emerges from the sophisticated integration of strategic predictors, each carefully chosen to reveal a different facet of customer behavior and risk potential. This methodology represents a paradigm shift in insurance analytics, transforming data points into a comprehensive narrative of risk, customer engagement, and strategic opportunity.

Future Directions

This project presented a comprehensive approach to insurance loss modeling by integrating advanced machine learning techniques with domain-specific considerations. While models such as CatBoost and LightGBM with Tweedie loss offered strong predictive performance, it's important to recognize that insurance loss prediction—especially under highly skewed, zero-inflated distributions like the Tweedie—is inherently complex and cannot be fully captured by algorithms alone. Effective pricing strategies in such contexts require not only robust modeling but also deep domain expertise to contextualize predictions, interpret risk drivers, and align outcomes with regulatory and business constraints.

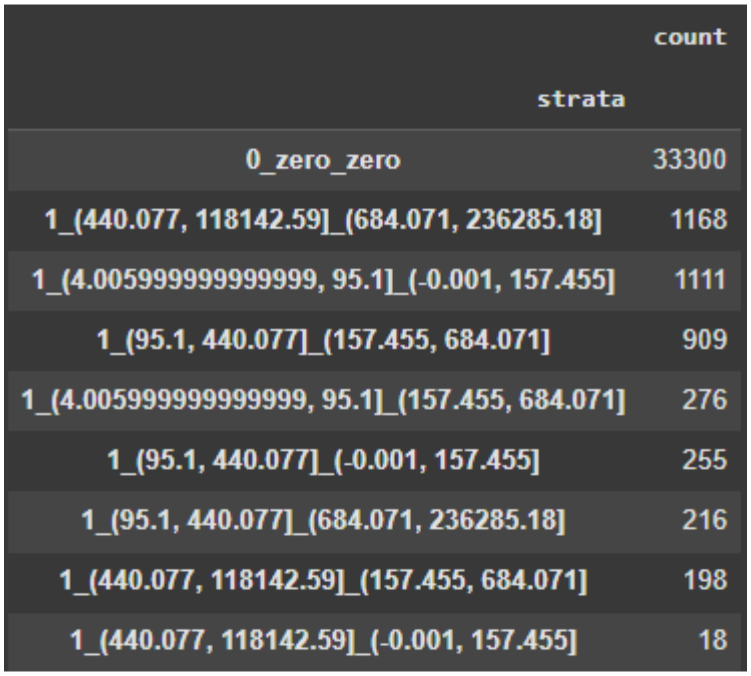
Our framework strikes this balance by combining accurate forecasting with interpretability and strategic relevance. It minimizes adverse selection, supports data-driven premium pricing, and delivers meaningful business insights. Though challenges remain, particularly in capturing rare but high-impact claims, the methodology we have developed establishes a scalable foundation for future work. With further integration of domain knowledge and operational constraints, this modeling pipeline can be extended and refined to support real-world underwriting and risk management processes across insurance domains.

Predictive Power = f (Advanced Analytics, Domain Expertise, Strategic Vision)

Appendix

Table of Contents:

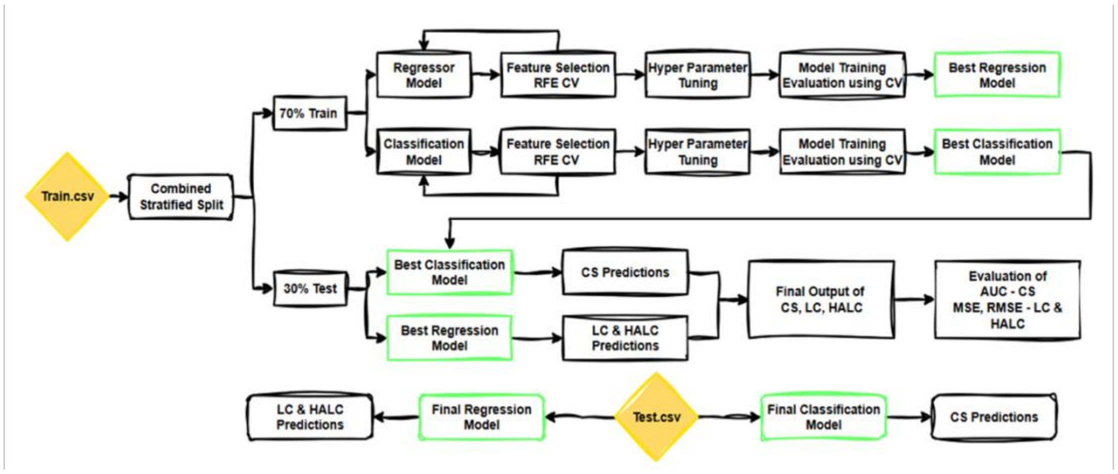
1. Strata Column Data Distribution



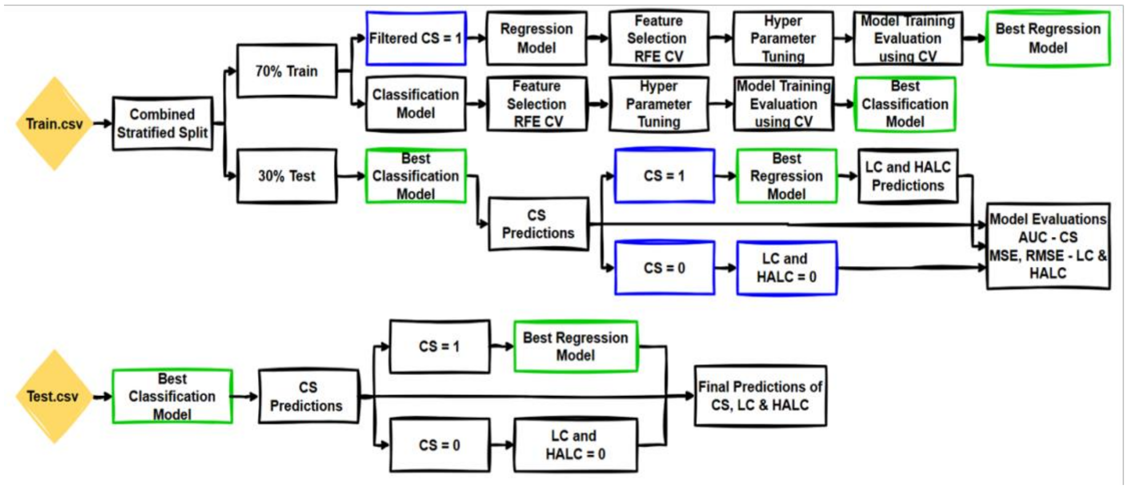
	count
strata	
0_zero_zero	33300
1_(440.077, 118142.59]_(684.071, 236285.18]	1168
1_(4.005999999999999, 95.1]_(-0.001, 157.455]	1111
1_(95.1, 440.077]_(157.455, 684.071]	909
1_(4.005999999999999, 95.1]_(157.455, 684.071]	276
1_(95.1, 440.077]_(-0.001, 157.455]	255
1_(95.1, 440.077]_(684.071, 236285.18]	216
1_(440.077, 118142.59]_(157.455, 684.071]	198
1_(440.077, 118142.59]_(-0.001, 157.455]	18

2. Methodology Flows

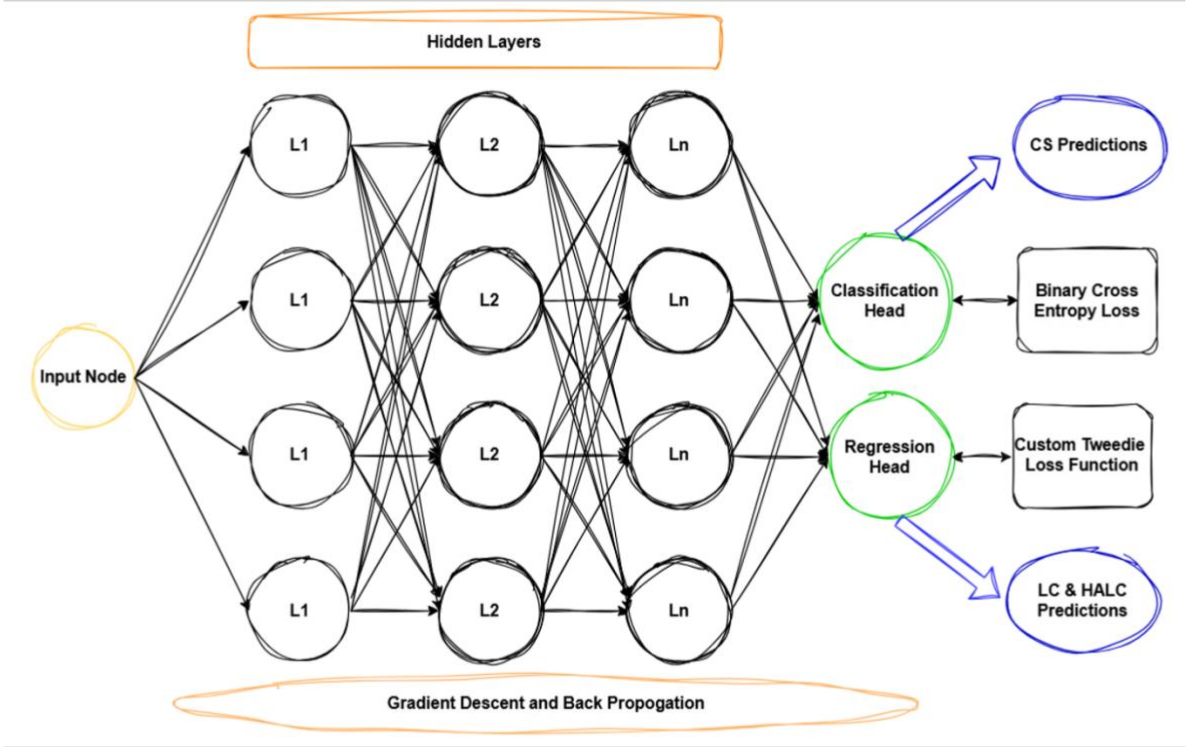
2.1. Individual Methodology



2.2. Two-Step Methodology



2.3 Three Headed Neural Network

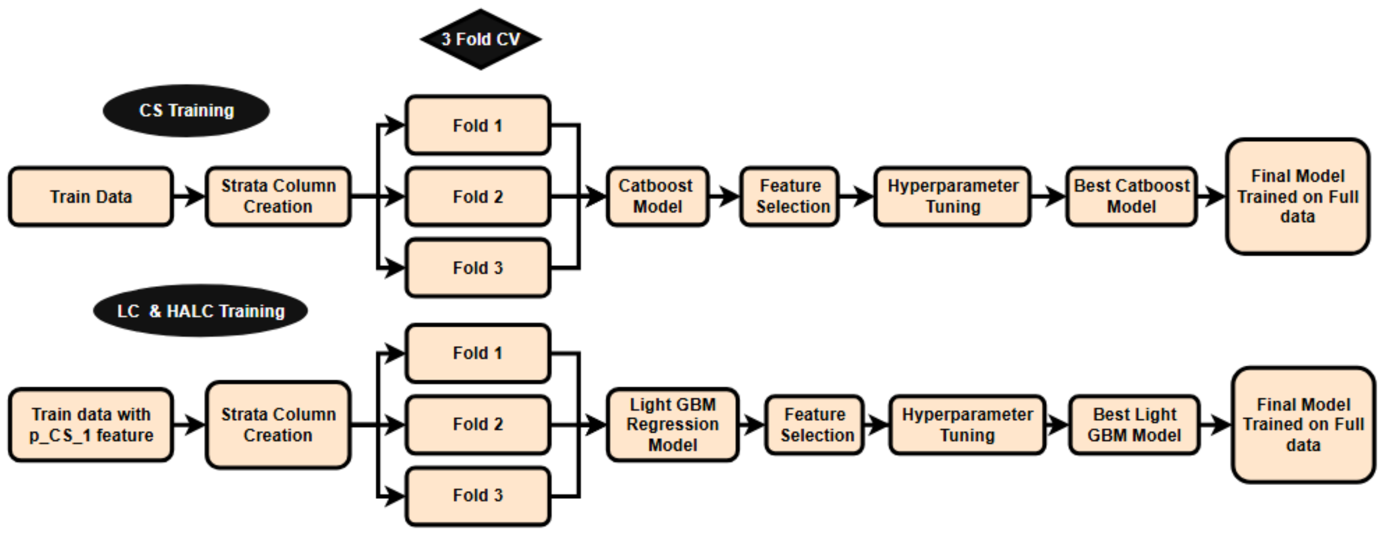


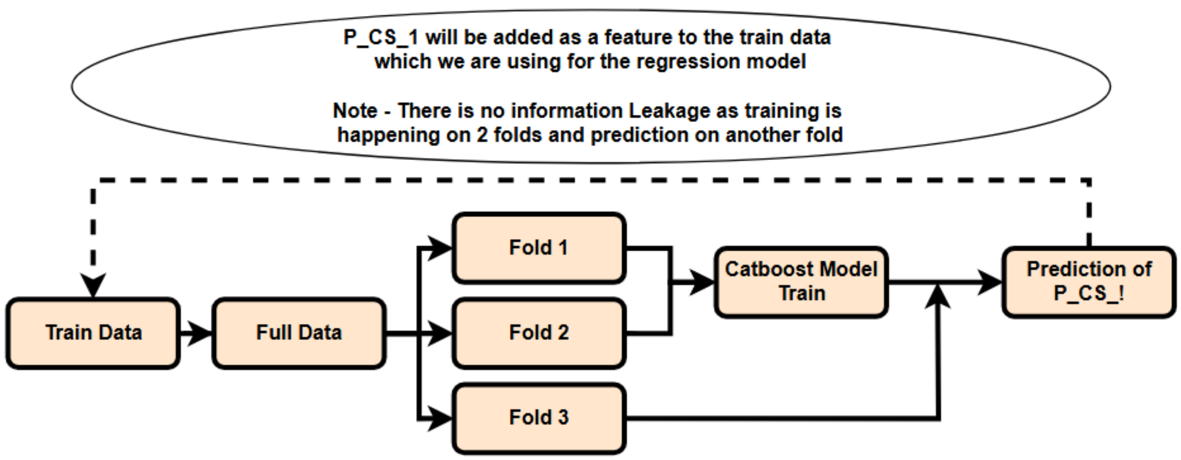
2.4 Final Methodology

Training

In our final methodology, we first stratify the training data and perform 3-fold cross-validation using a CatBoost classification model. We tune the hyperparameters to obtain the best-performing CatBoost model. This model is then retrained on the entire training dataset to generate probability scores for the class **CS = 1**. These probability scores are added as a new feature to the training data.

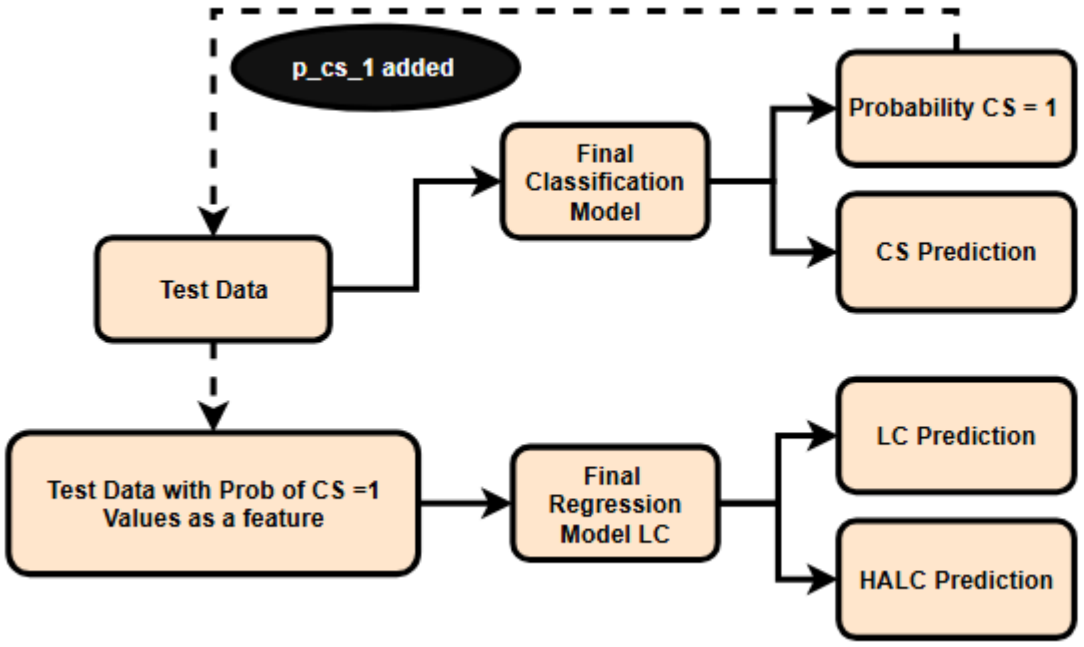
Subsequently, we train separate LightGBM regression models — one for predicting LC and another for HALC — using the enriched dataset (which now includes the CatBoost probability feature). We perform hyperparameter tuning for each regression task to obtain the best LightGBM models for LC and HALC, respectively.




P_CS_1 will be added as a feature to the train data which we are using for the regression model
Note - There is no information Leakage as training is happening on 2 folds and prediction on another fold
Train Data
Full Data
Fold 1
Fold 2
Fold 3
Catboost Model Train
Prediction of P_CS_!

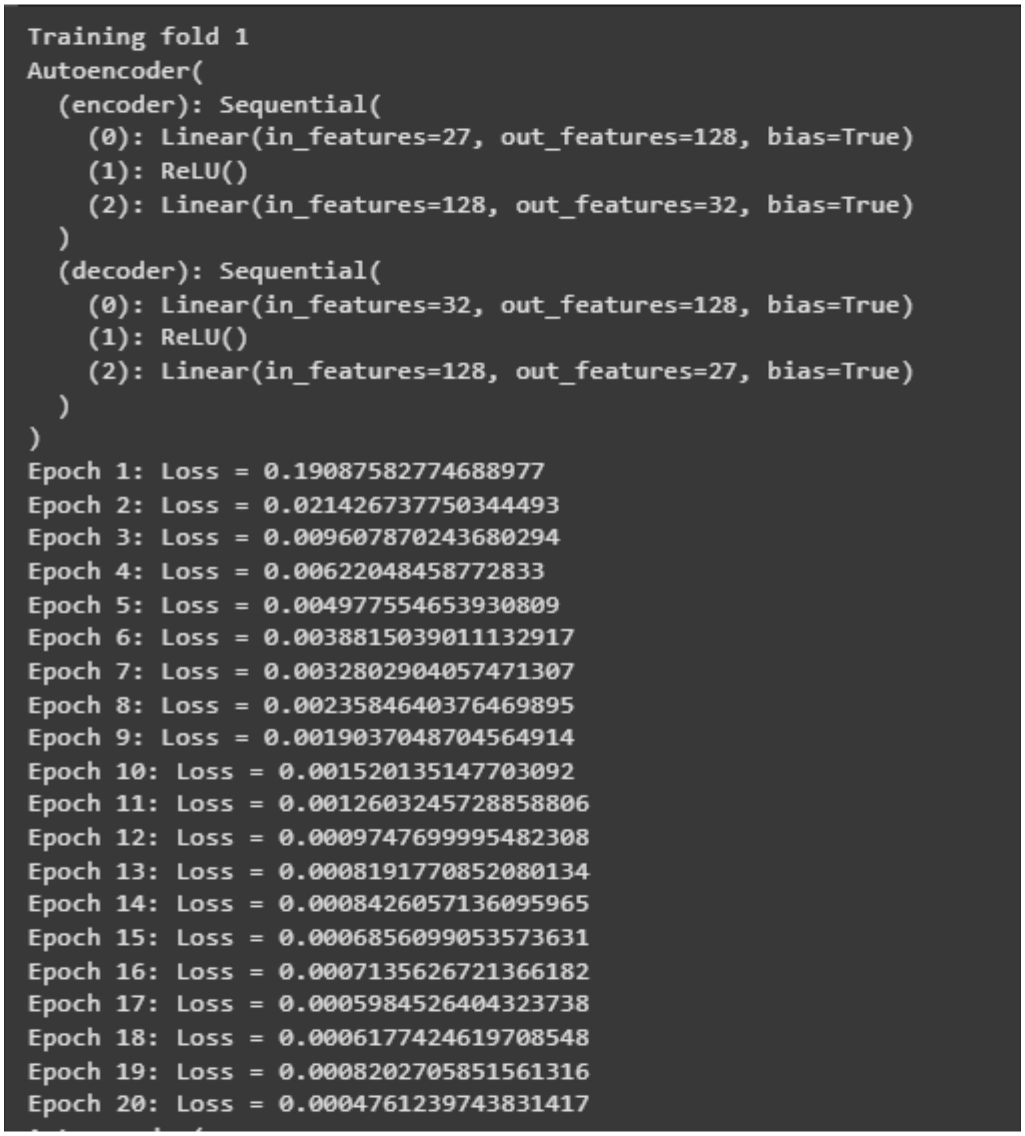
Prediction

On our test data, we run the above obtained classification model, to predict the probability of Claim Status being 1, and this extra column is added to our test data and we pass it through the regression models for LC and HALC, to obtain the corresponding predictions.


p_cs_1 added
Probability CS = 1
Final Classification Model
CS Prediction
Test Data
LC Prediction
Test Data with Prob of CS =1 Values as a feature
Final Regression Model LC
HALC Prediction

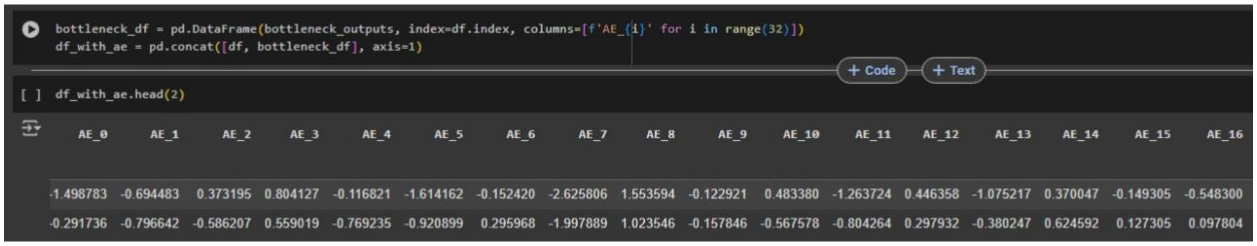
3. Innovation

3.1. Autoencoder Losses



```
Training fold 1
Autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=27, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=32, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=32, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=27, bias=True)
  )
)
Epoch 1: Loss = 0.19087582774688977
Epoch 2: Loss = 0.021426737750344493
Epoch 3: Loss = 0.009607870243680294
Epoch 4: Loss = 0.00622048458772833
Epoch 5: Loss = 0.004977554653930809
Epoch 6: Loss = 0.0038815039011132917
Epoch 7: Loss = 0.0032802904057471307
Epoch 8: Loss = 0.0023584640376469895
Epoch 9: Loss = 0.0019037048704564914
Epoch 10: Loss = 0.001520135147703092
Epoch 11: Loss = 0.0012603245728858806
Epoch 12: Loss = 0.0009747699995482308
Epoch 13: Loss = 0.0008191770852080134
Epoch 14: Loss = 0.0008426057136095965
Epoch 15: Loss = 0.0006856099053573631
Epoch 16: Loss = 0.0007135626721366182
Epoch 17: Loss = 0.0005984526404323738
Epoch 18: Loss = 0.0006177424619708548
Epoch 19: Loss = 0.0008202705851561316
Epoch 20: Loss = 0.00047612397438314l7
```

3.2. Extracting 16 features from bottleneck

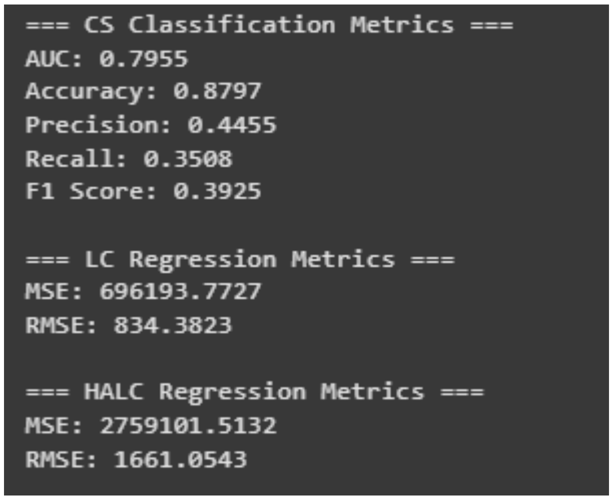


```
bottleneck_df = pd.DataFrame(bottleneck_outputs, index=df.index, columns=[f'AE_{i}' for i in range(32)])
df_with_ae = pd.concat([df, bottleneck_df], axis=1)
```

```
[ ] df_with_ae.head(2)
```

AE_0	AE_1	AE_2	AE_3	AE_4	AE_5	AE_6	AE_7	AE_8	AE_9	AE_10	AE_11	AE_12	AE_13	AE_14	AE_15	AE_16
-1.498783	-0.694483	0.373195	0.804127	-0.116821	-1.614162	-0.152420	-2.625806	1.553594	-0.122921	0.483380	-1.263724	0.446358	-1.075217	0.370047	-0.149305	-0.548300
-0.291736	-0.796642	-0.586207	0.559019	-0.769235	-0.920899	0.295968	-1.997889	1.023546	-0.157846	-0.567578	-0.804264	0.297932	-0.380247	0.624592	0.127305	0.097804

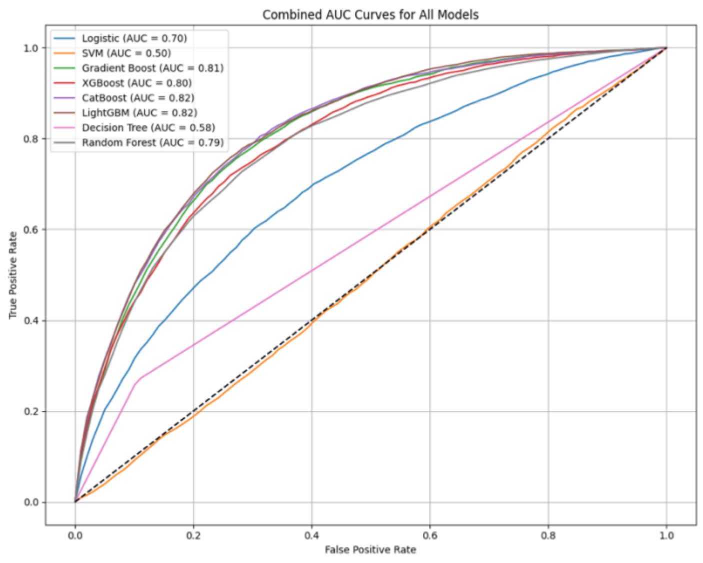
3.3. Three-Head NN Model

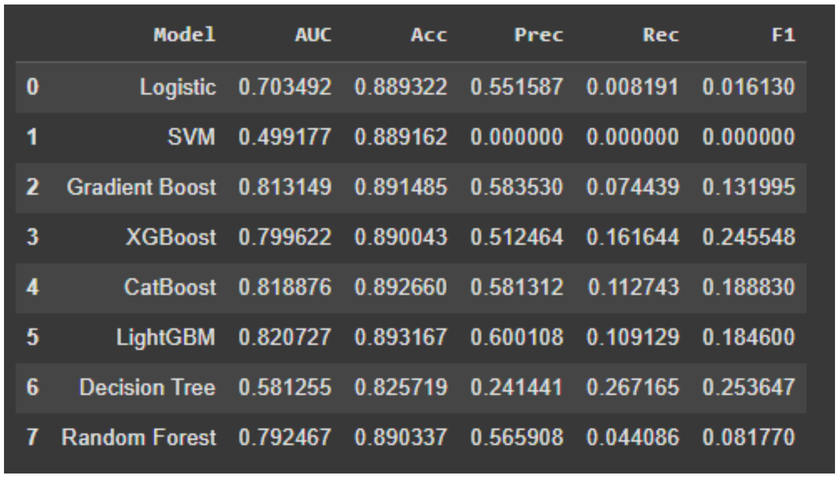


```
=== CS Classification Metrics ===
AUC: 0.7955
Accuracy: 0.8797
Precision: 0.4455
Recall: 0.3508
F1 Score: 0.3925

=== LC Regression Metrics ===
MSE: 696193.7727
RMSE: 834.3823

=== HALC Regression Metrics ===
MSE: 2759101.5132
RMSE: 1661.0543
```

4. Initial Classification Models

4.1. ROC-AUC Curve



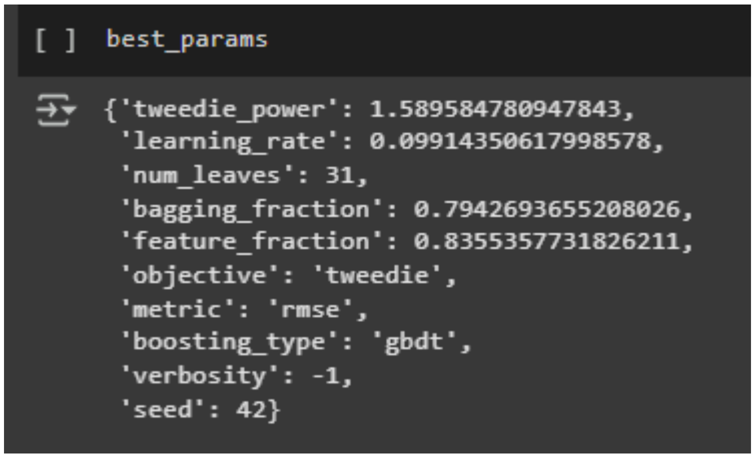


	Model	AUC	Acc	Prec	Rec	F1
0	Logistic	0.703492	0.889322	0.551587	0.008191	0.016130
1	SVM	0.499177	0.889162	0.000000	0.000000	0.000000
2	Gradient Boost	0.813149	0.891485	0.583530	0.074439	0.131995
3	XGBoost	0.799622	0.890043	0.512464	0.161644	0.245548
4	CatBoost	0.818876	0.892660	0.581312	0.112743	0.188830
5	LightGBM	0.820727	0.893167	0.600108	0.109129	0.184600
6	Decision Tree	0.581255	0.825719	0.241441	0.267165	0.253647
7	Random Forest	0.792467	0.890337	0.565908	0.044086	0.081770

5. Final CATBoost Model Reason

5.1. CATBoost Hyperparameters with Optuna

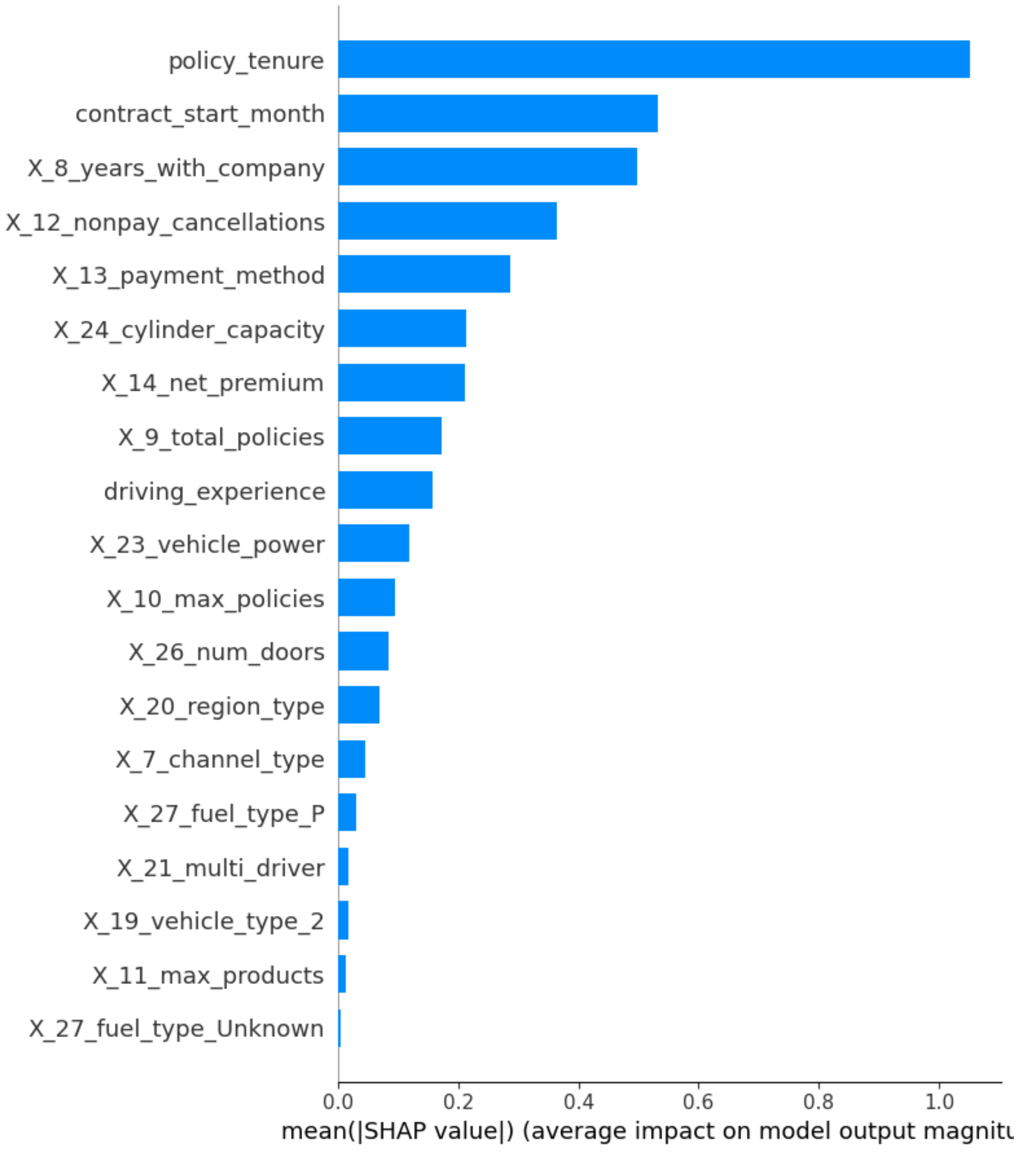
{'iterations': 890, 'depth': 5, 'learning_rate': 0.06209161650306317, 'l2_leaf_reg': 2.7111989338188773, 'border_count': 97, 'random_strength': 8.173243980892623, 'bagging_temperature': 0.6666430773430065, 'scale_pos_weight': 7.590358421201677}



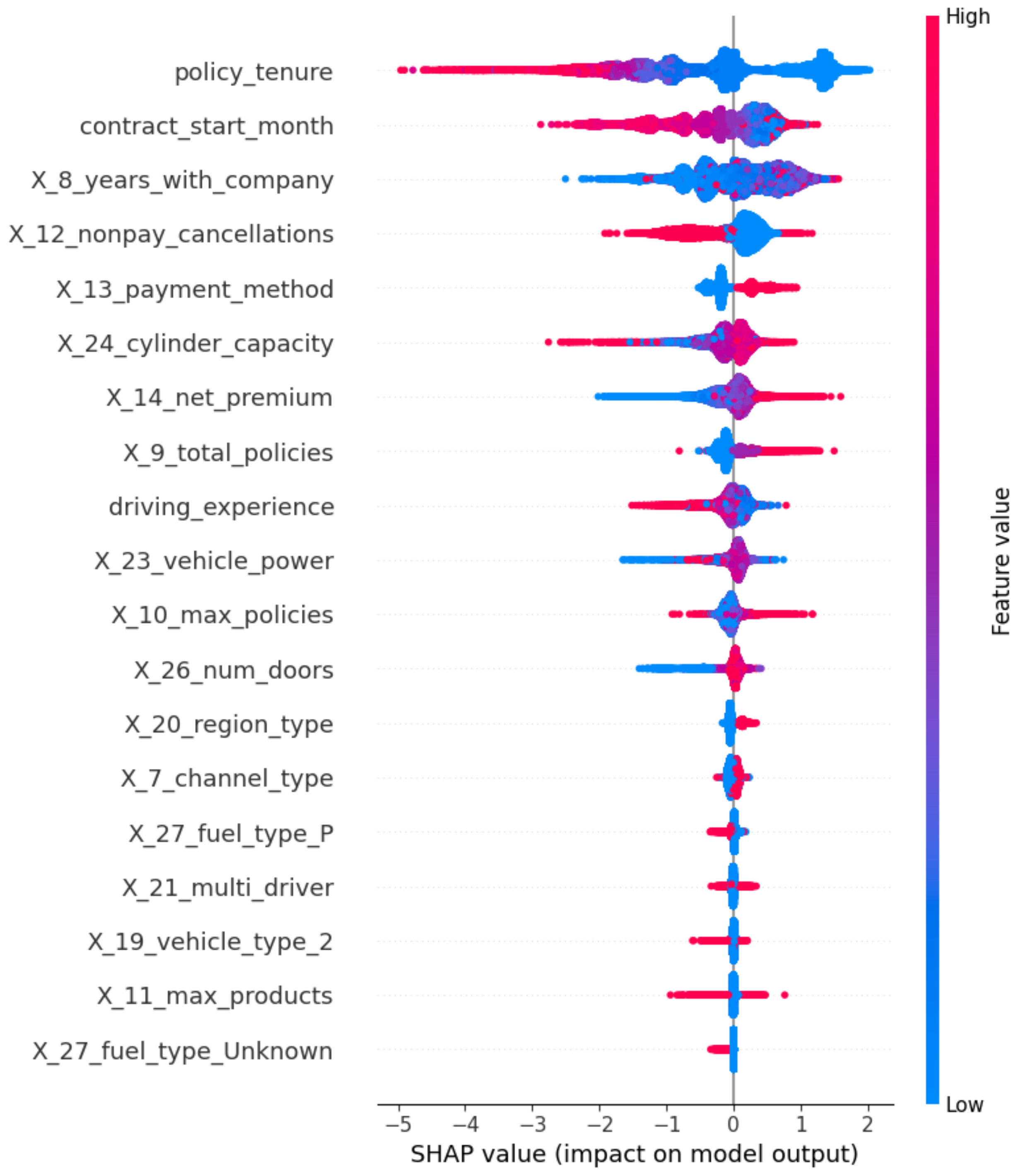
```
[ ] best_params

{'tweedie_power': 1.589584780947843,
 'learning_rate': 0.09914350617998578,
 'num_leaves': 31,
 'bagging_fraction': 0.7942693655208026,
 'feature_fraction': 0.8355357731826211,
 'objective': 'tweedie',
 'metric': 'rmse',
 'boosting_type': 'gbdt',
 'verbosity': -1,
 'seed': 42}
```

6. Claim Status Results

6.1. Feature Importance

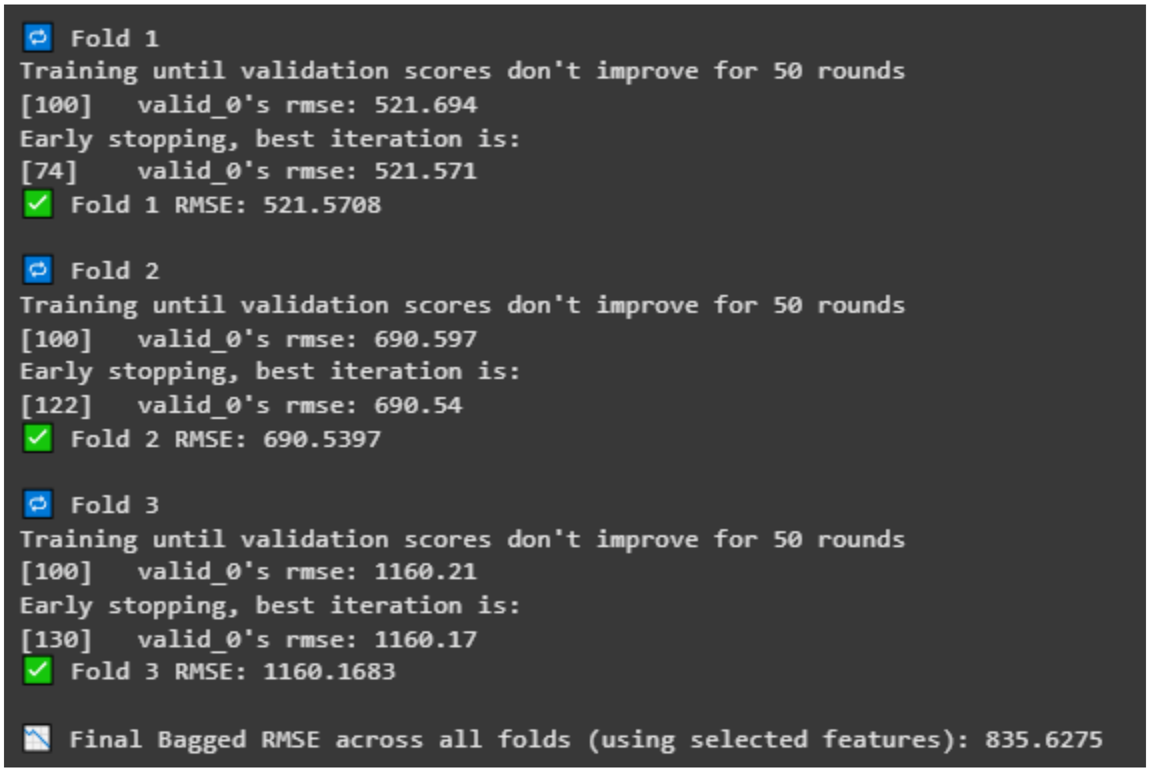


6.2. Beeswarm Plot



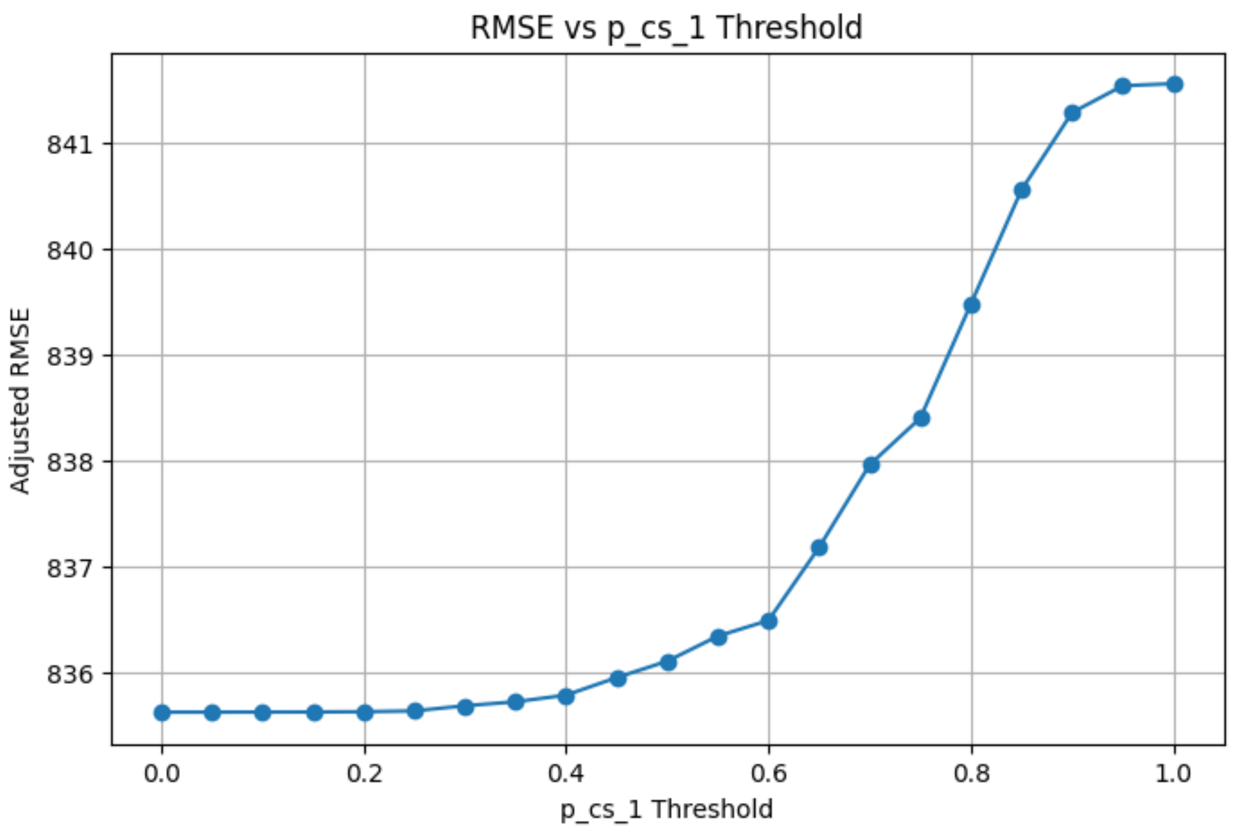
7. LC Regression Results

7.1. Best Model Result and Parameters

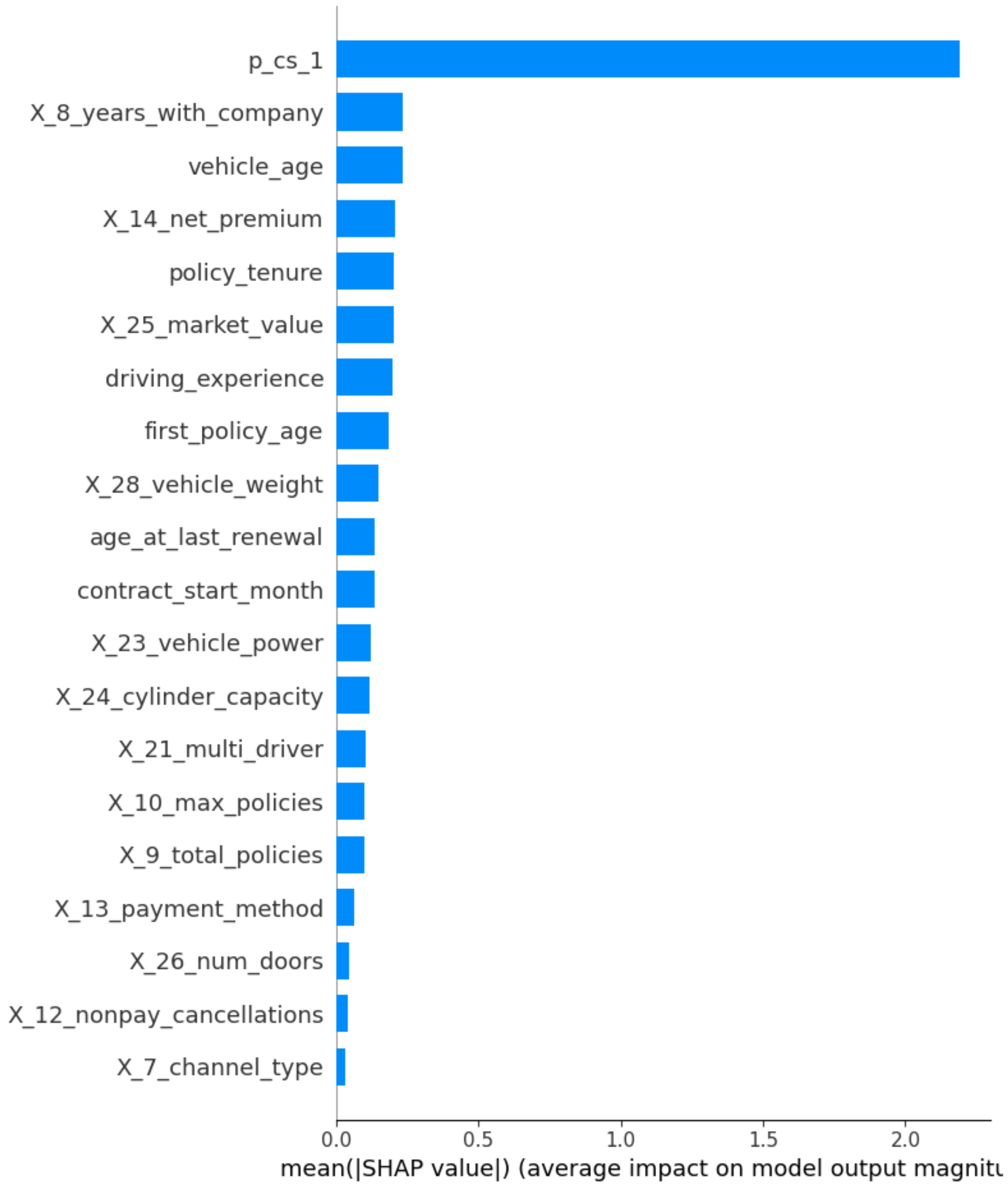


```
🔁 Fold 1
Training until validation scores don't improve for 50 rounds
[100]   valid_0's rmse: 521.694
Early stopping, best iteration is:
[74]    valid_0's rmse: 521.571
✅ Fold 1 RMSE: 521.5708

🔁 Fold 2
Training until validation scores don't improve for 50 rounds
[100]   valid_0's rmse: 690.597
Early stopping, best iteration is:
[122]   valid_0's rmse: 690.54
✅ Fold 2 RMSE: 690.5397

🔁 Fold 3
Training until validation scores don't improve for 50 rounds
[100]   valid_0's rmse: 1160.21
Early stopping, best iteration is:
[130]   valid_0's rmse: 1160.17
✅ Fold 3 RMSE: 1160.1683

📉 Final Bagged RMSE across all folds (using selected features): 835.6275
```

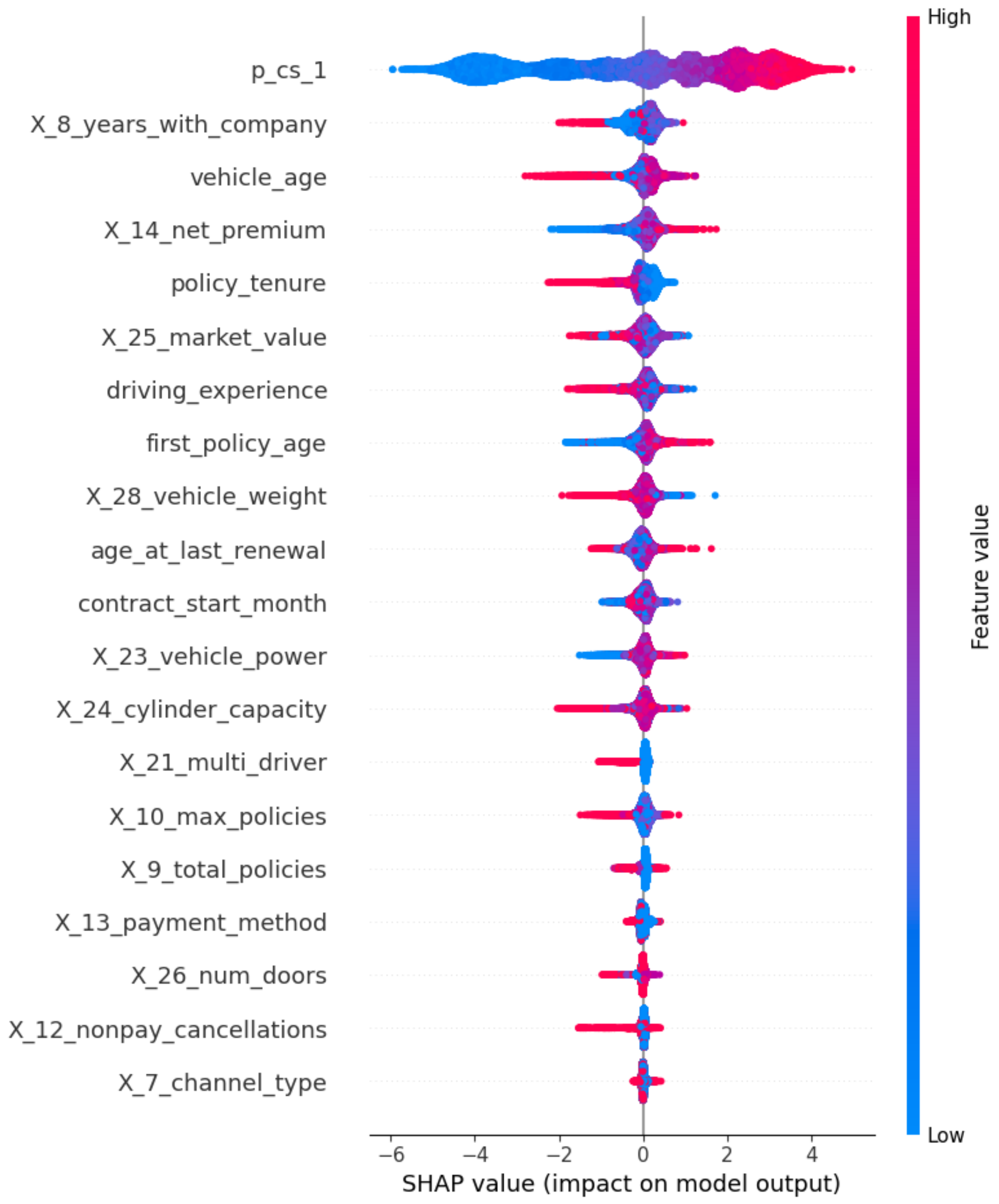
7.2. RMSE with various probability of CS_1 thresholds



7.3. Feature Importance

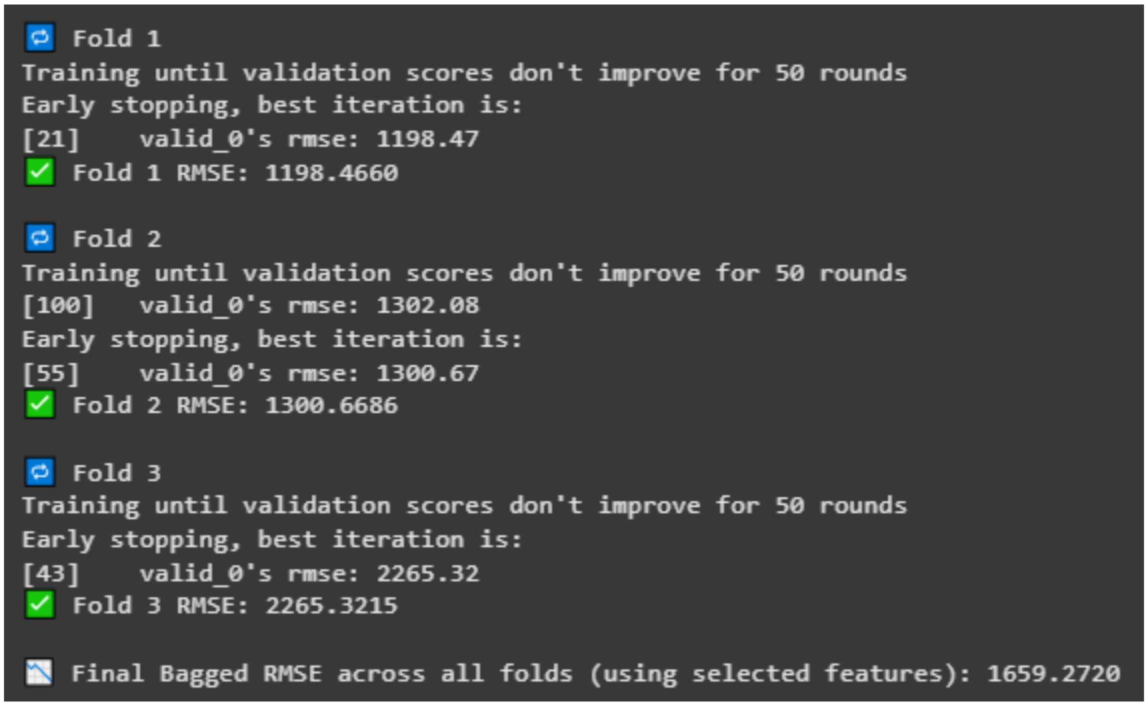


7.4. Beeswarm Plot



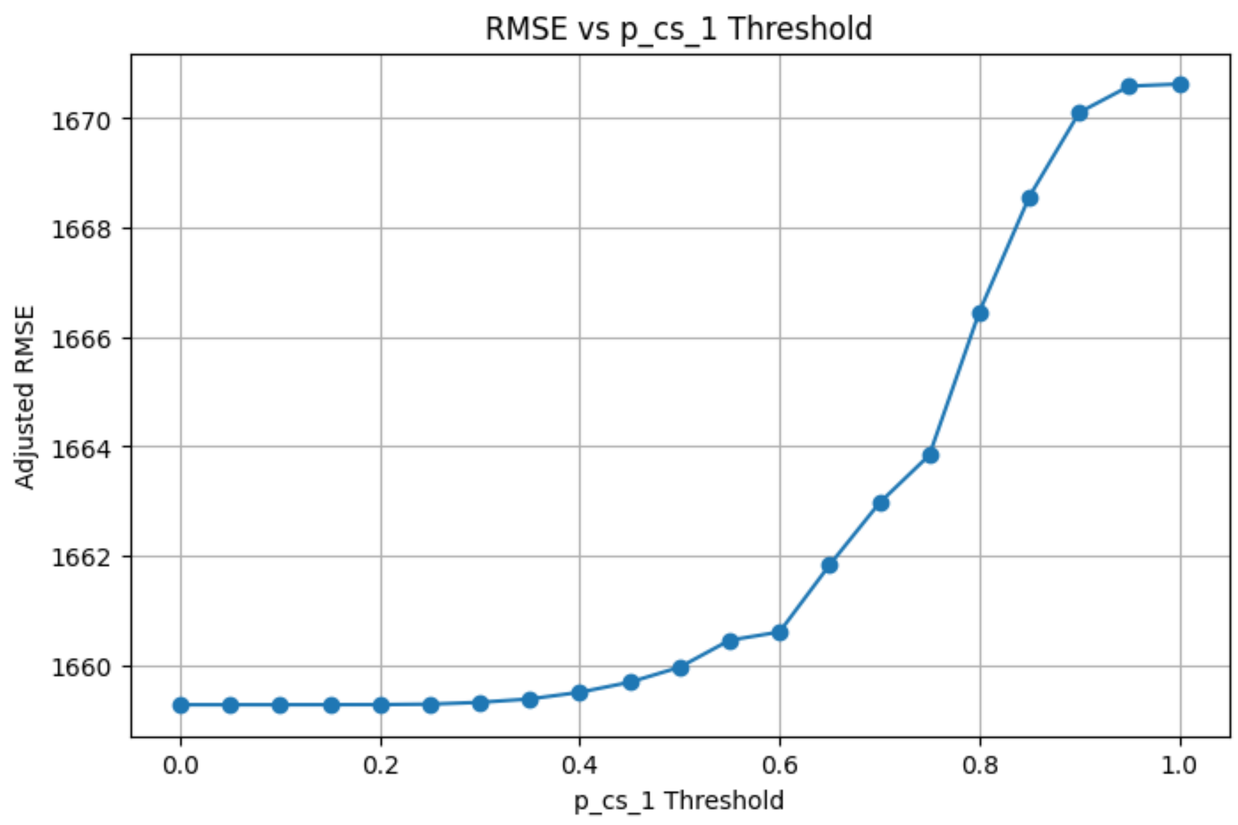
8. HALC Results

8.1. Model Results

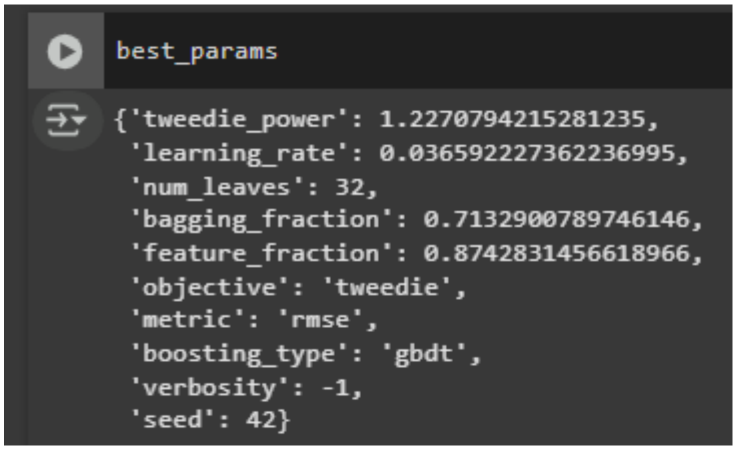


```
🔁 Fold 1
Training until validation scores don't improve for 50 rounds
Early stopping, best iteration is:
[21]    valid_0's rmse: 1198.47
✅ Fold 1 RMSE: 1198.4660

🔁 Fold 2
Training until validation scores don't improve for 50 rounds
[100]   valid_0's rmse: 1302.08
Early stopping, best iteration is:
[55]    valid_0's rmse: 1300.67
✅ Fold 2 RMSE: 1300.6686

🔁 Fold 3
Training until validation scores don't improve for 50 rounds
Early stopping, best iteration is:
[43]    valid_0's rmse: 2265.32
✅ Fold 3 RMSE: 2265.3215

📉 Final Bagged RMSE across all folds (using selected features): 1659.2720
```

8.2. HALC RMSE vs P_CS Threshold Analysis



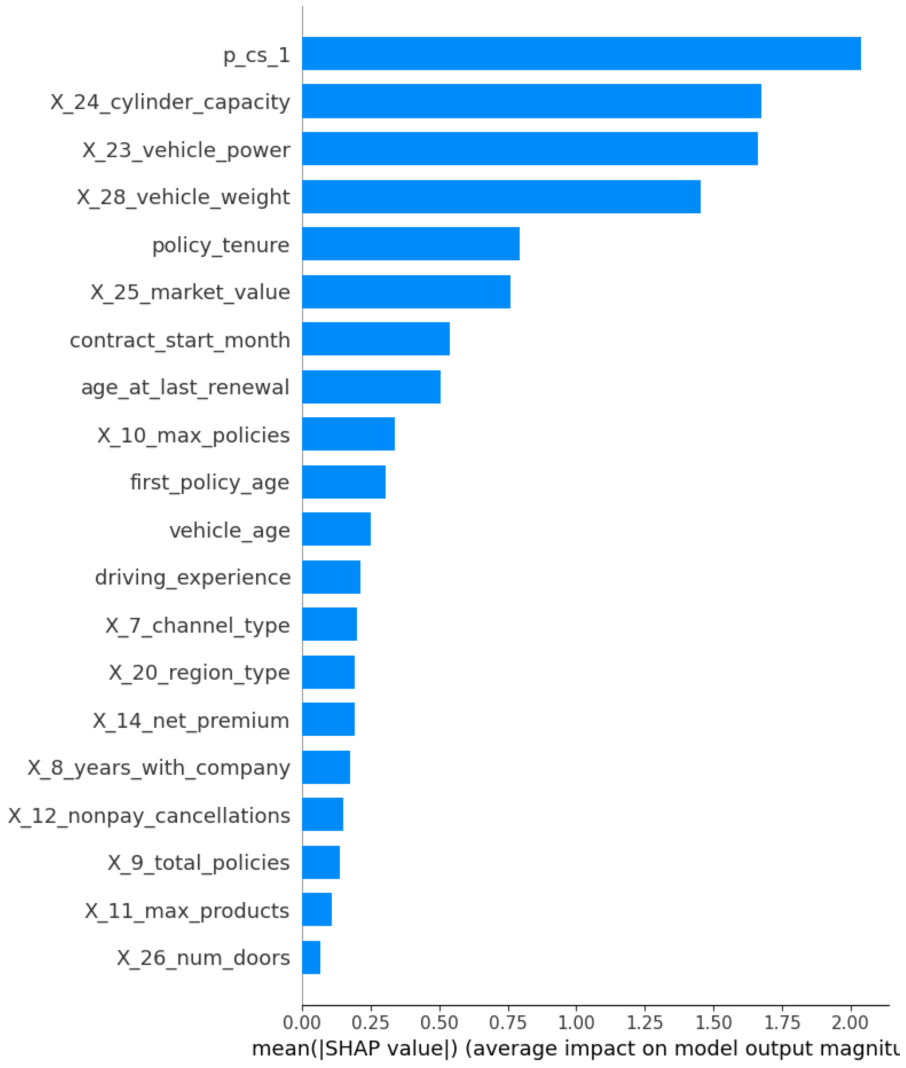
8.3. Best Parameter Results



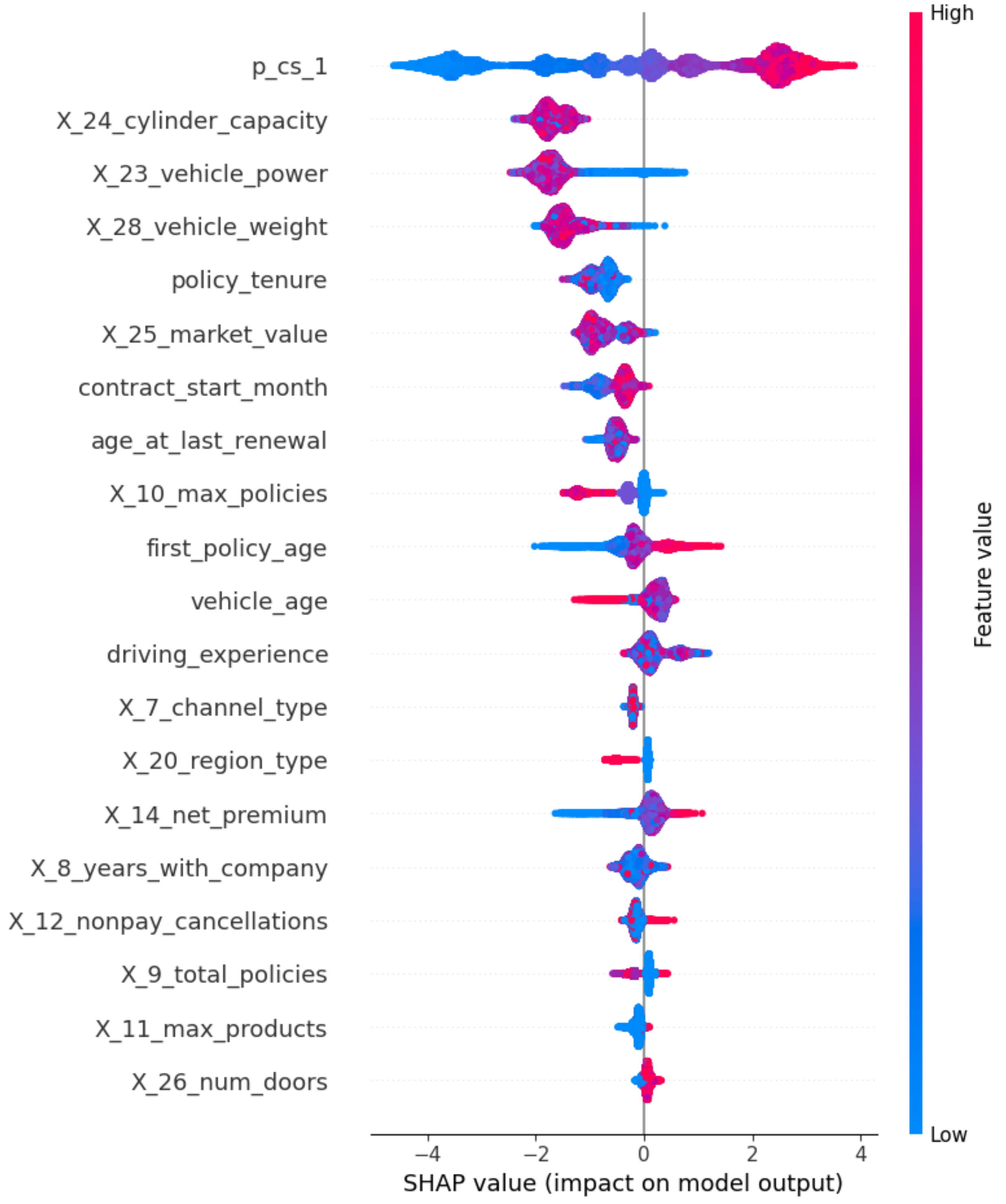
```
best_params
```

```
{'tweedie_power': 1.2270794215281235,
 'learning_rate': 0.036592227362236995,
 'num_leaves': 32,
 'bagging_fraction': 0.7132900789746146,
 'feature_fraction': 0.8742831456618966,
 'objective': 'tweedie',
 'metric': 'rmse',
 'boosting_type': 'gbdt',
 'verbosity': -1,
 'seed': 42}
```

8.4. Feature Importance

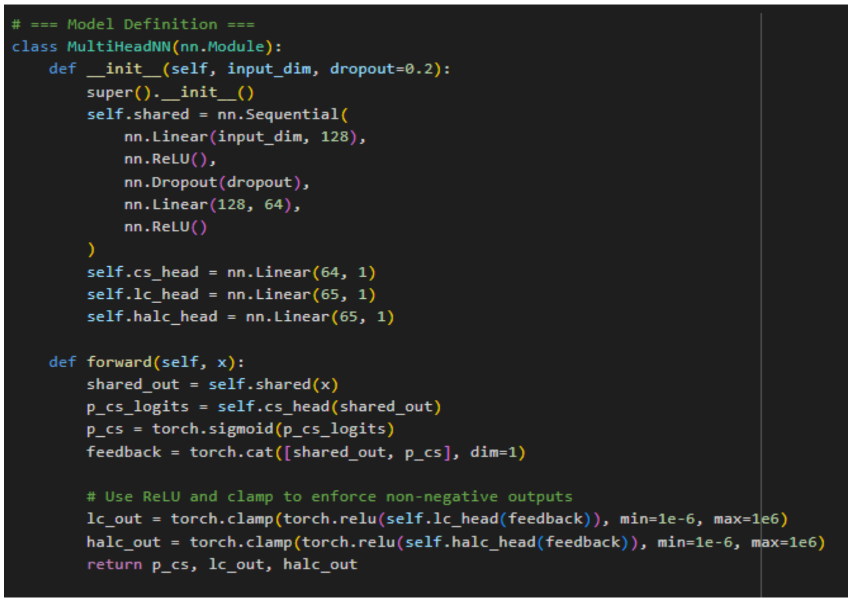


8.5. Beeswarm Plot

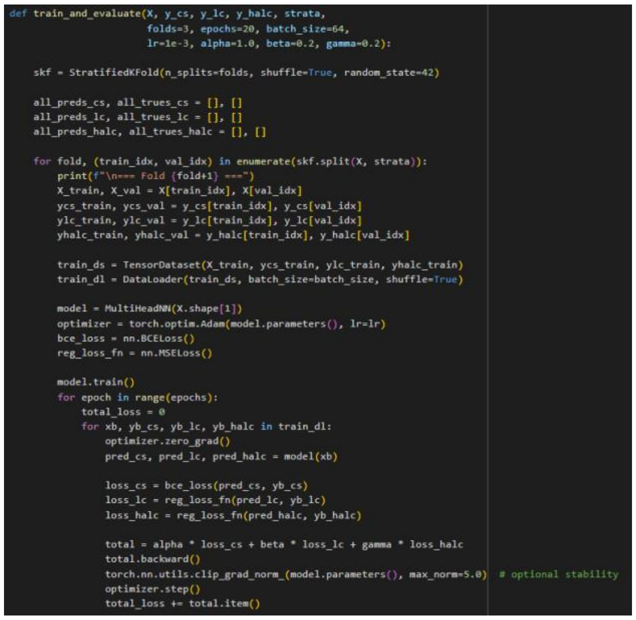


9. Code Snippets

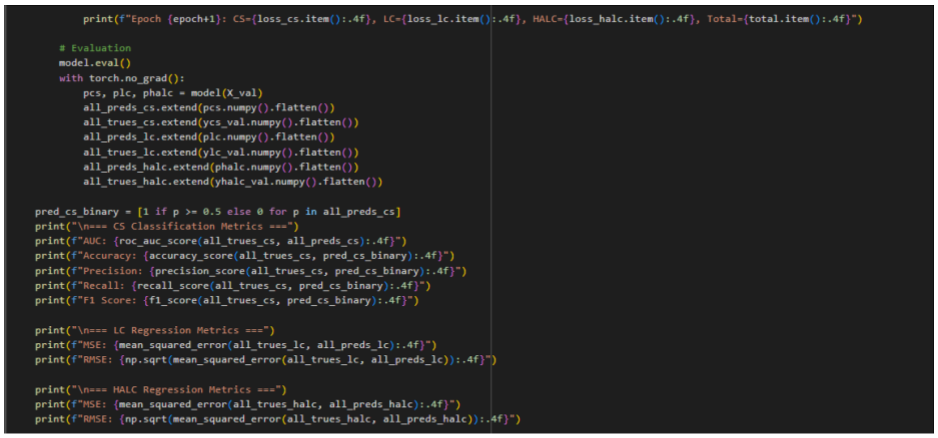
9.1. Three-Head NN Model



```python
# === Model Definition ===
class MultiHeadNN(nn.Module):
    def __init__(self, input_dim, dropout=0.2):
        super().__init__()
        self.shared = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(128, 64),
            nn.ReLU()
        )
        self.cs_head = nn.Linear(64, 1)
        self.lc_head = nn.Linear(65, 1)
        self.halc_head = nn.Linear(65, 1)

    def forward(self, x):
        shared_out = self.shared(x)
        p_cs_logits = self.cs_head(shared_out)
        p_cs = torch.sigmoid(p_cs_logits)
        feedback = torch.cat([shared_out, p_cs], dim=1)

        # Use ReLU and clamp to enforce non-negative outputs
        lc_out = torch.clamp(torch.relu(self.lc_head(feedback)), min=1e-6, max=1e6)
        halc_out = torch.clamp(torch.relu(self.halc_head(feedback)), min=1e-6, max=1e6)
        return p_cs, lc_out, halc_out
```



```python
def train_and_evaluate(X, y_cs, y_lc, y_halc, strata,
                       folds=3, epochs=20, batch_size=64,
                       lr=1e-3, alpha=1.0, beta=0.2, gamma=0.2):

    skf = StratifiedKFold(n_splits=folds, shuffle=True, random_state=42)

    all_preds_cs, all_trues_cs = [], []
    all_preds_lc, all_trues_lc = [], []
    all_preds_halc, all_trues_halc = [], []

    for fold, (train_idx, val_idx) in enumerate(skf.split(X, strata)):
        print(f"\n=== Fold {fold+1} ===")
        X_train, X_val = X[train_idx], X[val_idx]
        ycs_train, ycs_val = y_cs[train_idx], y_cs[val_idx]
        ylc_train, ylc_val = y_lc[train_idx], y_lc[val_idx]
        yhalc_train, yhalc_val = y_halc[train_idx], y_halc[val_idx]

        train_ds = TensorDataset(X_train, ycs_train, ylc_train, yhalc_train)
        train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True)

        model = MultiHeadNN(X.shape[1])
        optimizer = torch.optim.Adam(model.parameters(), lr=lr)
        bce_loss = nn.BCELoss()
        reg_loss_fn = nn.MSELoss()

        model.train()
        for epoch in range(epochs):
            total_loss = 0
            for xb, yb_cs, yb_lc, yb_halc in train_dl:
                optimizer.zero_grad()
                pred_cs, pred_lc, pred_halc = model(xb)

                loss_cs = bce_loss(pred_cs, yb_cs)
                loss_lc = reg_loss_fn(pred_lc, yb_lc)
                loss_halc = reg_loss_fn(pred_halc, yb_halc)

                total = alpha * loss_cs + beta * loss_lc + gamma * loss_halc
                total.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5.0)  # optional stability
                optimizer.step()
                total_loss += total.item()
```



```python
            print(f"Epoch {epoch+1}: CS={loss_cs.item():.4f}, LC={loss_lc.item():.4f}, HALC={loss_halc.item():.4f}, Total={total.item():.4f}")

        # Evaluation
        model.eval()
        with torch.no_grad():
            pcs, plc, phalc = model(X_val)
            all_preds_cs.extend(pcs.numpy().flatten())
            all_trues_cs.extend(ycs_val.numpy().flatten())
            all_preds_lc.extend(plc.numpy().flatten())
            all_trues_lc.extend(ylc_val.numpy().flatten())
            all_preds_halc.extend(phalc.numpy().flatten())
            all_trues_halc.extend(yhalc_val.numpy().flatten())

    pred_cs_binary = [1 if p >= 0.5 else 0 for p in all_preds_cs]
    print("\n=== CS Classification Metrics ===")
    print(f"AUC: {roc_auc_score(all_trues_cs, all_preds_cs):.4f}")
    print(f"Accuracy: {accuracy_score(all_trues_cs, pred_cs_binary):.4f}")
    print(f"Precision: {precision_score(all_trues_cs, pred_cs_binary):.4f}")
    print(f"Recall: {recall_score(all_trues_cs, pred_cs_binary):.4f}")
    print(f"F1 Score: {f1_score(all_trues_cs, pred_cs_binary):.4f}")

    print("\n=== LC Regression Metrics ===")
    print(f"MSE: {mean_squared_error(all_trues_lc, all_preds_lc):.4f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(all_trues_lc, all_preds_lc)):.4f}")

    print("\n=== HALC Regression Metrics ===")
    print(f"MSE: {mean_squared_error(all_trues_halc, all_preds_halc):.4f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(all_trues_halc, all_preds_halc)):.4f}")
```