Title: Car Insurance Claim Prediction

By,

- Harish K M

Abstract:

The automobile insurance industry faces the dual challenge of offering competitive premiums while controlling claim-related losses. Accurate prediction of insurance claims can significantly enhance underwriting decisions, risk management, and customer segmentation. This project leverages structured data encompassing policyholder demographics, vehicle specifications, safety features, and historical claim information to predict the likelihood of a policyholder filing a claim.

Exploratory Data Analysis (EDA) highlights key patterns, such as higher claim rates among younger drivers, older vehicles, high-power engines, and densely populated urban areas. Data preprocessing includes missing value imputation, categorical encoding, and feature scaling. Feature selection using SelectKBest identifies the top 15 predictive features. Multiple machine learning models—Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest, XGBoost, and LightGBM—are evaluated, with LightGBM achieving the highest ROC-AUC and F1-score.

A Streamlit application is developed to allow interactive prediction based on user inputs. The deployed model and preprocessing pipeline ensure accurate, reproducible predictions. The project demonstrates how predictive analytics can support risk-based pricing, customer retention strategies, and fraud detection, ultimately improving profitability and operational efficiency for insurers.

Objective:

To predict the likelihood of a car insurance policyholder making a claim using historical policyholder, vehicle, and safety feature data. The model helps insurers optimize risk management, pricing, and customer segmentation strategies.

Introduction:

The automobile insurance sector has experienced significant growth over the past decades, driven by rising vehicle ownership, evolving regulatory frameworks, and increasing consumer awareness. Insurers must balance providing affordable premiums to customers with maintaining profitability by managing claim-related risks. One of the most critical challenges in this domain is predicting whether a policyholder is likely to file a claim. Accurate claim

prediction enables insurers to optimize underwriting, reduce losses, and tailor products to individual risk profiles.

This project focuses on predicting car insurance claims using machine learning techniques. The dataset contains information on policyholder demographics, vehicle characteristics, safety features, and historical claims, providing a comprehensive basis for risk assessment. Key objectives include:

1. **Exploratory Data Analysis (EDA):** Understanding relationships between features and claim probability, such as the impact of age, car type, engine specifications, and geographical factors.
2. **Data Preprocessing:** Handling missing values, encoding categorical variables, scaling features, and performing feature selection.
3. **Modeling:** Evaluating multiple machine learning algorithms including Logistic Regression, Decision Tree, Random Forest, XGBoost, and LightGBM to identify the best-performing model.
4. **Deployment:** Building an interactive Streamlit application that predicts claim probability for new users based on the trained model.

The insights gained from this project not only support predictive modeling but also provide actionable business recommendations. For example, risk-based pricing, customer segmentation, and safety-based premium discounts can improve both profitability and customer satisfaction. Ultimately, this project demonstrates the power of data-driven decision making in the automobile insurance industry.

Methodology/Approach:

The project followed a structured approach combining data exploration, preprocessing, modeling, and deployment to predict insurance claims. Each step is described below:

Data Collection:

- The dataset includes detailed information about policyholders, vehicles, policy attributes, safety features, and historical claim status (`is_claim`).
- It contains both numeric (e.g., age, engine power, premium, car age) and categorical (e.g., fuel type, car make, segment, area cluster) features.

Exploratory Data Analysis:

1) Overall claim rate and how does it vary by policy_tenure:
   Claim risk increases with policy tenure, peaking around 1–2 years. New customers are safer, but insurers need strong renewal strategies that both retain customers and mitigate rising claim risk.

2) Claim rate by age_of_policyholder:

Pricing: Apply age-based premium adjustments, not just flat rates.
Retention: Reward low-risk age segments (26–35) with loyalty offers.

3) Claim frequency vs age_of_car (continuous + buckets)

Counterintuitive trend: Claim frequency decreases as car age increases — opposite to usual expectations. Possible reasons: fewer old cars insured, or owners of older cars avoid filing small claims.

4) Claim rate by area_cluster and by population_density:

Even though we might assume urban areas with higher population density have more claims due to traffic and accidents, the data shows claim frequency is cluster-specific rather than population-specific. Insurers can prioritize cluster-based underwriting rules rather than general urban/rural assumptions.

5) Claim rate by make and model:

Claims vary significantly by model even within the same make. High-claim models (M2, M6, M7) need risk-adjusted pricing and preventive programs.

6) Claim rate by segment:
Car segment is a moderate predictor of claims. Higher claims in B2, C1, and C2 segments likely reflect vehicle value, usage, and repair complexity. Insurers can leverage this insight for segment-specific pricing, marketing, and preventive campaigns to reduce claims and improve profitability.

7) Relationship between fuel_type and claim rate & severity:

Fuel type moderately predicts claim frequency. City-focused CNG vehicles have slightly lower claims → opportunity for loyalty programs or bundled discounts.

8) Effect of engine displacement / cylinders on claims

Claim rate increases slightly with engine displacement from small cars (<=800cc) to mid-sized engines (1200–1600cc). Very high-displacement engines (1600+) have insufficient data, likely few policies.

Both displacement and cylinder show small positive correlation with claims (likely ~0.01–0.02). Larger engines or more cylinders slightly increase claim risk, possibly due to higher repair costs or performance-related accidents.

9) Claim rate vs max_torque / max_power (parse numeric)

Both torque and power have very low positive correlation with claims. Slight trend indicates more powerful cars may have marginally higher claim rates, possibly due to sporty or performance-oriented usage.

Claim rate is lowest for 60–100 Nm and peaks at 100–150 Nm. High-torque cars (100+ Nm) are slightly more prone to claims, likely sporty or performance vehicles.

Mid-power cars (70–90 BHP) show the highest claim rate, possibly reflecting popular performance models. Low-power cars (<70 BHP) have the lowest claims, mainly small or economy cars.

10) Transmission Type (Manual vs Automatic) and claim behaviour

Claim rates for Automatic vs Manual are very close (~6.4%). Slightly higher for Automatic cars, possibly due to urban usage, newer cars, or driver demographics.

11) Safety features bundle impact: count of safety features → claim rate

Claim rates do not decrease monotonically with more safety features. Surprisingly, score 4 shows the lowest claim rate (4.13%), while 5–6 features show slightly higher claim rates.

12) NCAP rating vs claim frequency & severity

Claim frequency ranges between 6.2% and 6.7% across NCAP ratings (0–5). NCAP 5-star vehicles surprisingly show the highest claim frequency (6.68%), even higher than unrated or 2–3 star cars.

13) Parking-related features (sensors/camera) and small-claim frequency

- Camera only → lowest claim rate (5.39%)
- Sensors only → higher claim rate (6.41%)

14) Rear brake type (Drum vs Disc) and claims.

- Vehicles with Disc rear brakes → Claim rate 6.43%
- Vehicles with Drum rear brakes → Claim rate 6.39%

15) Relationship between turning_radius/steering_type and urban claims

Vehicles with different steering types (e.g., Power, Manual, Electronic) do not show large claim rate differences. Turning radius has negligible correlation with claim frequency, suggesting maneuverability is not a direct predictor of insurance claims.

16) Interaction: age_of_policyholder × car_segment on claim probability

Small sample bias (few youngest policyholders). Many may drive lower-powered entry-level cars (A, B1).

17) Claim probability by number of airbags

Cars with 1 airbag have slightly lower claim probability (6.0%) than those with 2 (6.36%) or 6 airbags (6.5%). The dataset is dominated by 2-airbag vehicles, so the effect size may appear muted.

18) Analyze is_parking_camera vs frequency of low-cost claims.

- No Parking Camera: Claim probability = 6.41%, count = 35,704 policies
- Yes Parking Camera: Claim probability = 6.37%, count = 22,888 policies

19) Relationship between is_ecw/is_speed_alert and claims

ECW (Engine Check Warning): Vehicles with ECW show slightly higher claim probability (6.5% vs 6.1%). Likely because ECW-equipped cars are newer, higher-value, or more complex → slightly higher filing.

20) Claim probability vs gross_weight (vehicle mass)

Claim probability peaks in the 1185–1335 kg range (~6.86%). Slight decline for heavier cars (>1335 kg), possibly because heavier cars are premium, better-engineered, or driven more cautiously.

Data Preprocessing:

- **Missing Value Handling:**
  Numeric features: Imputed using median values. Categorical features: Imputed using mode values.

  Code:

```
# Separate target from training data
target = "is_claim"

# Features only (drop target in train)
X_train = train.drop(columns=[target, "policy_id"])
y_train = train[target]

# Keep test as it is (no target column)
X_test = test

# Missing Value Handling
# Numeric columns
num_cols =
X_train.select_dtypes(include=[np.number]).columns
for col in num_cols:
    median_val = X_train[col].median()
    X_train[col] = X_train[col].fillna(median_val)
    X_test[col] = X_test[col].fillna(median_val)
```

```
# Categorical columns
cat_cols =
X_train.select_dtypes(include=["object"]).columns
for col in cat_cols:
    mode_val = X_train[col].mode()[0]
    X_train[col] = X_train[col].fillna(mode_val)
    X_test[col] = X_test[col].fillna(mode_val)

# Verify
print("Missing values after imputation - Train:",
X_train.isnull().sum().sum())
print("Missing values after imputation - Test:",
X_test.isnull().sum().sum())
```

- **Train-Validation Split:**
  80%-20% split with stratification to preserve class balance.

  Code:

```
# Split into Train (80%) and Validation (20%)

X_tr, X_val, y_tr, y_val = train_test_split(

    X_train, y_train,

    test_size=0.2,

    random_state=42,

    stratify=y_train   # keeps class balance in both
splits

)



print("Train shape:", X_tr.shape)

print("Validation shape:", X_val.shape)
```

- **Encoding Categorical Variables:**
  One-Hot Encoding (OHE) applied via `ColumnTransformer`.

  Code:

```
# Preprocess: numeric + categorical encoding

num_cols =
X_tr.select_dtypes(include=[np.number]).columns.tolist()

cat_cols =
X_tr.select_dtypes(include=["object"]).columns.tolist()

# Define Column Transformer

preprocessor = ColumnTransformer(

    transformers=[

        ("num", "passthrough", num_cols),

        ("cat", OneHotEncoder(handle_unknown="ignore"),
cat_cols)

    ]

)

# Fit on train, transform on all

X_tr_enc = preprocessor.fit_transform(X_tr)

X_val_enc = preprocessor.transform(X_val)

X_test_enc = preprocessor.transform(X_test)



print("Encoded Train shape:", X_tr_enc.shape)

print("Encoded Validation shape:", X_val_enc.shape)

print("Encoded Test shape:", X_test_enc.shape)
```

- **Feature Scaling:**
  StandardScaler applied to numeric and OHE features to normalize feature ranges.

  Code:

```
# Identify column types
num_cols =
X_tr.select_dtypes(include=[np.number]).columns.tolist()
```

```
cat_cols =
X_tr.select_dtypes(include=["object"]).columns.tolist()

# Fit only on training set, then transform
scaler = StandardScaler(with_mean=False)   # important
for sparse OHE output
X_tr_scaled = scaler.fit_transform(X_tr_enc)
X_val_scaled = scaler.transform(X_val_enc)
X_test_scaled = scaler.transform(X_test_enc)

print("Scaled Train shape:", X_tr_scaled.shape)
print("Scaled Validation shape:", X_val_scaled.shape)
print("Scaled Test shape:", X_test_scaled.shape)
```

Model Training and Evaluation:

- **Algorithms Evaluated:**

  Logistic Regression, Decision Tree, K-Nearest Neighbors, Support Vector Machine, Random Forest, XGBoost, LightGBM.

- **Class Imbalance Handling:**
  `class_weight='balanced'` or `scale_pos_weight` used for models.
- **Metrics Evaluated:**
  Accuracy, Precision, Recall, F1-Score, ROC-AUC, PR-AUC, and Confusion Matrix.

Code:

```
lr = LogisticRegression(class_weight='balanced',
random_state=42, max_iter=1000)

# Train on balanced training data
lr.fit(X_tr_scaled, y_tr)

# Predict on validation set
y_val_pred_lr = lr.predict(X_val_scaled)
y_val_prob_lr = lr.predict_proba(X_val_scaled)[:, 1]

pr_auc = average_precision_score(y_val, y_val_prob_lr)

# Initialize
dt = DecisionTreeClassifier(random_state=42,
class_weight='balanced')

# Train
dt.fit(X_tr_scaled, y_tr)
```

```
# Predict
y_val_pred_dt = dt.predict(X_val_scaled)
y_val_prob_dt = dt.predict_proba(X_val_scaled)[:, 1]

# Initialize
rf = RandomForestClassifier(n_estimators=200,
random_state=42, class_weight='balanced')

# Train
rf.fit(X_tr_scaled, y_tr)

# Predict
y_val_pred_rf = rf.predict(X_val_scaled)
y_val_prob_rf = rf.predict_proba(X_val_scaled)[:, 1]

# Initialize
xgb_model = xgb.XGBClassifier(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=6,
    scale_pos_weight=(y_tr.value_counts()[0] /
y_tr.value_counts()[1]),
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)

# Train
xgb_model.fit(X_tr_scaled, y_tr)

# Predict
y_val_pred_xgb = xgb_model.predict(X_val_scaled)
y_val_prob_xgb = xgb_model.predict_proba(X_val_scaled)[:,
1]

# Initialize
lgb_model = lgb.LGBMClassifier(
    n_estimators=200,
    learning_rate=0.1,
    class_weight='balanced',
    random_state=42
)

# Train
lgb_model.fit(X_tr_scaled, y_tr)

# Predict
y_val_pred_lgb = lgb_model.predict(X_val_scaled)
y_val_prob_lgb = lgb_model.predict_proba(X_val_scaled)[:,
1]
```

```
print("Accuracy:", accuracy_score(y_val, y_val_pred_lgb))
print("Precision:", precision_score(y_val,
y_val_pred_lgb))
print("Recall:", recall_score(y_val, y_val_pred_lgb))
print("F1-Score:", f1_score(y_val, y_val_pred_lgb))
print("ROC-AUC:", roc_auc_score(y_val, y_val_prob_lgb))
print("PR-AUC:", average_precision_score(y_val,
y_val_prob_lgb))
print("Confusion Matrix:\n", confusion_matrix(y_val,
y_val_pred_lgb))
print("\nClassification Report:\n",
classification_report(y_val, y_val_pred_lgb))
```

- **Hyperparameter Tuning:**
  GridSearchCV applied to optimize model parameters based on F1-Score.

Code:

```
# Imports
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score,
average_precision_score, confusion_matrix,
classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import lightgbm as lgb

# Define models and param grids
models_params = {
    "Logistic Regression": {
        "model": LogisticRegression(random_state=42,
max_iter=1000, class_weight='balanced'),
        "params": {
            "C": [0.01, 0.1, 1, 10],
            "penalty": ["l2"],
            "solver": ["lbfgs", "liblinear"],
            "class_weight": ["balanced", None]
        }
    },
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42,
class_weight='balanced'),
        "params": {
            "max_depth": [None, 5, 10, 20],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4],
            "class_weight": ["balanced", None]
```

```
        }
    },
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42,
class_weight='balanced'),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2],
            "class_weight": ["balanced",
"balanced_subsample"]
        }
    },
    "XGBoost": {
        "model": xgb.XGBClassifier(
            eval_metric="logloss",
            scale_pos_weight=(y_tr.value_counts()[0] /
y_tr.value_counts()[1]),
            random_state=42
        ),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [3, 6, 10],
            "learning_rate": [0.01, 0.1],
            "subsample": [0.8, 1],
            "colsample_bytree": [0.8, 1]
        }
    },
    "LightGBM": {
        "model":
lgb.LGBMClassifier(class_weight="balanced",
random_state=42),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [-1, 10, 20],
            "learning_rate": [0.01, 0.1],
            "num_leaves": [31, 50]
        }
    }
}

# Function to train & evaluate
def train_evaluate_grid(model_name, model, param_grid,
X_train, y_train, X_val, y_val):
    print(f"\n Tuning {model_name}")
    grid = GridSearchCV(model, param_grid, cv=5,
scoring="f1", n_jobs=1, verbose=1)
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
```

```
    print(f"Best params for {model_name}:",
grid.best_params_)

    # Predict
    y_pred = best_model.predict(X_val)
    if hasattr(best_model, "predict_proba"):
        y_prob = best_model.predict_proba(X_val)[:, 1]
    else:
        y_prob = best_model.decision_function(X_val)

    # Metrics
    results = {
        "Model": model_name,
        "Best_Params": grid.best_params_,
        "Accuracy": accuracy_score(y_val, y_pred),
        "Precision": precision_score(y_val, y_pred),
        "Recall": recall_score(y_val, y_pred),
        "F1": f1_score(y_val, y_pred),
        "ROC-AUC": roc_auc_score(y_val, y_prob),
        "PR-AUC": average_precision_score(y_val, y_prob),
        "Confusion_Matrix": confusion_matrix(y_val,
y_pred)
    }

    print(f" {model_name} Evaluation ")
    print("F1-Score:", results["F1"])
    print("ROC-AUC:", results["ROC-AUC"])
    print("PR-AUC:", results["PR-AUC"])
    print("Confusion Matrix:\n",
results["Confusion_Matrix"])
    return results

# Loop over all models
all_results = []
for name, mp in models_params.items():
    res = train_evaluate_grid(name, mp["model"],
mp["params"], X_tr_scaled, y_tr, X_val_scaled, y_val)
    all_results.append(res)

# Comparison Table
import pandas as pd
results_df = pd.DataFrame(all_results)
# Show main metrics sorted by F1-score
print("\nAll Models Comparison")
print(results_df[["Model", "Best_Params", "Accuracy",
"Precision", "Recall", "F1", "ROC-AUC", "PR-
AUC"]].sort_values(by="F1", ascending=False))
```

- **Best Model:**
  LightGBM trained and achieved the highest ROC-AUC and PR-AUC scores.

Code:

```
import pandas as pd

results_df = pd.DataFrame(all_results)

# Sort by ROC-AUC

results_sorted = results_df.sort_values(by="ROC-AUC",
ascending=False)

print("\n Model Comparison (sorted by ROC-AUC)")

print(results_sorted[["Model", "Best_Params", "Accuracy",
"Precision", "Recall", "F1", "ROC-AUC", "PR-AUC"]])



# Get the best model (highest ROC-AUC)

best_model_info = results_sorted.iloc[0]

best_model_name = best_model_info["Model"]

best_model_params = best_model_info["Best_Params"]

best_roc_auc = best_model_info["ROC-AUC"]



print("\n Best Model Based on ROC-AUC ")

print(f"Model: {best_model_name}")

print(f"Best Parameters: {best_model_params}")

print(f"ROC-AUC Score: {best_roc_auc:.4f}")
```

Model Deployment:

- **Preprocessing Pipeline Saved:**

  Includes OHE, scaling, and SelectKBest mapping (`preprocessor_inference.joblib`).

- **Best Model Saved:**

  LightGBM trained on top 15 features (`best_model_top15.joblib`)

Code:

```
import joblib

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler,
OneHotEncoder

from sklearn.compose import ColumnTransformer



# Define ColumnTransformer

num_cols =
X_tr.select_dtypes(include=[np.number]).columns.tolist()

cat_cols =
X_tr.select_dtypes(include=["object"]).columns.tolist()



preprocessor = ColumnTransformer(

    transformers=[

        ("num", "passthrough", num_cols),

        ("cat", OneHotEncoder(handle_unknown="ignore"),
cat_cols)

    ]

)



# Fit on training set

X_tr_enc = preprocessor.fit_transform(X_tr)

X_val_enc = preprocessor.transform(X_val)

X_test_enc = preprocessor.transform(X_test)
```

```
# Scale

scaler = StandardScaler(with_mean=False)

X_tr_scaled = scaler.fit_transform(X_tr_enc)

X_val_scaled = scaler.transform(X_val_enc)

X_test_scaled = scaler.transform(X_test_enc)


# Combine preprocessor + scaler into one object

preprocessing_pipeline = {

    "preprocessor": preprocessor,

    "scaler": scaler,

    "num_cols": num_cols,

    "cat_cols": cat_cols

}


# Save preprocessing pipeline using joblib

joblib.dump(preprocessing_pipeline,
"preprocessor.joblib")

print("Preprocessing pipeline saved as
'preprocessor.joblib'")


# Train & select best model based on ROC-AUC (from
previous GridSearchCV results)

results_df = pd.DataFrame(all_results)

results_sorted = results_df.sort_values(by="ROC-AUC",
ascending=False)


best_model_info = results_sorted.iloc[0]
```

```
best_model_name = best_model_info["Model"]

best_model_estimator =
models_params[best_model_name]["model"]



# Refit the best model on full scaled training data

best_model_estimator.set_params(**best_model_info["Best_P
arams"])

best_model_estimator.fit(X_tr_scaled, y_tr)



# Save the best model using joblib

joblib.dump(best_model_estimator, "best_model.joblib")

print(f"Best model '{best_model_name}' saved as
'best_model.joblib' "

      f"with ROC-AUC={best_model_info['ROC-AUC']:.4f}")



import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder,
StandardScaler

from sklearn.feature_selection import SelectKBest,
f_classif

import joblib

import lightgbm as lgb



# Load Data

train = pd.read_csv("train.csv")
```

```
target = "is_claim"

X_train = train.drop(columns=[target, "policy_id"])

y_train = train[target]



# Missing Values

for col in
X_train.select_dtypes(include=[np.number]).columns:

    X_train[col] =
X_train[col].fillna(X_train[col].median())



for col in
X_train.select_dtypes(include=["object"]).columns:

    X_train[col] =
X_train[col].fillna(X_train[col].mode()[0])



# Train-Val Split

X_tr, X_val, y_tr, y_val = train_test_split(

    X_train, y_train, test_size=0.2, random_state=42,
stratify=y_train

)



# Preprocessing

num_cols =
X_tr.select_dtypes(include=[np.number]).columns.tolist()

cat_cols =
X_tr.select_dtypes(include=["object"]).columns.tolist()



preprocessor = ColumnTransformer(

    transformers=[
```

```
        ("num", "passthrough", num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"),
cat_cols)
    ]
)

X_tr_enc = preprocessor.fit_transform(X_tr)
X_val_enc = preprocessor.transform(X_val)

# Scaling
scaler = StandardScaler(with_mean=False)
X_tr_scaled = scaler.fit_transform(X_tr_enc)
X_val_scaled = scaler.transform(X_val_enc)

# Feature Selection
skb = SelectKBest(score_func=f_classif, k=15)
X_tr_top15 = skb.fit_transform(X_tr_scaled, y_tr)

# Get all feature names after OHE
ohe = preprocessor.named_transformers_["cat"]
encoded_cat_features =
ohe.get_feature_names_out(cat_cols)
all_feature_names = np.concatenate([num_cols,
encoded_cat_features])

# Map OHE features back to original
feature_scores = pd.DataFrame({
```

```
    "feature": all_feature_names,
    "score": skb.scores_
})

def map_to_original(col_name):
    for cat in cat_cols:
        if col_name.startswith(cat + "_"):
            return cat
    return col_name

feature_scores["original_feature"] =
feature_scores["feature"].map(map_to_original)

# Aggregate F-score by original feature
agg_scores = (
    feature_scores.groupby("original_feature")["score"]
    .max()
    .reset_index()
    .sort_values("score", ascending=False)
)

# Top 15 ORIGINAL features
top15_original_features =
agg_scores.head(15)["original_feature"].tolist()
print("Top 15 ORIGINAL selected features:",
top15_original_features)
```

```
# Map top 15 original features to OHE columns

top15_ohe_cols = [col for col in all_feature_names

                  for orig in top15_original_features if
col.startswith(orig)]



# Train data restricted to top 15 features

X_tr_top15_model = pd.DataFrame(X_tr_scaled,
columns=all_feature_names)[top15_ohe_cols]



# Best Model LightGBM on top 15 features

best_model_top15 = lgb.LGBMClassifier(

    learning_rate=0.1,

    max_depth=20,

    n_estimators=200,

    num_leaves=50,

    random_state=42,

    class_weight='balanced'

)

best_model_top15.fit(X_tr_top15_model, y_tr)



# Save Pipeline + Model with joblib

inference_pipeline = {

    "preprocessor": preprocessor,

    "scaler": scaler,

    "selectkbest": skb,

    "all_features": all_feature_names,
```

```
    "num_cols": num_cols,

    "cat_cols": cat_cols,

    "top15_original_features": top15_original_features,

    "top15_ohe_cols": top15_ohe_cols

}


joblib.dump(inference_pipeline,
"preprocessor_inference.joblib")

joblib.dump(best_model_top15, "best_model_top15.joblib")


print("Pipeline and model saved successfully using
joblib!")
```

- **Streamlit Application:**

  Allows users to input policyholder and vehicle details. Predicts claim probability and returns binary claim/no-claim output.

  Code:

```
%%writefile car_insurance.py

import streamlit as st

import pandas as pd

import joblib


pipeline = joblib.load("preprocessor_inference.joblib")

model = joblib.load("best_model_top15.joblib")


preprocessor = pipeline["preprocessor"]

scaler = pipeline["scaler"]
```

```
all_features = pipeline["all_features"]

num_cols = pipeline["num_cols"]

cat_cols = pipeline["cat_cols"]

top15_original_features =
pipeline["top15_original_features"]

top15_ohe_cols = pipeline["top15_ohe_cols"]



# Sidebar for Navigation

st.sidebar.title("Navigation")

page = st.sidebar.radio("Go to", ["Introduction",
"Prediction"])



# Page 1: Introduction

if page == "Introduction":

    st.title("Insurance Claim Prediction Project")

    st.markdown("""

    ## Overview

    This project predicts whether a customer will make an
insurance claim or not.

    It uses a **machine learning model (LightGBM)**
trained on insurance policyholder data.



    ## Dataset

    - Includes **numeric** (age, premium, vehicle age,
etc.) and **categorical** (fuel type, region, etc.)
features.

    - Target variable: `is_claim` (Yes/No).
```

```
    ## Approach

    1. Preprocessing with **OneHotEncoding** +
**Scaling**.

    2. Feature selection using **SelectKBest (Top 15
features)**.

    3. Trained **LightGBM with class balancing**.



    ## App Functionality

    - Page 1: Project introduction.

    - Page 2: Input user details → Predict claim
probability.

    """)



# Page 2: Prediction

elif page == "Prediction":

    st.title("Insurance Claim Prediction")



    # User Input

    user_input = {}

    for feature in top15_original_features:

        if feature in num_cols:

            user_input[feature] =
st.number_input(f"{feature}", value=0.0)

        elif feature in cat_cols:

            ohe = preprocessor.named_transformers_["cat"]

            idx = cat_cols.index(feature)

            categories = ohe.categories_[idx].tolist()
```

```
            user_input[feature] =
st.selectbox(f"{feature}", options=categories)


    # Build full row for preprocessing
    row = {}
    for col in num_cols:
        row[col] = 0
    for i, col in enumerate(cat_cols):
        row[col] =
preprocessor.named_transformers_["cat"].categories_[i][0]
    row.update(user_input)
    user_df = pd.DataFrame([row])


    # Prediction
    if st.button("Predict Claim"):
        user_encoded = preprocessor.transform(user_df)
        user_scaled = scaler.transform(user_encoded)
        user_top15 = pd.DataFrame(user_scaled,
columns=all_features)[top15_ohe_cols]


        pred = model.predict(user_top15)[0]
        pred_prob = model.predict_proba(user_top15)[0][1]


        result_text = "Yes (Claim)" if pred == 1 else "No
(No Claim)"
        st.subheader("Prediction Result")
        st.write(f"Claim: **{result_text}**")
```

```
        st.write(f"Prediction Probability:
{pred_prob:.4f}")
```

Link for Deployment: http://10.58.165.222:8501

Business Insights:

Policyholder Demographics:

Policy Tenure:

The policy_tenure feature reflects the duration for which a customer has held their insurance policy. Policy tenure provides significant insights into customer loyalty and risk exposure.

- Customers with short tenures (new policyholders) may carry a higher risk, as insurers often lack historical claim behavior for them. New drivers or newly insured individuals may also be less experienced in managing policies.
- Customers with longer tenures are generally considered lower risk, as loyalty often correlates with consistent driving habits, fewer claims, and trust in the insurer.

From a business perspective, analyzing tenure helps insurers identify customer retention opportunities and implement loyalty-based discounts. Long-term customers who rarely claim can be rewarded with lower premiums, while new policyholders may need closer monitoring and slightly higher initial premiums until their driving behavior stabilizes.

Age of Policyholder:

The age_of_policyholder indicates the normalized age of the insured individual. Age is a crucial factor in assessing risk exposure.

- Younger policyholders (e.g., under 25) are often considered higher risk due to inexperience and higher likelihood of reckless driving.
- Middle-aged policyholders tend to represent the lowest risk group, balancing experience and cautious driving behavior.
- Older policyholders may face increased risk due to slower reflexes, health-related issues, or reduced driving frequency that could lead to unfamiliarity with evolving traffic patterns.

Insurance companies often integrate age into pricing models, offering reduced premiums for middle-aged groups while applying moderate adjustments for very young or elderly drivers.

Area Cluster & Population Density:

The area_cluster and population_density provide geographical context for policyholders.

- Densely populated urban areas (e.g., city clusters with high population density) are associated with higher claim probabilities, due to traffic congestion, greater accident frequency, and higher car theft rates.
- Rural or less dense areas typically present lower accident risk, but may introduce different risks such as poor road conditions.

From a business standpoint, insurers can utilize this information to geo-segment policies, offering tailored premiums based on risk levels in different regions. For instance:

- Policyholders in high-risk urban clusters may receive higher base premiums.
- Customers in low-risk rural areas may be rewarded with discounts.

This helps insurers achieve fairer pricing while controlling claim ratios.

Vehicle Characteristics:

Age of Car:

The age_of_car is a direct predictor of claim probability.

- Newer cars are less likely to experience mechanical failures and are often equipped with modern safety technology, reducing accident risks.
- Older cars may face higher maintenance issues, outdated safety systems, and greater likelihood of breakdowns, which may increase the probability of claims.

Insurers can integrate car age into their risk-based pricing models, adjusting premiums based on the expected reliability and safety of the car.

Car Make, Model, and Segment:

The make, model, and segment of a car describe its manufacturer, encoded model identifier, and classification (A, B1, B2, C1, C2, or Utility). These features carry strong business implications.

- Premium or luxury cars often attract higher repair costs, meaning even minor claims can be expensive.
- Budget cars may have lower repair costs but could lack advanced safety features, potentially increasing accident probability.
- Utility vehicles (SUVs, MUVs) may be associated with more rugged usage, higher mileage, and therefore different claim patterns.

Analysing claims across makes and segments helps insurers design segment-specific insurance plans. For example, SUVs might require higher premiums due to off-road risks, while compact cars in urban areas may need accident-prone risk adjustments.

Fuel Type:

The fuel_type (Petrol, Diesel, CNG) provides another perspective:

- Diesel cars are generally driven more for long distances, often linked to commercial use, which may increase exposure to accidents.
- CNG vehicles are cost-efficient but may have lower engine power and could be associated with specific urban driving patterns.
- Petrol vehicles are widespread, versatile, and cover a range of usage patterns.

Business-wise, insurers can link fuel type with usage patterns. For example, CNG cars in urban clusters may need specialized coverage, while diesel cars with heavy annual mileage may warrant higher premiums.

Engine Specifications:

Several engine-related features are included, such as max_torque, max_power, engine_type, displacement, cylinders, transmission_type, and gear_box.

- High-power cars are often associated with aggressive driving and higher accident risk.
- Engine displacement and cylinders reflect the vehicle's capacity; larger engines tend to correlate with higher speed and potential claim risk.
- Transmission type (manual vs. automatic) affects driving behaviour. Automatic cars are often safer in congested urban environments, while manual cars may carry slightly higher risk depending on driver skill.

From a business perspective, these specifications allow insurers to stratify vehicles by performance risk. High-performance cars may require stricter underwriting, while standard-engine vehicles may enjoy lower premiums.

Vehicle Dimensions and Weight:

Features like length, width, height, gross_weight, turning_radius, and steering_type capture physical characteristics of the vehicle.

- Larger vehicles (SUVs, sedans) may suffer more severe accidents due to size and speed but could offer higher protection to occupants.
- Smaller cars may be more vulnerable to damage but cheaper to repair.
- Gross weight directly impacts repair and claim costs—heavier cars typically incur more expensive claims.

Insurers can use these parameters to refine claim cost predictions and improve the accuracy of loss ratios.

Safety Features & Technology:

Passive Safety Features:

The dataset includes airbags, rear brakes type (Disc/Drum), and NCAP rating.

- More airbags and disc brakes are strong indicators of reduced injury severity in accidents.
- NCAP rating, being a global safety benchmark, directly correlates with lower accident fatality rates.

From a business viewpoint, vehicles with higher safety standards may qualify for reduced premiums as they lower claim likelihood and severity.

Active Safety Features:

Features like is_esc (Electronic Stability Control), is_tpms (Tire Pressure Monitoring System), is_parking_sensors, is_parking_camera, and is_brake_assist enhance driving safety.

- ESC and Brake Assist significantly reduce accident probability.
- Parking sensors and cameras lower the risk of minor accident claims in urban settings.
- TPMS ensures tire safety, minimizing blowout-related accidents.

These features empower insurers to implement feature-based discounts. For example, cars with ESC, airbags, and high NCAP ratings could receive special safety rebates.

Comfort and Convenience Features:

Other features such as adjustable steering, power door locks, central locking, power steering, fog lights, rear wiper/washer/defogger, day-night rearview mirror, ECW (Engine Check Warning), and speed alerts provide convenience but indirectly reduce risk.

- Features like ECW and speed alerts help in accident prevention by alerting drivers proactively.
- Rear visibility features reduce minor collisions in parking scenarios.

Though not as impactful as airbags or ESC, these features enhance overall risk management and can be included in differentiated product pricing.

Impact on Insurance Claims:

The is_claim target variable provides the foundation for predictive analytics. By analyzing historical claims with respect to the above features, insurers can draw valuable patterns:

- Higher claims are expected from younger policyholders with high-power cars in urban, high-density areas.
- Lower claims are likely from middle-aged drivers with standard cars, advanced safety features, and rural residence.
- Luxury and SUV owners may not claim frequently but when they do, claims tend to be expensive due to repair costs.

Thus, predictive models help insurers balance claim frequency (probability of claim) and claim severity (cost of claim).

Strategic Recommendations:

Risk-Based Premium Pricing:

- Incorporate features such as policyholder age, car age, safety ratings, and population density into dynamic pricing models.
- Provide discounts for safe cars and loyal customers to retain profitable policyholders.

Underwriting Guidelines:

- Flag high-risk profiles (young drivers, old cars, high-power engines in dense urban clusters).
- Introduce stricter documentation for luxury car insurance.

Customer Segmentation:

- Segment customers by risk score and design tailored insurance products.
- Offer usage-based insurance (e.g., lower premiums for low-mileage urban drivers).

Fraud Detection:

- Analyse unusual claim patterns (e.g., frequent claims from short-tenure policies).
- Apply machine learning models to flag anomalies.

Profitability Management:

- By combining claim frequency and severity analysis, insurers can identify profitable clusters.
- Focus marketing efforts on low-claim segments (e.g., safe drivers with mid-range cars).

Conclusion:

This project successfully developed a predictive model to estimate the likelihood of motor insurance claims based on policyholder, vehicle, and policy-related attributes. Through rigorous EDA, the analysis revealed several important behavioral and risk-related patterns:

- Younger and very senior policyholders showed a relatively higher claim frequency.
- Claims were more common for older vehicles, particularly those over 10 years of age.
- Vehicle specifications such as engine power, displacement, and fuel type demonstrated significant influence on claim likelihood.
- Regional differences indicated that urban clusters had higher claim rates, reflecting traffic density and accident exposure.
- Policy-related factors such as tenure, premium, and IDV were found to be directly linked to claim probability.

By applying feature selection and advanced machine learning models, LightGBM trained on the top 15 features delivered the best performance, striking a balance between precision and recall. This ensures that the model is not only accurate but also practical in minimizing false positives and negatives in claim prediction.

The deployment of the model as a Streamlit application provides an interactive and user-friendly platform for insurers. This enables underwriters, actuaries, and claims managers to:

- Assess risk more objectively,
- Personalize premium pricing, and
- Improve fraud detection and claim management processes.

Overall, the project demonstrates the value of data-driven decision-making in the insurance sector. By leveraging predictive analytics, insurers can achieve better risk management, operational efficiency, and customer satisfaction, ultimately driving competitive advantage in a dynamic industry.