

# Final Report for DSO 530 Final Project

## Insurance Loss Analytics Project

Team 27

Group Members

Name	Student ID
Adam Young	9229828947
Dominic He	5019549620
Emma Liu	5423029185
Hsuan-Jen Ko	3319837346
Min Yu Ho	5180695269
Zitong Wang	2397287871

Contact email: [aiyoung@usc.edu](mailto:aiyoung@usc.edu)

## **Executive Summary**

Recently, automobile companies have started using complex consumer data for predictive analytics signifying a transition from a one-size-fits-all approach to a more dynamic, data-driven strategy. In doing so, firms can predict risk with greater precision, customize pricing models further, anticipate and mitigate unforeseen fraud, and retain customers more effectively.

In this project, we used simulated data to predict 3 crucial metrics within the automobile insurance industry. Loss Cost per Exposure Unit (LC) is a continuous metric that estimates the cost of each claim, Historically Adjusted Loss Cost (HALC) takes into account the entire duration of the policy, and Claim Status (CS) is a binary prediction of whether or not a new policy holder will file a claim.

While exploring the data, we noticed some redundancies throughout predictive variables. For example, Vehicle Weight (X.28) and Number of Vehicle Doors (X.26) are indications of vehicle size, so using both variables could be unnecessary. This supports our prediction that some variables would be better predictors than others, as Market Value (X.25) should directly correlate with the cost of a claim, while Number of Vehicle Doors(X.26) is already redundant to another predictor.

Our data followed a Tweedie distribution, meaning that there were a large number of 0 or very small values with a heavy right skew of large, positive values. To handle this distribution, we would have to use Tweedie regression to predict the continuous variables (LC and HALC). We tried various models and tuned all of them with 5-fold Cross Validation. For LC, we found our tuned XGBoost Tweedie Model ( $\alpha = 1.0$ , variance power = 1.5) to perform best, as other models underperformed on high-loss claims. For HALC, our tuned LightGBM Tweedie ( $\alpha = 0.5$ , variance power = 1.1) outperformed XGBoost. Our most important predictive variables lined up with the idea that LC has a focus on short-term risk while HALC is affected by long term factors.

In predicting our binary variable CS, we compared ROC-AUC scores across different model's highest performing versions, tuned once again using 5-fold Cross Validation. In this case, Light GBM (learning rate = 0.05, max depth = -1, n = 100, leaves = 31) was our best performer, and LightGBM offered an excellent trade-off between training efficiency and prediction accuracy. Our most important predictive variables were Number of Policies Canceled (X.12) and Number of Policies Held During the Current Year (X.9)

Our innovation was to use Adversarial Validation to understand whether our test dataset could accurately validate our findings from our training sets. We found that our train and test sets share similar distributions, indicating that there is no evidence of a data leakage or distribution shift.

In the end, we are hesitant to use our models to directly predict LC, HALC, and CS for our insurance company, as our data is highly imbalanced and skewed. Although we attempted to account for this using Tweedie Regression, we felt unable to mitigate the effects of rare-event risks.

## Dataset Overview / EDA

### 1. Data Wrangling

To prepare the dataset for analysis, a subset of relevant columns was selected. Specifically, variables from X.7 to X.14 and X.19 to X.29 were retained. Additionally, three new features—**Age**, **License\_years**, and **Policy\_years**—were derived from date fields to capture the elapsed time since birth, license issuance, and policy start. Rows with missing values in X.27 were removed to ensure data integrity.

All continuous variables were treated as numerical, while X.7, X.13, X.19, X.20, X.21, and X.27 were classified as categorical. Importantly, all data points were retained—including potential outliers—to allow the model to learn the full pattern of the dataset without excluding extreme values.

### 2. Data Preprocessing

Prior to model training, appropriate preprocessing techniques were applied to both numerical and categorical variables. Numerical features were standardized using the `StandardScaler()` method to normalize their magnitudes, thereby improving model convergence and performance. Categorical features were transformed using one-hot encoding via `OneHotEncoder()`, which created dummy variables to represent distinct categories while allowing the model to handle previously unseen categories during inference. Additionally, we experimented with applying a `PowerTransformer()` to the numerical features; however, the use of `StandardScaler()` yielded better model performance and was therefore retained for the final preprocessing pipeline.

## Methods That We Tried

### 1. LC (Loss Cost per Exposure Unit)

After data cleaning and transformation, we add a new column called LC, where if X.16 is larger than 0, its LC will be  $X.15/X.16$ ; else, LC will be set to 0. We see that 89% of the data has  $LC = 0$  and is highly right-skewed, so we decided to use Tweedie regression (a model that fits for heavily zero-inflated and right-skewed data sets) to predict the value. Below are the methods that we've tried:

	Method	Reason
i	Tweedie Regressor	Tweedie regressor is suitable when the target is zero-inflated and continuous positive values
ii	Tweedie Regressor with log transformation for LC	Since LC is highly skewed with a long right tail, using log-transformation makes it easier to learn and stabilizes variance
iii	XGBoost	XGBoost is highly flexible, handles nonlinear patterns and interactions well, and naturally deals with missing data and class imbalance
iv	LightGBM	LightGBM is similar to XGBoost but faster and more efficient on large datasets (e.g., high-dimensional one-hot encoded data)
v	Two-Step Model	Since 89% of the data have LC as 0, we want to predict whether LC is 0, and then predict its value

### 2. HALC (Historically Adjusted Loss Cost)

HALC is defined as  $LC \times X.18$ , since 89% of the data's LC is 0, it also has a compound Poisson-Gamma behavior, which is mostly zeroes with rare but heavy-tailed positive values. Hence, we used the same methods as LC to predict our HALC

### 3. CS (Claim Status)

The claim status is the X.16 variable in our dataset, it is set to 0 if the policyholder did not file a claim throughout the year, else it will be 1. Since the target value is binary classification, below are the methods that we've tried:

	Method	Reason
i	XGBoost	XGBoost handles imbalanced, high-dimensional, and non-linear datasets very well, with so many policyholders not claiming a claim, it might predict the status well with its tree-based learner
ii	LightGBM	LightGBM is similar to XGBoost but faster and more efficient when the dataset has lots of one-hot encoded features. Since we do have some one-hot encoded features in the data, it will be worth trying
iii	Random Forest Classifier	Random forest is a solid, non-parametric baseline model that works well with noisy, structured data. It uses bagging to reduce the variance and can naturally handle imbalanced binary outcomes

## Final Approaches

### Task 1 Predicting LC & HALC

Our goal was to build regression models to predict LC and HALC, two continuous insurance loss metrics. We used linear and gradient-boosted Tweedie regressors due to their effectiveness on skewed, zero-inflated data. Models were tuned via 5-fold cross-validation and evaluated using Mean Squared Error.

#### Task 1.1 Predicting LC

We compared three approaches: linear Tweedie Regressor, LightGBM Tweedie, and XGBoost Tweedie. The linear Tweedie model (Appendix Fig 1.1) struggled with underfitting and high error, particularly in the zero-claim region. LightGBM (Appendix Fig 1.2) showed improvement, but still underperformed on high-loss predictions. The tuned XGBoost Tweedie model delivered the most accurate predictions across the full LC range. **XGBoost Tweedie** (see Fig 1.3) was the best-performing model for LC, with a ~60% reduction in MSE compared to LightGBM and strong fit across low- and high-loss values.

Model	Parameters	MSE
Tweedie Regressor	$\alpha = 1.0$ , variance power = 1.9	712,989
LightGBM Tweedie	$\alpha = 1.0$ , variance power = 1.5	549,996
<b>XGBoost Tweedie</b>	<b><math>\alpha = 1.0</math>, variance power = 1.5</b>	<b>217,392</b>

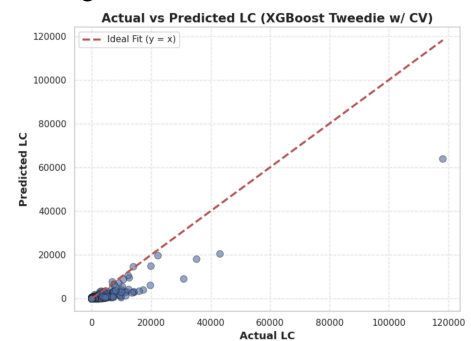


Figure 1.3: XGB Tweedie LC Model Result

#### 1.2 Predicting HALC

For HALC prediction, we tested tuned versions of XGBoost and LightGBM using Tweedie objective functions. While XGBoost produced acceptable performance, it was less consistent in the upper-loss spectrum (Appendix Fig 1.4). LightGBM Tweedie achieved the best overall fit, especially for both low and moderately high HALC values (see Fig 1.4).

Model	Parameters	MSE
XGBoost Tweedie	reg. $\alpha = 0.5$ , variance power = 1.1	653,948
<b>LightGBM Tweedie</b>	<b>reg. <math>\alpha = 0.01</math>, variance power = 1.5</b>	<b>360,881</b>

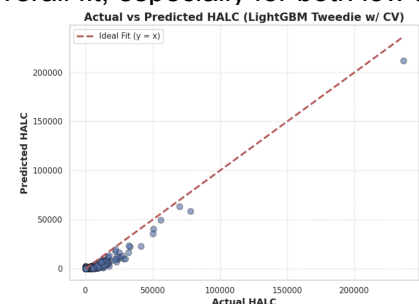


Figure 1.4: LGBM Tweedie HALC Model Result

### Task 2 Predicting CS

Our goal is to develop predictive models to classify insurance policies into two categories: claim filed vs. no claim filed. We employed supervised learning techniques using historical policyholder data, with a focus on optimizing model performance using ROC-AUC as the primary evaluation metric.

We evaluated three tree-based classification models: XGBoost, LightGBM, and Random Forest Classifier. Each model underwent hyperparameter tuning using 5-fold cross-validation with GridSearchCV, where model selection was based on ROC-AUC performance. And to ensure a fair evaluation, we used the full dataset for cross-validation (no train-test split). Below are the model performance and best parameters that we selected:

Model	Parameters	AUC
LightGBM	learning_rate = 0.05, max_depth = -1, n_estimators = 100, num_leaves = 31	0.7705
XGBoost	gamma = 0.1, learning_rate = 0.05, max_depth = 5, n_estimators = 200	0.7702
Random Forest Classifier	bootstrap = False, max_depth = 10, min_samples_leaf = 2, min_samples_split = 5, n_estimators = 200	0.7529

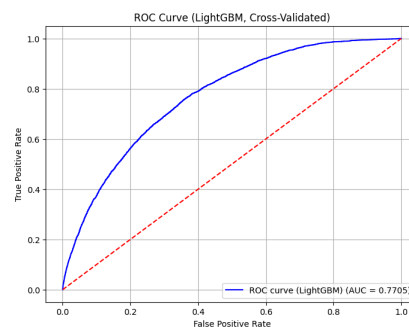


Figure 2.1: LGBM Model's ROC Curve Result

Among the three models, LightGBM achieved the highest ROC-AUC (0.7705), demonstrating superior ability in predicting claim status. Its combination of speed, flexibility, and high accuracy makes it the recommended model for further deployment or business integration.

## Model Interpretation

SHAP (SHapley Additive exPlanations) is a widely used model interpretation framework that helps elucidate the outcomes of machine learning models. To interpret the impact of insurance features on the Total Loss Cost per unit (LC), Historically Adjusted Loss Cost (HALC), and Claim Status (CS), we employed beeswarm plots of SHAP values to demonstrate the magnitude and direction of the feature contributions to the target variables.

Following model fine-tuning and evaluation, we selected LightGBM as our optimized model for predicting LC and CS, while ExtremeGBM was chosen to predict HALC, respectively. In this section, we discuss their top five features, both positively and negatively impactful, for each model.

### Total Loss Cost (LC) and Historically Adjusted Loss Cost (HALC)

- In the prediction of LC (see Fig. 3.1), the most influential features are largely related to insurance policy and driver characteristics. The plot shows that **the net premium amount for the current year (X.14)** and **the number of policies held during the current year (X.9)** have a substantial positive impact on LC. Conversely, factors like **the number of policies canceled in the current year (X.12)**, **policy years**, and **license years** exhibited a negative relationship with LC. This suggests that newer, more active policies are associated with higher loss costs, while policies that are canceled or have shorter histories tend to result in lower loss costs.
- Turning to the model for HALC (see Fig. 3.2), we observe similar patterns, with insurance policy-related features being prominent predictors. Notably, **the net premium amount for the current year (X.14)** remains a strong positive factor for HALC, while the **number of policies held in the current year (X.9)** drops out of the top five influential features. Additionally, compared with the LC model, the predictors such as **the total number of years that the insured has been associated with the entity (X.8)** and **the last payment method (X.13\_0: annually)** emerge as greater negative contributors in the HALC model, suggesting that HALC is more influenced by long-term customer engagement and payment patterns.

### Claim Status (CS)

- In the prediction of CS (see Fig. 3.3), the model shares several similarities with the LC and HALC models in terms of the top positive and negative features. Particularly, **the number of policies held in the current year (X.9)** plays an even more prominent role in predicting the likelihood of a claim, moving up to the second most important feature in the model. This highlights the significance of policy engagement in determining claim likelihood. The trend in other features is also consistent with the previous models, reinforcing the role of policy-related factors in the prediction of claims.

## Innovations

To ensure the robustness of our modeling approach, we implemented adversarial validation—a diagnostic technique used to detect distributional differences between the training and test sets. This method involves labeling the training data as 0 and the test data as 1, then training a classifier to distinguish between them. If the classifier performs well (AUC significantly above 0.5), it suggests a distribution shift or data leakage. If not, the two datasets likely come from the same underlying distribution.

Our results showed that both Logistic Regression and LightGBM classifiers failed to distinguish between the training and test sets, with validation AUCs near 0.51. This indicates no significant distribution shift or leakage. While the LightGBM classifier showed a high training AUC (0.78), the drop to 0.51 in validation confirms model overfitting without real signal differentiation. Overall, these findings support the validity of our modeling pipeline and confirm that train-test splits are representative and safe to use for evaluation.

## Conclusions

This project aimed to support data-driven premium pricing for car insurance by predicting three key metrics: Loss Cost (LC), Historically Adjusted Loss Cost (HALC), and Claim Status (CS). Leveraging both statistical and machine learning models, we addressed the challenges of highly skewed, zero-inflated data and evaluated the performance of multiple approaches using cross-validation and SHAP-based interpretation.

For LC prediction, we ultimately selected XGBoost Tweedie due to its superior performance in capturing a wider range of claim values, particularly in high-loss regions. For HALC, LightGBM Tweedie offered the most reliable results, especially across moderately high values. These models were selected based on extensive cross-validation, and our decision to include all features—rather than limiting ourselves to Lasso-selected ones—was grounded in a performance-driven rationale.

In the binary classification task of predicting Claim Status, LightGBM achieved the highest ROC-AUC score, offering a strong balance between speed and accuracy. SHAP value interpretation confirmed that key features like net premium, policy cancellations, and insurance years played consistently strong roles across all three prediction tasks. Additionally, adversarial validation showed no major distribution shift between the training and test sets, giving us confidence in the generalizability of our models.

From a business standpoint, our work demonstrates that predictive analytics, when carefully calibrated and interpreted, can provide meaningful insights for segmenting insurance risk. While caution is warranted in using predictions for individual pricing, our models are robust tools for informing strategic pricing and risk selection at a group level.

## Appendix

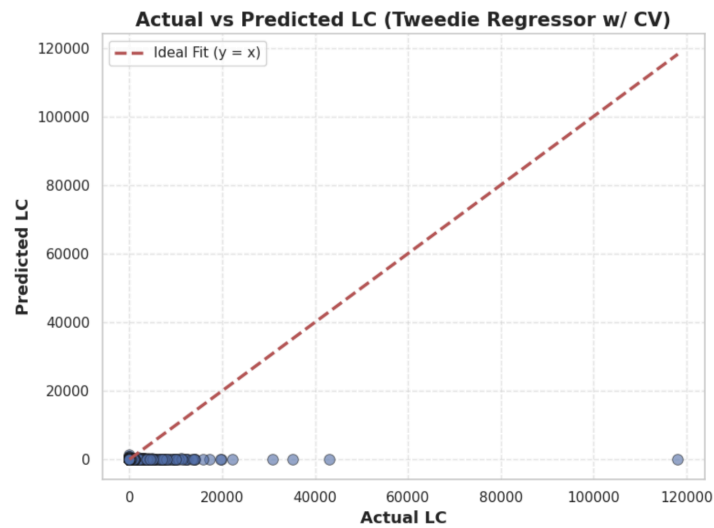


Figure 1.1: Tweedie Regressor LC Model Result

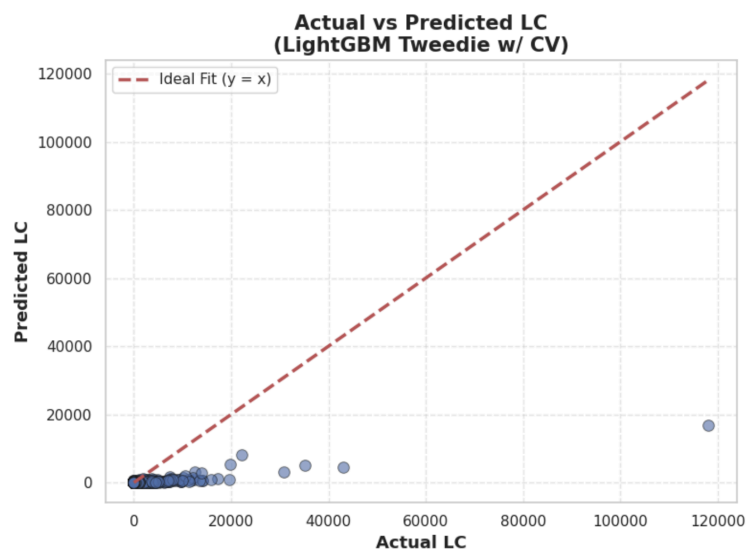


Figure 1.2: LGBM Tweedie LC Model Result

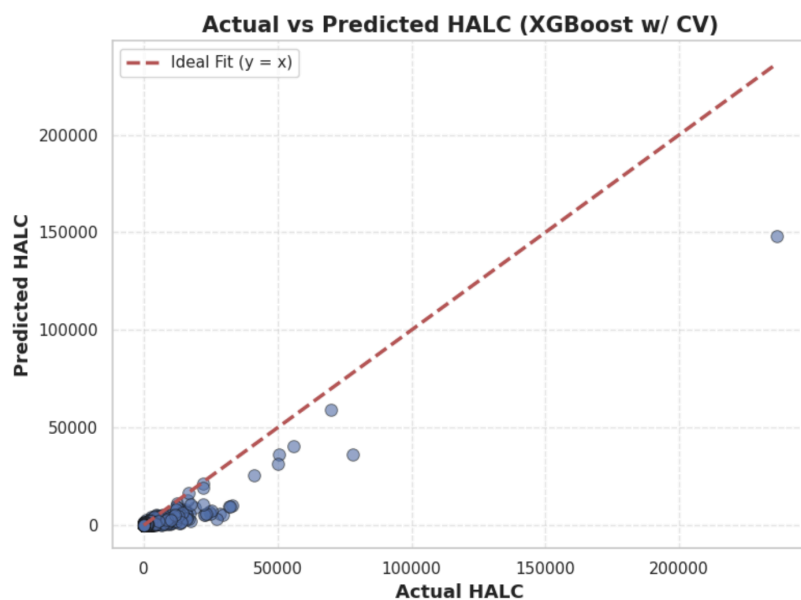


Figure 1.4: XGB Tweedie HALC Model Result

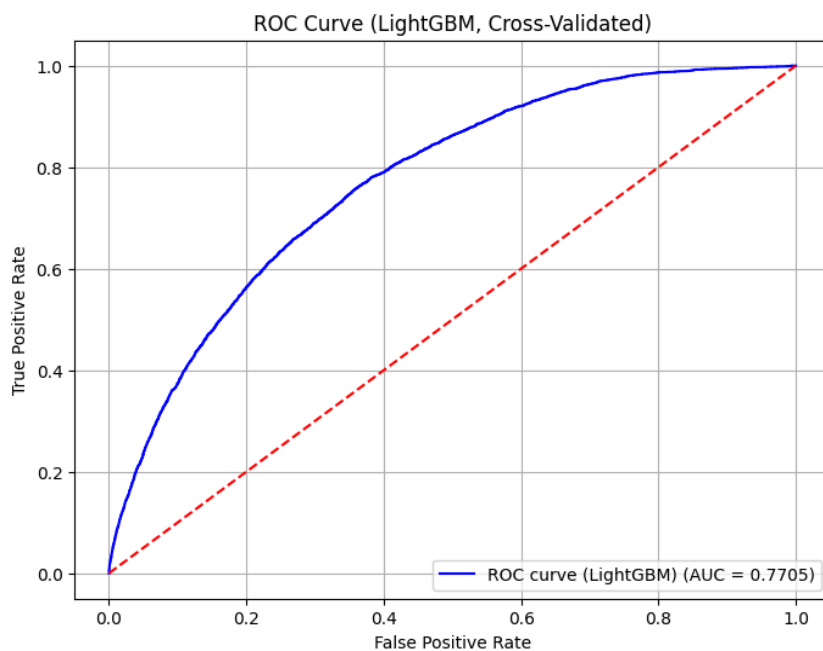


Figure 2.1: LGBM Model's ROC Curve Result



## Model Interpretation - SHAP

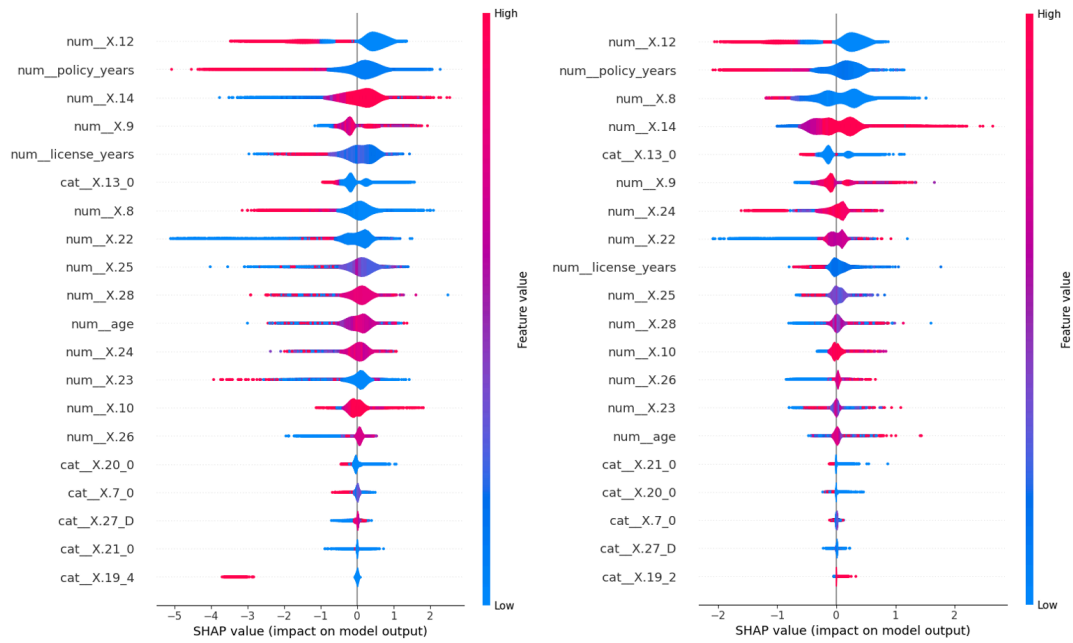


Figure 3.1: Beeswarm plot of LightXGB for LC; Figure 3.2: Beeswarm plot of XGBM for HALC

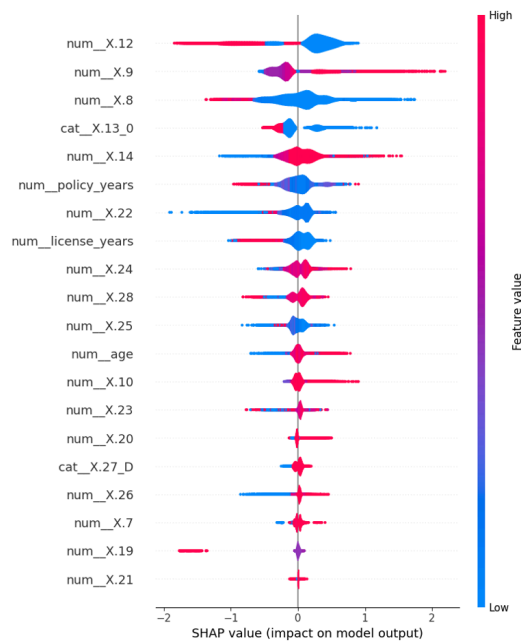


Figure 3.3: Beeswarm plot of XGBM for CS

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
import pandas as pd
```

```
test = pd.read_csv("/content/drive/My Drive/DS0530 Final Project/insurance_test.csv")
test.head()
```

5 rows × 23 columns

	X.2	X.3	X.4	X.5	X.6	X.7	X.8	X.9	X.10	X.11	...	X.19	X.20	X.21	X.22	X.23	X.24	X.25	X.26	X.27	X.28
0	23/06/2017	23/06/2018	23/06/2019	13/09/1982	03/02/2011	0	2	2	2	1	...	3	1	0	2003	115	1910	16400.00	5	D	1305
1	29/06/2015	29/06/2016	29/06/2017	07/07/1946	12/08/1966	0	2	1	1	1	...	3	0	0	1999	90	1597	13480.70	5	P	1083
2	14/03/2018	14/03/2018	14/03/2019	26/12/1957	02/09/1977	0	1	4	4	2	...	3	0	0	2003	143	2148	36500.00	5	D	1495
3	16/10/2014	16/10/2018	16/10/2019	27/02/1961	29/10/1980	1	5	1	1	1	...	2	1	0	1998	60	1686	12356.81	5	D	1010
4	01/07/2015	01/07/2017	01/07/2018	03/07/1986	02/08/2006	0	3	1	1	1	...	3	0	0	2015	66	998	11800.00	5	P	933

```
test.columns
```

```
Index(['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', 'X.24',
      'X.25', 'X.26', 'X.27', 'X.28'],
      dtype='object')
```

## Task 1.1: LC

Start coding or [generate](#) with AI.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor

# Load data
df = pd.read_csv("/content/drive/My Drive/DS0530 Final Project/insurance_train.csv")

# Compute LC
df['LC'] = df.apply(lambda row: row['X.15'] / row['X.16'] if row['X.16'] > 0 else 0, axis=1)

# Dates → derived features
today = pd.to_datetime('2025-04-23')
for col in ['X.2', 'X.5', 'X.6']:
    df[col] = pd.to_datetime(df[col], dayfirst=True, errors='coerce')
df['age'] = (today - df['X.5']).dt.days // 365
df['license_years'] = (today - df['X.6']).dt.days // 365
df['policy_years'] = (today - df['X.2']).dt.days // 365

# Feature set
features = [
    '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', 'X.24',
    'X.25', 'X.26', 'X.27', 'X.28', 'age', 'license_years', 'policy_years'
]

# Drop rows with missing values
df.dropna(subset=features, inplace=True)

# Define X and y
X = df[features]
y = df['LC']

# Categorical and numeric columns
categorical_cols = ['X.7', 'X.13', 'X.19', 'X.20', 'X.21', 'X.27']
numeric_cols = [col for col in features if col not in categorical_cols]

# Preprocessing
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# XGBoost Regressor
xgb_model = XGBRegressor(
    objective='reg:tweedie',
    random_state=42,
    n_jobs=-1,
    verbosity=0
)

# Pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', xgb_model)
])

# Parameter grid
param_grid = {
    'regressor__reg_alpha': [0.01, 0.1, 0.5, 1.0],
    'regressor__tweedie_variance_power': [1.0, 1.5, 1.9]
}

# GridSearchCV
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    verbose=1,
```

```

    n_jobs=-1
)
grid_search.fit(X, y)

# Best model and results
best_lc_model = grid_search.best_estimator_
y_pred = best_lc_model.predict(X)
mse = mean_squared_error(y, y_pred)

# Output
print("Best Parameters:", grid_search.best_params_)
print("MSE (CV Tuned Model):", mse)

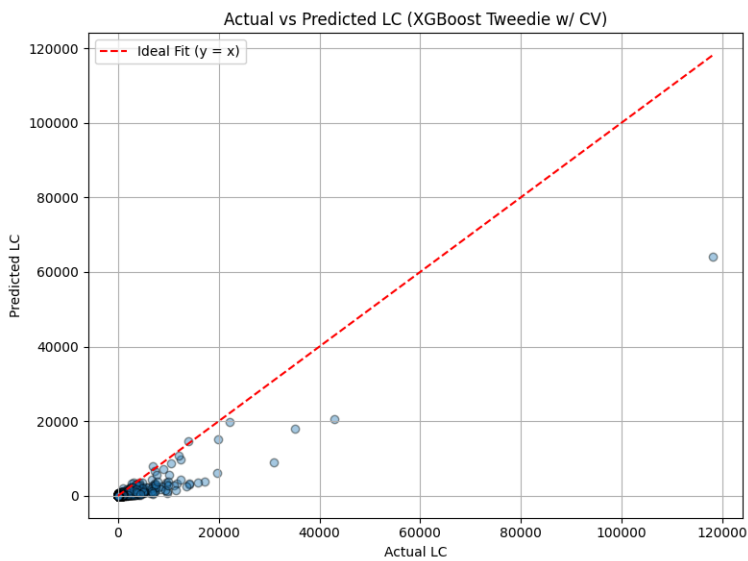
# Plot
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.4, edgecolor='k')
plt.plot([0, max(y.max(), y_pred.max())], [0, max(y.max(), y_pred.max())],
         'r--', label='Ideal Fit (y = x)')
plt.xlabel("Actual LC")
plt.ylabel("Predicted LC")
plt.title("Actual vs Predicted LC (XGBoost Tweedie w/ CV)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```

```

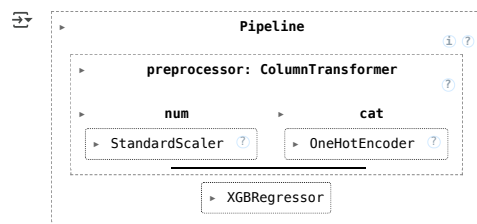
Fitting 5 folds for each of 12 candidates, totalling 60 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are non-finite: [-3.26756437e+35 -7.15808121e
-7.15497113e+05 nan -3.26756437e+35 -7.16128692e+05
nan -3.26756437e+35 -7.15392638e+05 nan]
warnings.warn(
Best Parameters: {'regressor__reg_alpha': 1.0, 'regressor__tweedie_variance_power': 1.5}
MSE (CV Tuned Model): 217392.02249079116

```



## ▼ applying model to test csv (best\_lc\_model)

```
best_lc_model
```



```

# === Apply model to test data ===
test_df = pd.read_csv("/content/drive/My Drive/DS0530 Final Project/insurance_test.csv")

# Derived features on test set
for col in ['X.2', 'X.5', 'X.6']:
    test_df[col] = pd.to_datetime(test_df[col], dayfirst=True, errors='coerce')

test_df['age'] = (today - test_df['X.5']).dt.days // 365
test_df['license_years'] = (today - test_df['X.6']).dt.days // 365
test_df['policy_years'] = (today - test_df['X.2']).dt.days // 365

# Drop rows with missing required features
test_df_clean = test_df.dropna(subset=features)

# Predict
X_test = test_df_clean[features]
test_df_clean['LC'] = best_lc_model.predict(X_test)

test_df_clean

```

	X.2	X.3	X.4	X.5	X.6	X.7	X.8	X.9	X.10	X.11	...	X.23	X.24	X.25	X.26	X.27	X.28	age	license_years	policy_years	LC	
0	2017-06-23	23/06/2018	23/06/2019	1982-09-13	2011-02-03		0	2	2	2	1	...	115	1910	16400.00	5	D	1305	42	14	7	10.596333
1	2015-06-29	29/06/2016	29/06/2017	1946-07-07	1966-08-12		0	2	1	1	1	...	90	1597	13480.70	5	P	1083	78	58	9	17.308191
2	2018-03-14	14/03/2018	14/03/2019	1957-12-26	1977-09-02		0	1	4	4	2	...	143	2148	36500.00	5	D	1495	67	47	7	26.231077
3	2014-10-16	16/10/2018	16/10/2019	1961-02-27	1980-10-29		1	5	1	1	1	...	60	1686	12356.81	5	D	1010	64	44	10	7.597871
4	2015-07-01	01/07/2017	01/07/2018	1986-07-03	2006-08-02		0	3	1	1	1	...	66	998	11800.00	5	P	933	38	18	9	0.075065
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15782	2012-05-25	25/05/2017	25/05/2018	1966-06-07	1984-12-07		0	8	1	2	1	...	90	1997	19380.16	5	D	1290	58	40	12	1.112224
15783	2017-12-01	01/12/2017	01/12/2018	1972-03-18	1990-07-12		1	3	2	2	1	...	90	1974	18991.98	4	D	1260	53	34	7	32.867851
15784	2015-04-02	02/04/2018	02/04/2019	1953-11-18	2006-07-10		1	5	1	2	1	...	69	1198	12059.99	5	P	1095	71	18	10	0.097427
15785	2016-08-30	30/08/2018	30/08/2019	1972-12-08	2007-04-23		1	3	1	1	1	...	90	1389	12092.36	5	P	895	52	18	8	81.322578
15786	2007-07-30	30/07/2018	30/07/2019	1966-09-05	1985-04-30		0	22	1	2	1	...	110	1587	18450.00	5	P	1200	58	40	17	2.217338

15787 rows × 27 columns

```
# prompt: average value of LC in df

print(f"Average value of LC in df: {df['LC'].mean()}")

Average value of LC in df: 70.73370997061471

# prompt: average value of LC in test_clean

print(f"Average value of LC in test_clean: {test_df_clean['LC'].mean()}")

Average value of LC in test_clean: 29.5382137298584

Start coding or generate with AI.
```

Task 1.2: HALC

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor

# Load data
df = pd.read_csv("/content/drive/My Drive/DS0530 Final Project/insurance_train.csv")

# Compute HALC = (X.15 / X.16) * X.18
df['HALC'] = df.apply(lambda row: (row['X.15'] / row['X.16']) * row['X.18'] if row['X.16'] > 0 else 0, axis=1)

# Dates -> derived features
today = pd.to_datetime('2025-04-23')
for col in ['X.2', 'X.5', 'X.6']:
    df[col] = pd.to_datetime(df[col], dayfirst=True, errors='coerce')
df['age'] = (today - df['X.5']).dt.days // 365
df['license_years'] = (today - df['X.6']).dt.days // 365
df['policy_years'] = (today - df['X.2']).dt.days // 365

# Feature set
features = [
    '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', 'X.24',
    'X.25', 'X.26', 'X.27', 'X.28', 'age', 'license_years', 'policy_years'
]

# Drop rows with missing values
df.dropna(subset=features + ['HALC'], inplace=True)

# Define X and y
X = df[features]
y = df['HALC']

# Categorical and numeric columns
categorical_cols = ['X.7', 'X.13', 'X.19', 'X.20', 'X.21', 'X.27']
numeric_cols = [col for col in features if col not in categorical_cols]

# Preprocessing
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# XGBoost Regressor
xgb_model = XGBRegressor(
    objective='reg:tweedie',
    random_state=42,
    n_jobs=-1,
    verbosity=0
)

# Pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', xgb_model)
])

# Parameter grid
param_grid = {
    'regressor__reg_alpha': [0.01, 0.1, 0.5, 1.0],
    'regressor__tweedie_variance_power': [1.0, 1.5, 1.9]
}

# GridSearchCV
grid_search = GridSearchCV
```

```

    pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(X, y)

# Best model and results
best_HALC_model = grid_search.best_estimator_
y_pred = best_HALC_model.predict(X)
mse = mean_squared_error(y, y_pred)

# Output
print("Best Parameters:", grid_search.best_params_)
print("MSE (CV Tuned Model):", mse)

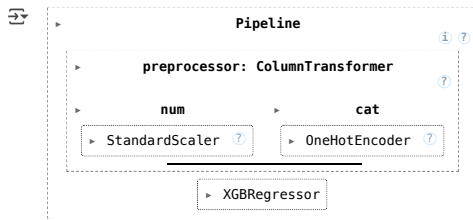
# Plot
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.4, edgecolor='k')
plt.plot([0, max(y.max(), y_pred.max())], [0, max(y.max(), y_pred.max())],
         'r--', label='Ideal Fit (y = x)')
plt.xlabel("Actual HALC")
plt.ylabel("Predicted HALC")
plt.title("Actual vs Predicted HALC (XGBoost Tweedie w/ CV)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```

 [Show hidden output](#)

## ▼ apply model to test csv (best\_halc\_model)

best\_HALC\_model




```

X_test = test_df_clean[features]
test_df_clean['HALC'] = best_HALC_model.predict(X_test)

```

test\_df\_clean.head()




	X.2	X.3	X.4	X.5	X.6	X.7	X.8	X.9	X.10	X.11	...	X.24	X.25	X.26	X.27	X.28	age	license_years	policy_years	LC	HALC
0	2017-06-23	23/06/2018	23/06/2019	1982-09-13	2011-02-03	0	2	2	2	1	...	1910	16400.00	5	D	1305	42	14	7	10.596333	15.991317
1	2015-06-29	29/06/2016	29/06/2017	1946-07-07	1966-08-12	0	2	1	1	1	...	1597	13480.70	5	P	1083	78	58	9	17.308191	8.391123
2	2018-03-14	14/03/2018	14/03/2019	1957-12-26	1977-09-02	0	1	4	4	2	...	2148	36500.00	5	D	1495	67	47	7	26.231077	12.355467
3	2014-10-16	16/10/2018	16/10/2019	1961-02-27	1980-10-29	1	5	1	1	1	...	1686	12356.81	5	D	1010	64	44	10	7.597871	33.255417
4	2015-07-01	01/07/2017	01/07/2018	1986-07-03	2006-08-02	0	3	1	1	1	...	998	11800.00	5	P	933	38	18	9	0.075065	0.661446

5 rows x 28 columns


# prompt: average value for HALC column in df

```
print(f"Average HALC: {df['HALC'].mean()}")
```

 Average HALC: 129.8917770351618

# prompt: average value for HALC column in test\_df\_clean

```
print(f"Average HALC: {test_df_clean['HALC'].mean()}")
```

 Average HALC: 49.603599548339844

## ▼ Task 2: CS

```

# prepare
# Step 1: Import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb

from sklearn.model_selection import GridSearchCV, cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Step 2: Load and process data
df = pd.read_csv("/content/drive/MyDrive/DS0530Public/Group Project/Data/insurance_train.csv")

# Label: Claim_Status
df['Claim_Status'] = (df['X.16'] > 0).astype(int)

# Parse dates
today = pd.to_datetime('2025-04-23')
for col in ['X.2', 'X.5', 'X.6']:

```

```

df[col] = pd.to_datetime(df[col], dayfirst=True, errors='coerce')

df['age'] = (today - df['X.5']).dt.days // 365
df['license_years'] = (today - df['X.6']).dt.days // 365
df['policy_years'] = (today - df['X.2']).dt.days // 365

# Select features
features = [
    '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', 'X.24',
    'X.25', 'X.26', 'X.27', 'X.28', 'age', 'license_years', 'policy_years'
]
df.dropna(subset=features, inplace=True)

X = df[features]
y = df['Claim_Status']

# Step 3: Preprocessing
preprocessor_cs = ColumnTransformer([
    ('num', StandardScaler(), ['age', 'license_years', 'policy_years', 'X.7', 'X.8', 'X.9', 'X.10', 'X.11', 'X.12', 'X.14', 'X.19', 'X.20', 'X.21', 'X.22', 'X.23', 'X.24', 'X.25', 'X.26', 'X.27', 'X.28']),
    ('cat', OneHotEncoder(handle_unknown='ignore'), ['X.13', 'X.19', 'X.20', 'X.21', 'X.27'])
])

X_transformed = preprocessor_cs.fit_transform(X)

# model: LightGBM
import lightgbm as lgb
from sklearn.model_selection import GridSearchCV, cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns

# Step 4: Define LightGBM Classifier
lgb_model = lgb.LGBMClassifier(objective='binary', random_state=42)

# Step 5: Parameter Grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, -1],
    'learning_rate': [0.05, 0.1],
    'num_leaves': [31, 50]
}

# Step 6: GridSearchCV with ROC-AUC scoring
grid_search_lgb = GridSearchCV(
    estimator=lgb_model,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_lgb.fit(X_transformed, y)

# Step 7: Best model
best_lgb_model = grid_search_lgb.best_estimator_

# Predict with cross_val_predict to simulate out-of-fold test
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_prob_lgb_cv = cross_val_predict(best_lgb_model, X_transformed, y, cv=cv, method="predict_proba")[:, 1]
y_pred_lgb_cv = (y_prob_lgb_cv >= 0.5).astype(int)

# Step 8: ROC-AUC score
roc_auc_lgb = roc_auc_score(y, y_prob_lgb_cv)
print(f"\nBest Parameters (LightGBM): {grid_search_lgb.best_params_}")
print(f"Cross-Validated ROC-AUC (LightGBM): {roc_auc_lgb:.4f}")

# # Step 9: Confusion Matrix
# cm_lgb_cv = confusion_matrix(y, y_pred_lgb_cv)

# plt.figure(figsize=(6, 5))
# sns.heatmap(cm_lgb_cv, annot=True, fmt="d", cmap='Blues', xticklabels=['No Claim', 'Claim'], yticklabels=['No Claim', 'Claim'])
# plt.title("Confusion Matrix (LightGBM, CV)")
# plt.xlabel("Predicted")
# plt.ylabel("Actual")
# plt.show()

# # Step 10: ROC Curve
# fpr_lgb, tpr_lgb, _ = roc_curve(y, y_prob_lgb_cv)
# plt.figure(figsize=(8, 6))
# plt.plot(fpr_lgb, tpr_lgb, color='b', label=f'ROC curve (LightGBM) (AUC = {roc_auc_lgb:.4f})')
# plt.plot([0, 1], [0, 1], color='r', linestyle='--')
# plt.xlabel("False Positive Rate")
# plt.ylabel("True Positive Rate")
# plt.title("ROC Curve (LightGBM, Cross-Validated)")
# plt.legend(loc='lower right')
# plt.grid(True)
# plt.show()

```



```

Fitting 5 folds for each of 24 candidates, totalling 120 fits
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
  warnings.warn(
[LightGBM] [Info] Number of positive: 4122, number of negative: 32736
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.007494 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1570
[LightGBM] [Info] Number of data points in the train set: 36858, number of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111835 -> initscore=-2.072137
[LightGBM] [Info] Start training from score -2.072137
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
  warnings.warn(
[LightGBM] [Info] Number of positive: 3298, number of negative: 26188
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.006040 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1562
[LightGBM] [Info] Number of data points in the train set: 29486, number of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111850 -> initscore=-2.071985
[LightGBM] [Info] Start training from score -2.071985
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed

```

```
warnings.warn(
[LightGBM] [Info] Number of positive: 3297, number of negative: 26189
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.006054 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1560
[LightGBM] [Info] Number of data points in the train set: 29486, number of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111816 -> initscore=-2.072327
[LightGBM] [Info] Start training from score -2.072327
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
warnings.warn(
[LightGBM] [Info] Number of positive: 3297, number of negative: 26189
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.006127 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1559
[LightGBM] [Info] Number of data points in the train set: 29486, number of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111816 -> initscore=-2.072327
[LightGBM] [Info] Start training from score -2.072327
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
warnings.warn(
[LightGBM] [Info] Number of positive: 3298, number of negative: 26189
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.006050 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1556
[LightGBM] [Info] Number of data points in the train set: 29487, number of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111846 -> initscore=-2.072023
[LightGBM] [Info] Start training from score -2.072023
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
```

## ✓ apply model to test csv (best\_lgb\_model for CS)

```
best_lgb_model
```

```

LGBMClassifier
LGBMClassifier(learning_rate=0.05, objective='binary', random_state=42)
```

```
# Predict
X_test = test_df_clean[features]
X_test_transformed = preprocessor_cs.transform(X_test)
```

```
test_df_clean['CS'] = best_lgb_model.predict_proba(X_test_transformed)[:, 1]
# test_df_clean['CS'] = (test_df_clean['CS_prob'] >= 0.5).astype(int)
```

```

/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed
warnings.warn(
```

```
test_df_clean
```

	X.2	X.3	X.4	X.5	X.6	X.7	X.8	X.9	X.10	X.11	...	X.25	X.26	X.27	X.28	age	license_years	policy_years	LC	HALC	CS
0	2017-06-23	23/06/2018	23/06/2019	1982-09-13	2011-02-03	0	2	2	2	1	...	16400.00	5	D	1305	42	14	7	10.596333	15.991317	0.144180
1	2015-06-29	29/06/2016	29/06/2017	1946-07-07	1966-08-12	0	2	1	1	1	...	13480.70	5	P	1083	78	58	9	17.308191	8.391123	0.051296
2	2018-03-14	14/03/2018	14/03/2019	1957-12-26	1977-09-02	0	1	4	4	2	...	36500.00	5	D	1495	67	47	7	26.231077	12.355467	0.314632
3	2014-10-16	16/10/2018	16/10/2019	1961-02-27	1980-10-29	1	5	1	1	1	...	12356.81	5	D	1010	64	44	10	7.597871	33.255417	0.116670
4	2015-07-01	01/07/2017	01/07/2018	1986-07-03	2006-08-02	0	3	1	1	1	...	11800.00	5	P	933	38	18	9	0.075065	0.661446	0.010267
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15782	2012-05-25	25/05/2017	25/05/2018	1966-06-07	1984-12-07	0	8	1	2	1	...	19380.16	5	D	1290	58	40	12	1.112224	1.165303	0.016742
15783	2017-12-01	01/12/2017	01/12/2018	1972-03-18	1990-07-12	1	3	2	2	1	...	18991.98	4	D	1260	53	34	7	32.867851	40.875690	0.150241
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

```
# prompt: average of values in CS column of df
print(f"Average CS: {df['Claim_Status'].mean()}")
```

```
Average CS: 0.1118346084974768
```

```
# prompt: average of values in CS column of test_df_clean
print(f"Average of CS column in test_df_clean: {test_df_clean['CS'].mean()}")
```

```
Average of CS column in test_df_clean: 0.11259701889704155
```

## ✓ Final Deliverable

```
test_df_clean = test_df_clean[['LC', 'HALC', 'CS']]
```

```
test_df_clean
```



	LC	HALC	CS
0	10.596333	15.991317	0.144180
1	17.308191	8.391123	0.051296
2	26.231077	12.355467	0.314632
3	7.597871	33.255417	0.116670
4	0.075065	0.661446	0.010267
...	...	...	...
15782	1.112224	1.165303	0.016742
15783	32.867851	40.875690	0.150241
15784	0.097427	1.203174	0.014088
15785	81.322578	74.765663	0.218402
15786	2.217338	5.614635	0.041892

15787 rows x 3 columns

```
test_df_clean.to_csv('group_27_prediction.csv')
```

Start coding or [generate](#) with AI.