```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.2.1-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.15.2-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in
/usr/local/lib/python3.11/dist-packages (from optuna) (2.0.39)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.11/dist-packages (from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages
(from alembic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in
/usr/local/lib/python3.11/dist-packages (from alembic>=1.5.0->optuna)
(4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.11/dist-packages (from sqlalchemy>=1.4.2-
>optuna) (3.1.1)
Downloading optuna-4.2.1-py3-none-any.whl (383 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 383.6/383.6 kB 7.7 MB/s eta
0:00:00
bic-1.15.2-py3-none-any.whl (231 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 231.9/231.9 kB 17.5 MB/s eta
0:00:00
bic, optuna
Successfully installed alembic-1.15.2 colorlog-6.9.0 optuna-4.2.1
```

```python
# ==========================================
# 1. Introduction & Setup
# ==========================================

# This notebook demonstrates an end-to-end workflow:
# EDA, data cleaning, feature engineering,
# modeling with XGBoost, hyperparameter tuning using Optuna,
# and explainability using SHAP.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# For modeling
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_squared_log_error

# XGBoost
import xgboost as xgb

# For Bayesian Optimization
import optuna

# For SHAP explainability
import shap

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 100)

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

```
Mounted at /content/drive
```

```python
train = pd.read_csv('/content/drive/My Drive/dataset/train.csv')
test = pd.read_csv('/content/drive/My Drive/dataset/test.csv')
# Check the shape
print("Train shape:", train.shape)
print("Test shape:", test.shape)
```

```
Train shape: (1200000, 21)
Test shape: (800000, 20)
```

```python
# Identify target variable and ID column
TARGET = 'Premium Amount'
ID_COL = 'id'

# ==========================================
# 3. Initial Exploration and EDA
# ==========================================

# Quick look at the data
display(train.head())
```

{"type":"dataframe"}

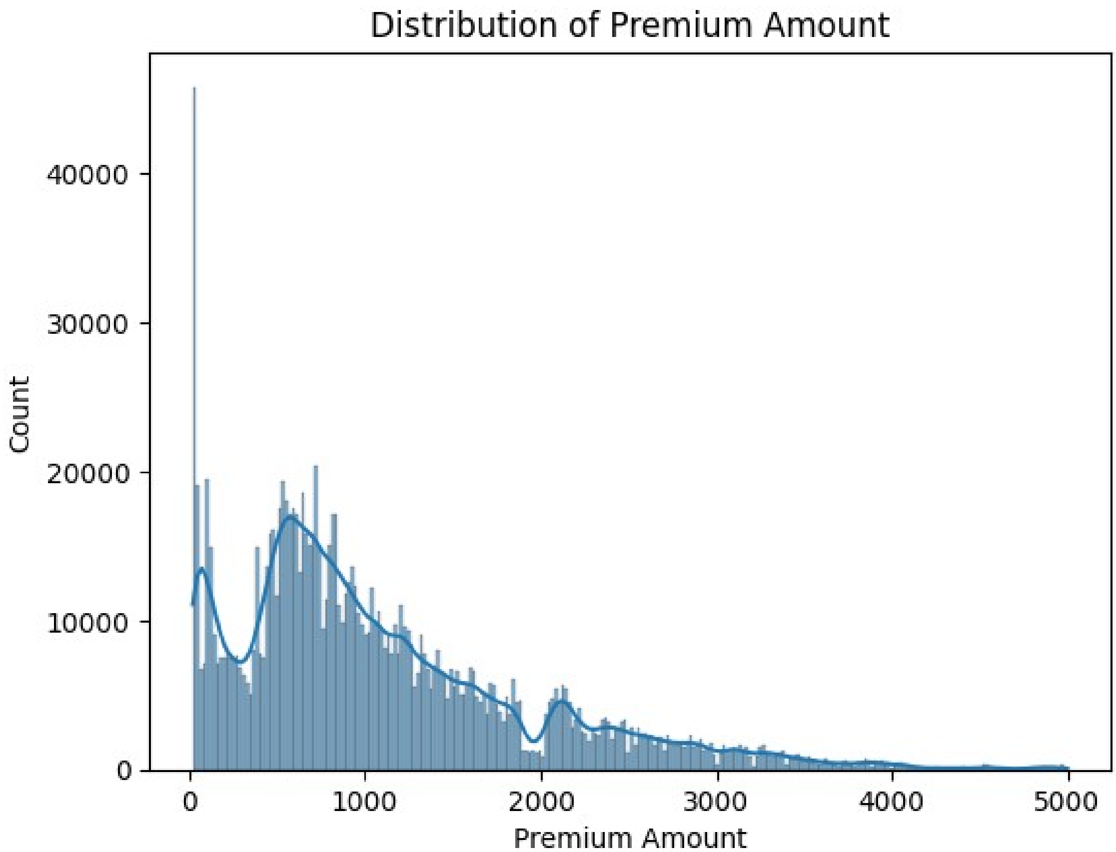
```python
display(train.describe(include='all'))
```

{"type":"dataframe"}

```python
# Check data types
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200000 entries, 0 to 1199999
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   id                    1200000 non-null  int64
 1   Age                   1181295 non-null  float64
 2   Gender                1200000 non-null  object
 3   Annual Income         1155051 non-null  float64
 4   Marital Status        1181471 non-null  object
 5   Number of Dependents  1090328 non-null  float64
 6   Education Level       1200000 non-null  object
 7   Occupation            841925 non-null   object
 8   Health Score          1125924 non-null  float64
 9   Location              1200000 non-null  object
 10  Policy Type           1200000 non-null  object
 11  Previous Claims       835971 non-null   float64
 12  Vehicle Age           1199994 non-null  float64
 13  Credit Score          1062118 non-null  float64
 14  Insurance Duration    1199999 non-null  float64
 15  Policy Start Date     1200000 non-null  object
 16  Customer Feedback     1122176 non-null  object
 17  Smoking Status        1200000 non-null  object
 18  Exercise Frequency    1200000 non-null  object
 19  Property Type         1200000 non-null  object
 20  Premium Amount        1200000 non-null  float64
dtypes: float64(9), int64(1), object(11)
memory usage: 192.3+ MB
None
```

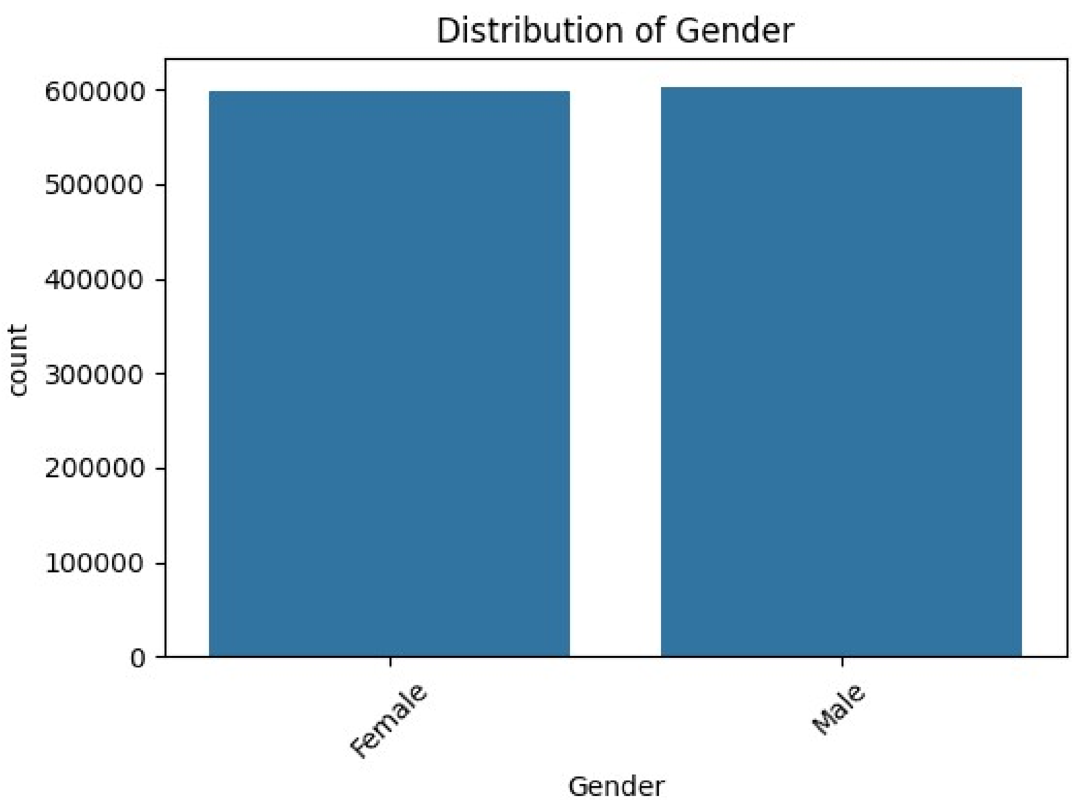
```
sns.histplot(train[TARGET], kde=True)
plt.title("Distribution of Premium Amount")
plt.show()
```

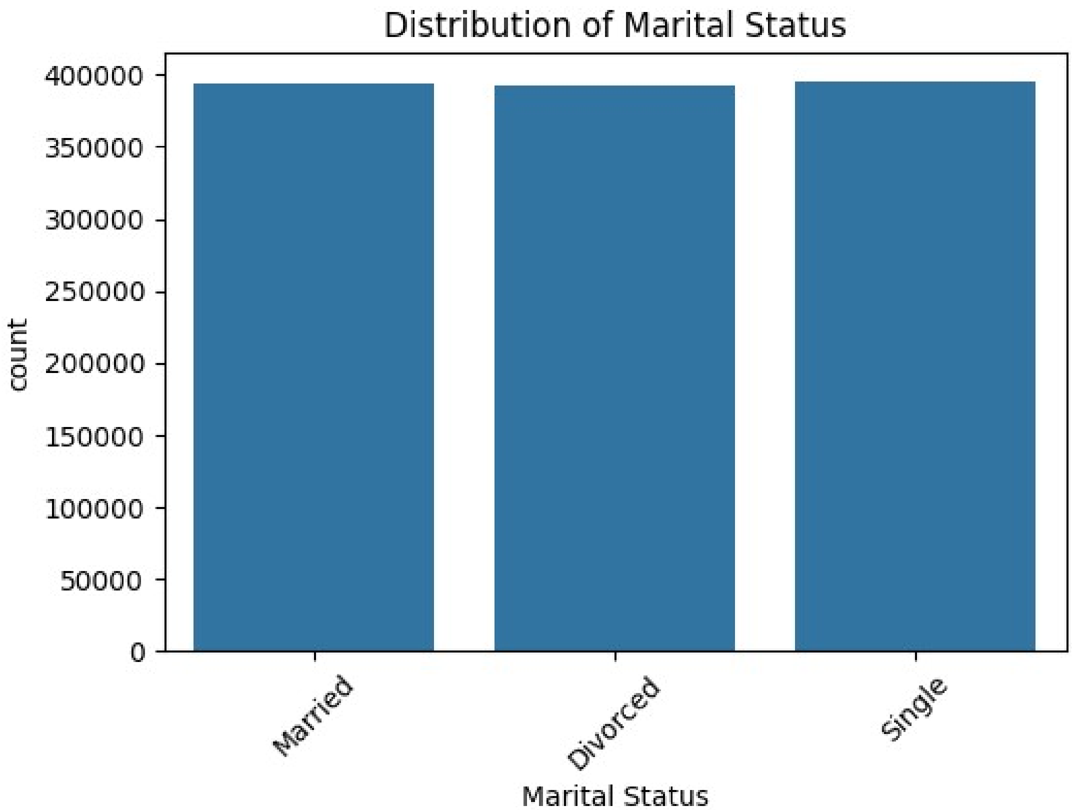


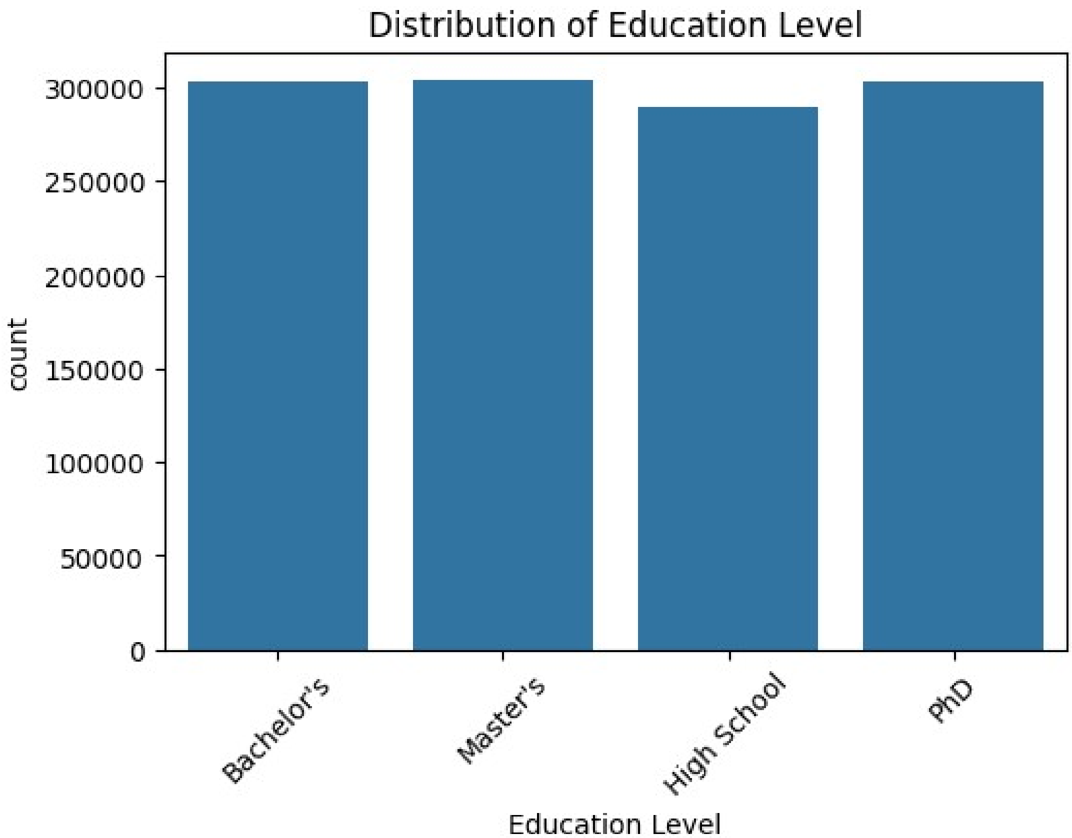

```
# Check missing values
missing_values = train.isnull().sum().sort_values(ascending=False)
print("Missing values in train:\n", missing_values)
```

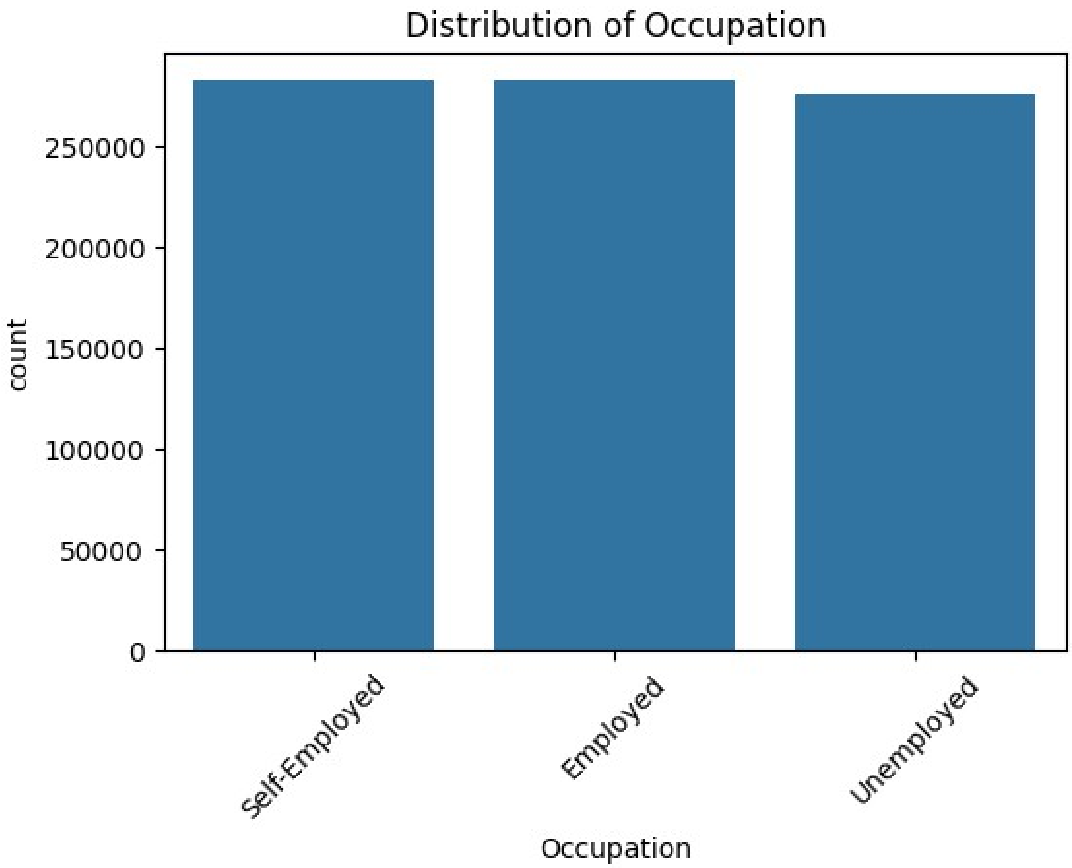
```
Missing values in train:
 Previous Claims        364029
Occupation             358075
Credit Score           137882
Number of Dependents   109672
Customer Feedback       77824
Health Score            74076
Annual Income           44949
Age                     18705
Marital Status          18529
Vehicle Age                 6
Insurance Duration          1
Gender                      0
id                          0
Location                    0
Policy Type                 0
Education Level             0
Policy Start Date           0
Smoking Status              0
```

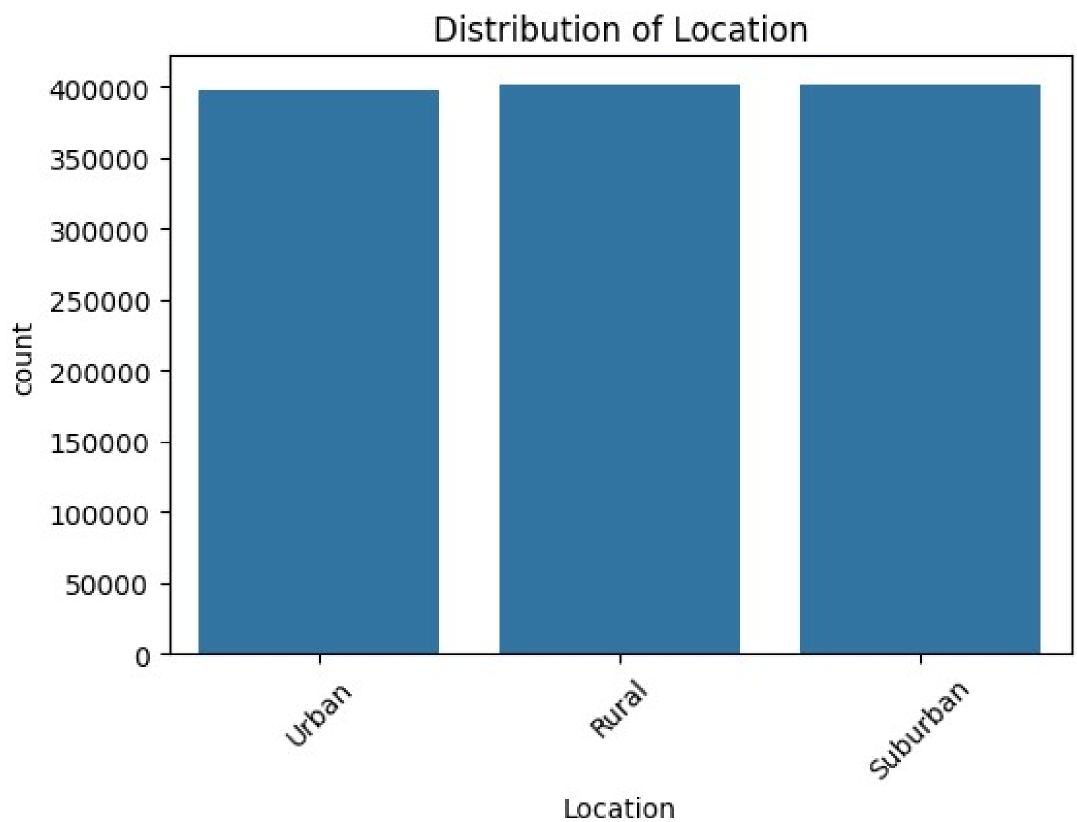
```
Exercise Frequency          0
Property Type               0
Premium Amount              0
dtype: int64

# Categorical Features distribution
cat_features = ['Gender','Marital Status','Education
Level','Occupation','Location','Policy Type','Smoking
Status','Exercise Frequency','Property Type']

for c in cat_features:
    plt.figure(figsize=(6,4))
    sns.countplot(x=c, data=train)
    plt.title(f"Distribution of {c}")
    plt.xticks(rotation=45)
    plt.show()
```

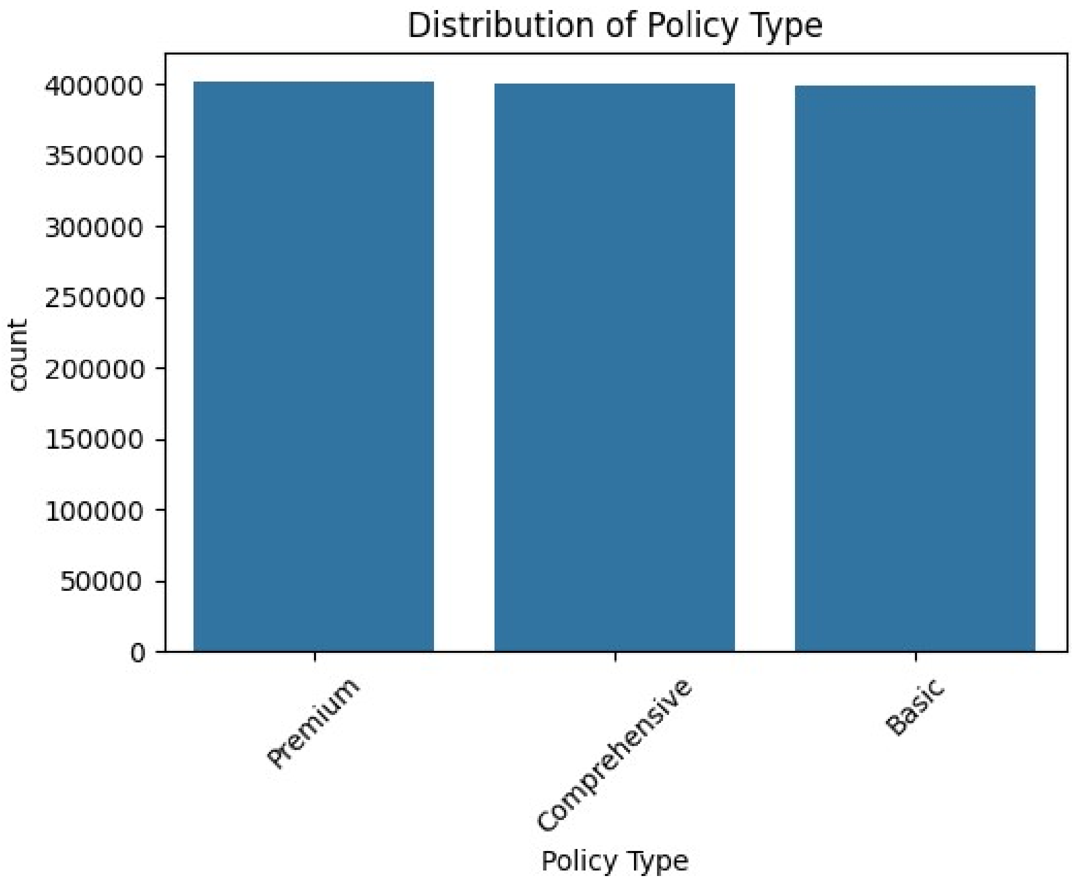


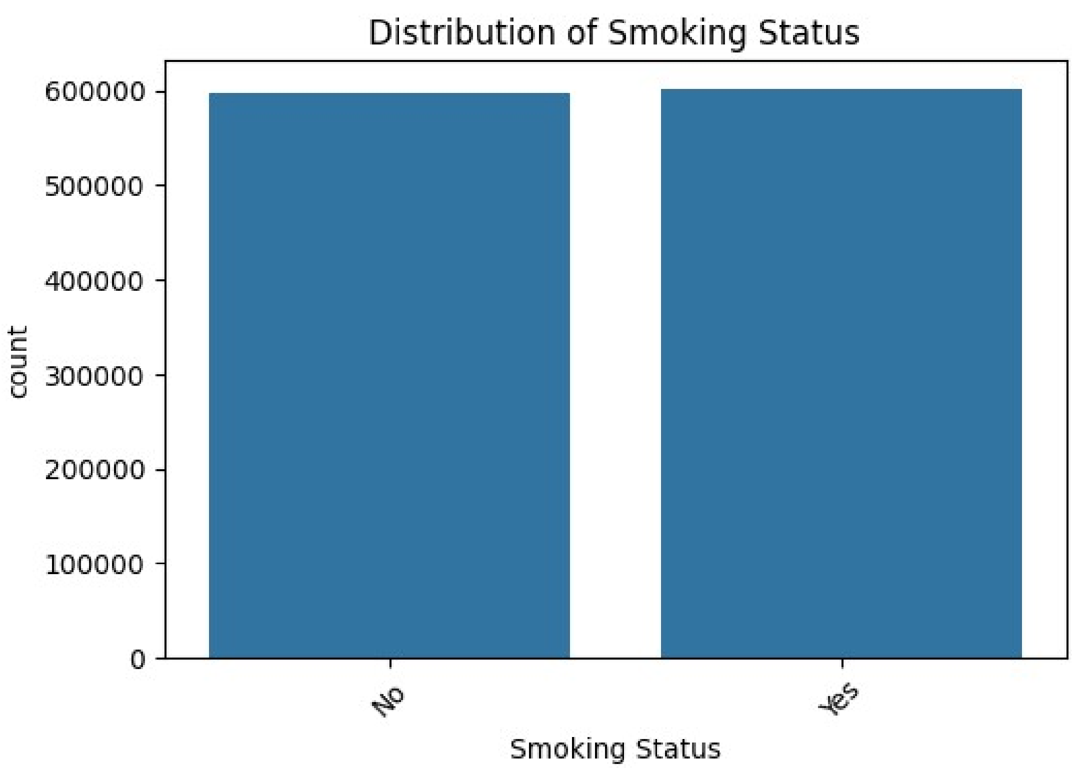

Distribution of Marital Status
400000
350000
300000
250000
200000
150000
100000
50000
0
count
Married
Divorced
Single
Marital Status

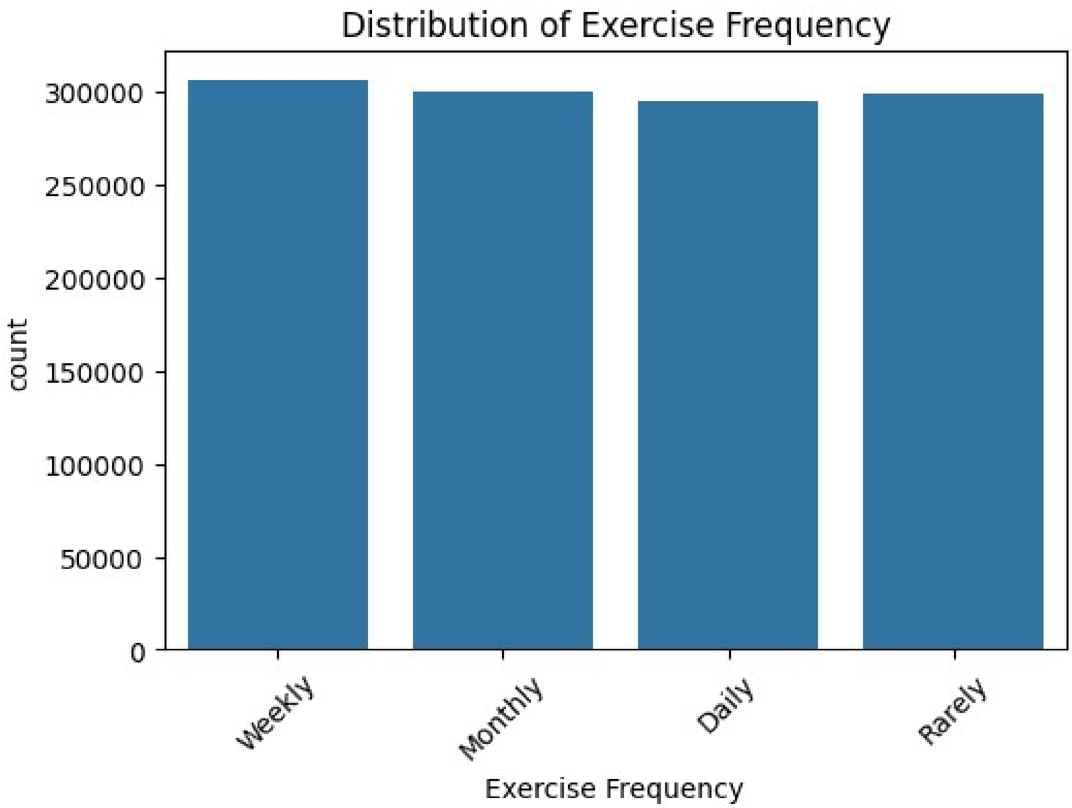

Distribution of Education Level
count
300000
250000
200000
150000
100000
50000
0
Bachelor's
Master's
High School
PhD
Education Level

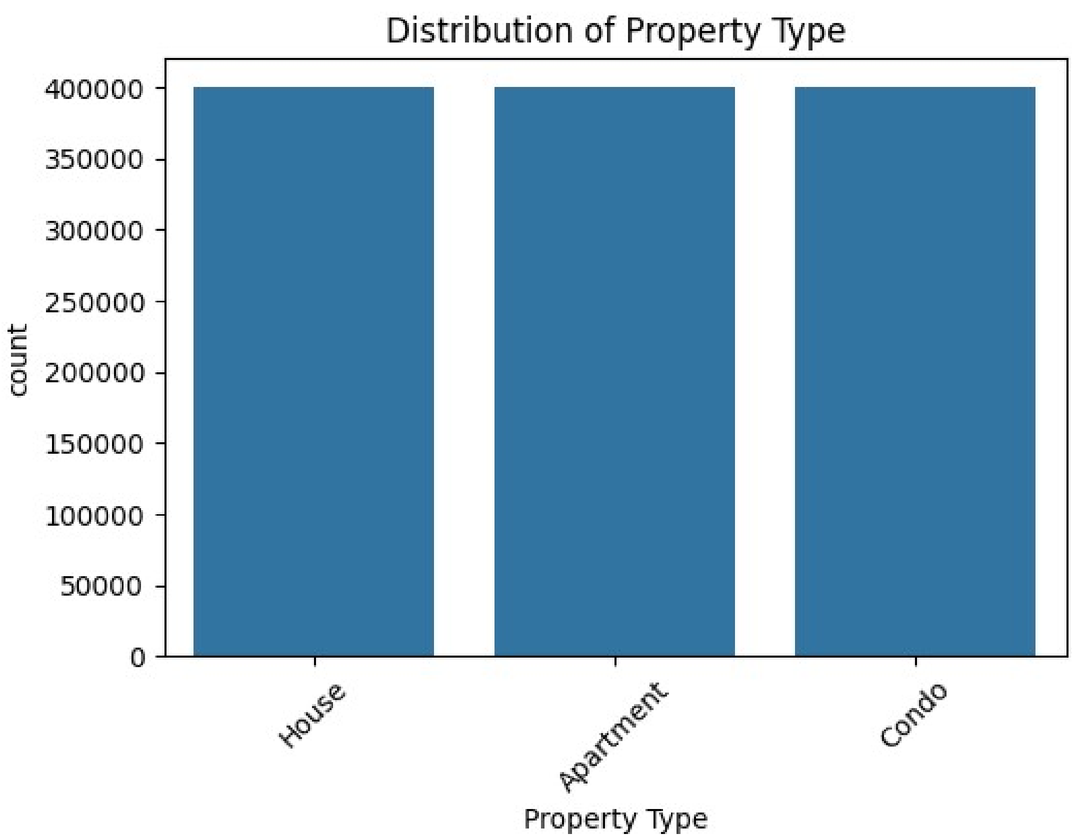

Distribution of Occupation
count
250000
200000
150000
100000
50000
0
Self-Employed
Employed
Unemployed
Occupation


Distribution of Location
400000
350000
300000
250000
200000
150000
100000
50000
0
count
Urban
Rural
Suburban
Location

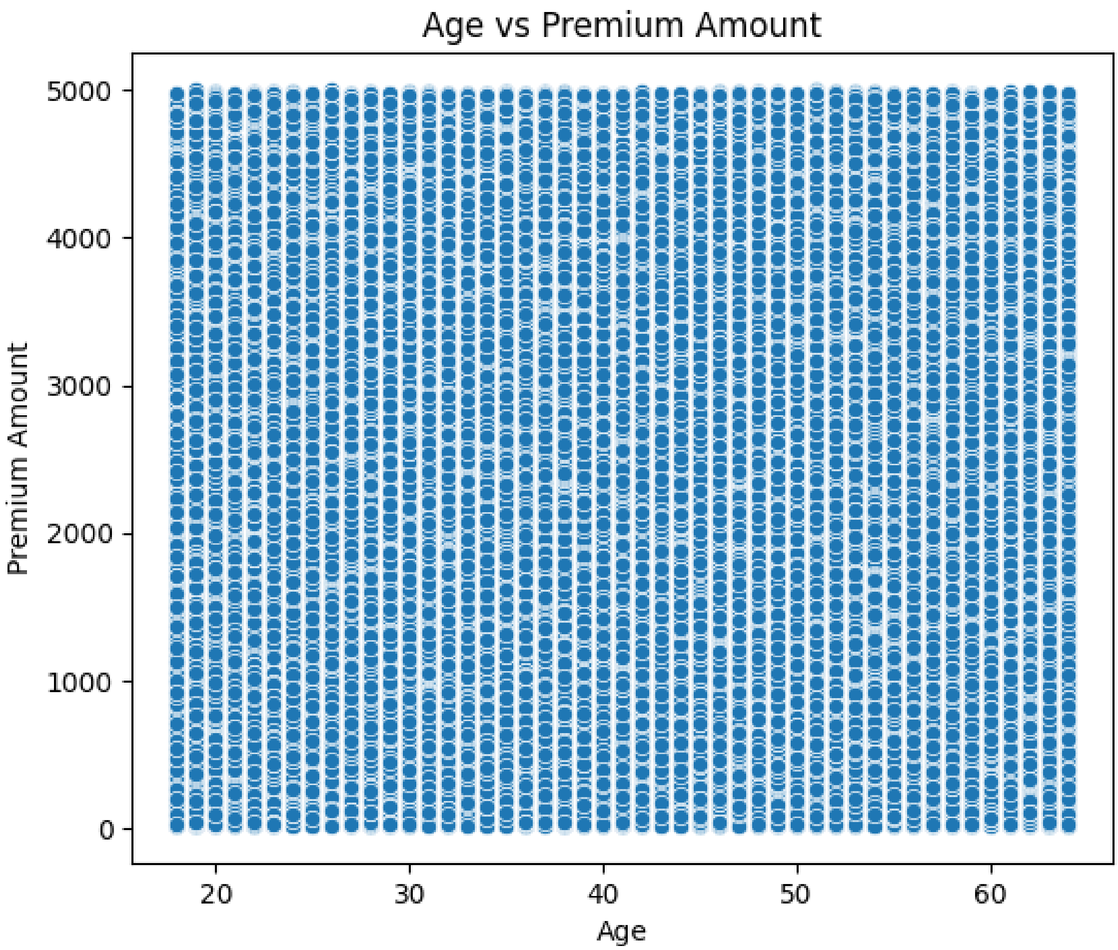

Distribution of Policy Type
count
400000
350000
300000
250000
200000
150000
100000
50000
0
Premium
Comprehensive
Basic
Policy Type

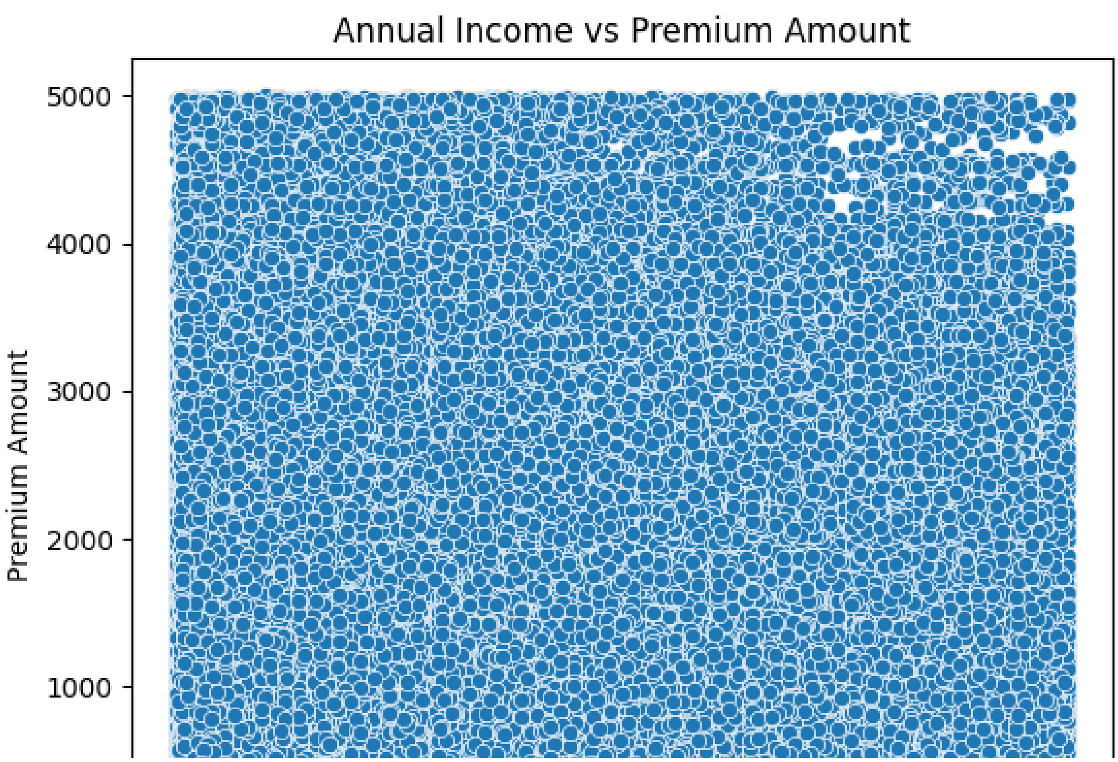

Distribution of Smoking Status
count
600000
500000
400000
300000
200000
100000
0
No
Yes
Smoking Status


Distribution of Exercise Frequency
count
300000
250000
200000
150000
100000
50000
0
Weekly
Monthly
Daily
Rarely
Exercise Frequency

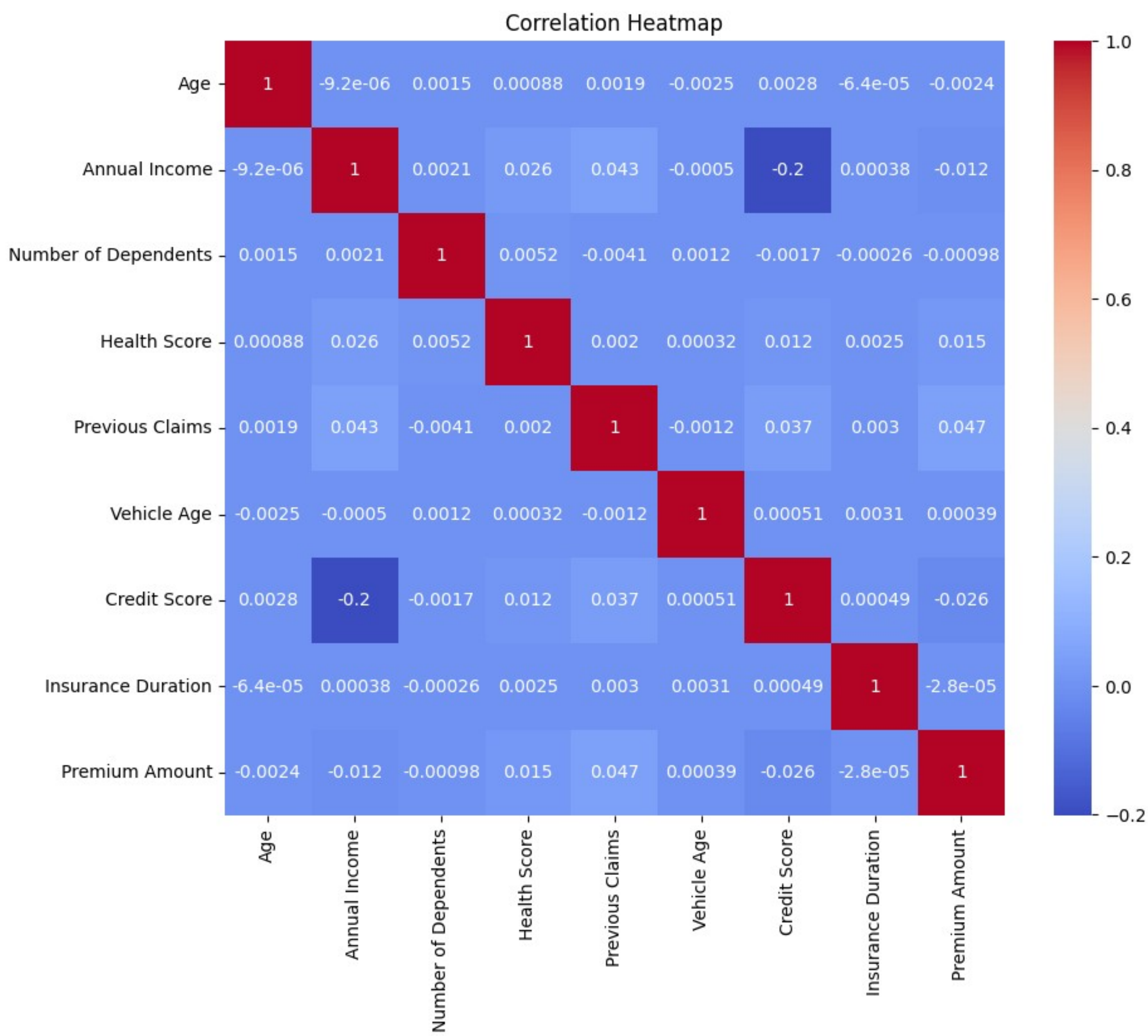

Distribution of Property Type
count
400000
350000
300000
250000
200000
150000
100000
50000
0
House
Apartment
Condo
Property Type

```
# Relationship between numeric features and target
num_features = ['Age','Annual Income','Number of Dependents','Health
Score',
                'Previous Claims','Vehicle Age','Credit
Score','Insurance Duration']
fig, axes = plt.subplots(len(num_features), 1, figsize=(6, 40))

for i, col in enumerate(num_features):
    sns.scatterplot(x=train[col], y=train[TARGET], ax=axes[i])
    axes[i].set_title(f"{col} vs {TARGET}")
plt.tight_layout()
plt.show()
```


Age vs Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
20
30
40
50
60
Age


Annual Income vs Premium Amount
Premium Amount
5000
4000
3000
2000
1000

```python
# Check correlation among numerical features
corr = train[num_features+[TARGET]].corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```




```python
# ==========================================
# 4. Data Cleaning and Preprocessing
# ==========================================

# Example of cleaning steps:
# - Fix incorrect data types
# Convert Policy Start Date to datetime if present and not in correct
format
```

```python
train['Policy Start Date'] = pd.to_datetime(train['Policy Start
Date'], errors='coerce')
test['Policy Start Date'] = pd.to_datetime(test['Policy Start Date'],
errors='coerce')
columns_with_missing = ['Previous Claims', 'Occupation', 'Credit
Score',
                        'Number of Dependents', 'Customer Feedback',
                        'Health Score', 'Annual Income', 'Age',
                        'Marital Status', 'Vehicle Age', 'Insurance
Duration'
                       ]

for col in columns_with_missing:
    # Create a new binary column indicating missingness
    train[col+'_missing'] = train[col].isnull().astype(int)
    test[col+'_missing'] = test[col].isnull().astype(int)
print(train.isnull().sum())
```

```
id                                   0
Age                              18705
Gender                               0
Annual Income                    44949
Marital Status                   18529
Number of Dependents            109672
Education Level                      0
Occupation                      358075
Health Score                     74076
Location                             0
Policy Type                          0
Previous Claims                 364029
Vehicle Age                          6
Credit Score                    137882
Insurance Duration                   1
Policy Start Date                    0
Customer Feedback                77824
Smoking Status                       0
Exercise Frequency                   0
Property Type                        0
Premium Amount                       0
Previous Claims_missing              0
Occupation_missing                   0
Credit Score_missing                 0
Number of Dependents_missing         0
Customer Feedback_missing            0
Health Score_missing                 0
Annual Income_missing                0
Age_missing                          0
Marital Status_missing               0
Vehicle Age_missing                  0
```

```
Insurance Duration_missing          0
dtype: int64
```

```
# Address skewed features: 'Annual Income', 'Premium Amount', 'Health
Score' may be skewed
# Apply log transform to reduce skewness if needed (only to non-
negative features)
# We will be careful with transforming the target for RMSLE
evaluation.

train['Annual Income'] = np.log1p(train['Annual Income'])
test['Annual Income'] = np.log1p(test['Annual Income'])

train['Health Score'] = np.log1p(train['Health Score'])
test['Health Score'] = np.log1p(test['Health Score'])
# Consider transforming the target:
# RMSLE is usually applied to positive targets. We can still predict
on normal scale,
# but optimizing a model on a log-transformed target often helps.
train[TARGET] = np.log1p(train[TARGET])  # log-transform the target
for modeling
# ========================================
# 5. Feature Engineering
# ========================================

# Example feature engineering:
# - Extract date features from 'Policy Start Date'
train['Policy_Year'] = train['Policy Start Date'].dt.year
train['Policy_Month'] = train['Policy Start Date'].dt.month
train['Policy_Day'] = train['Policy Start Date'].dt.day

test['Policy_Year'] = test['Policy Start Date'].dt.year
test['Policy_Month'] = test['Policy Start Date'].dt.month
test['Policy_Day'] = test['Policy Start Date'].dt.day
# Drop original date column if it's no longer needed
train.drop(['Policy Start Date'], axis=1, inplace=True)
test.drop(['Policy Start Date'], axis=1, inplace=True)

# Text feature: 'Customer Feedback'
# For example, we could do a length count or number of words as a
simple feature
train['Feedback_Length'] = train['Customer
Feedback'].astype(str).apply(lambda x: len(x))
test['Feedback_Length'] = test['Customer
Feedback'].astype(str).apply(lambda x: len(x))

train['Feedback_WordCount'] = train['Customer
Feedback'].astype(str).apply(lambda x: len(x.split()))
test['Feedback_WordCount'] = test['Customer
Feedback'].astype(str).apply(lambda x: len(x.split()))
```

```
# Drop the raw text if we prefer not to use it directly
train.drop(['Customer Feedback'], axis=1, inplace=True)
test.drop(['Customer Feedback'], axis=1, inplace=True)

# For categorical variables, we will use one-hot or label encoding
# Let's use one-hot encoding for simplicity
all_data = pd.concat([train, test], sort=False)
all_data = pd.get_dummies(all_data, columns=cat_features,
drop_first=True)

# Now split back into train and test
train = all_data.iloc[:train.shape[0], :]
test = all_data.iloc[train.shape[0]:, :]
train.shape
```

```
(1200000, 44)
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1200000 entries, 0 to 1199999
Data columns (total 44 columns):
 #   Column                        Non-Null Count    Dtype
---  ------                        --------------    -----
 0   id                            1200000 non-null  int64
 1   Age                           1181295 non-null  float64
 2   Annual Income                 1155051 non-null  float64
 3   Number of Dependents          1090328 non-null  float64
 4   Health Score                  1125924 non-null  float64
 5   Previous Claims               835971 non-null   float64
 6   Vehicle Age                   1199994 non-null  float64
 7   Credit Score                  1062118 non-null  float64
 8   Insurance Duration            1199999 non-null  float64
 9   Premium Amount                1200000 non-null  float64
 10  Previous Claims_missing       1200000 non-null  int64
 11  Occupation_missing            1200000 non-null  int64
 12  Credit Score_missing          1200000 non-null  int64
 13  Number of Dependents_missing  1200000 non-null  int64
 14  Customer Feedback_missing     1200000 non-null  int64
 15  Health Score_missing          1200000 non-null  int64
 16  Annual Income_missing         1200000 non-null  int64
 17  Age_missing                   1200000 non-null  int64
 18  Marital Status_missing        1200000 non-null  int64
 19  Vehicle Age_missing           1200000 non-null  int64
 20  Insurance Duration_missing    1200000 non-null  int64
 21  Policy_Year                   1200000 non-null  int32
 22  Policy_Month                  1200000 non-null  int32
 23  Policy_Day                    1200000 non-null  int32
 24  Feedback_Length               1200000 non-null  int64
 25  Feedback_WordCount            1200000 non-null  int64
```

```
 26  Gender_Male                     1200000 non-null  bool
 27  Marital Status_Married          1200000 non-null  bool
 28  Marital Status_Single           1200000 non-null  bool
 29  Education Level_High School     1200000 non-null  bool
 30  Education Level_Master's        1200000 non-null  bool
 31  Education Level_PhD             1200000 non-null  bool
 32  Occupation_Self-Employed        1200000 non-null  bool
 33  Occupation_Unemployed           1200000 non-null  bool
 34  Location_Suburban               1200000 non-null  bool
 35  Location_Urban                  1200000 non-null  bool
 36  Policy Type_Comprehensive       1200000 non-null  bool
 37  Policy Type_Premium             1200000 non-null  bool
 38  Smoking Status_Yes              1200000 non-null  bool
 39  Exercise Frequency_Monthly      1200000 non-null  bool
 40  Exercise Frequency_Rarely       1200000 non-null  bool
 41  Exercise Frequency_Weekly       1200000 non-null  bool
 42  Property Type_Condo             1200000 non-null  bool
 43  Property Type_House             1200000 non-null  bool
dtypes: bool(18), float64(9), int32(3), int64(14)
memory usage: 254.1 MB
```

```
# Make sure target is still in train
# We moved our target earlier, so it's safe. Just reconfirm structure
y = train[TARGET]
X = train.drop([TARGET, ID_COL], axis=1)
X_test = test.drop([TARGET, ID_COL], axis=1, errors='ignore') # test
doesn't have target

print("Final training shape:", X.shape)
print("Final test shape:", X_test.shape)
```

```
Final training shape: (1200000, 42)
Final test shape: (800000, 42)
```

```
# ==========================================
# 6. Train-Validation Split (Local)
# ==========================================
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
# ==========================================
# 7. Baseline Modeling with XGBoost
# ==========================================

def rmsle(y_true, y_pred):
    return np.sqrt(mean_squared_log_error(y_true, y_pred))

# Preliminary model
xgb_reg = xgb.XGBRegressor(random_state=42)
xgb_reg.fit(X_train, y_train)
```

```python
y_pred_val = xgb_reg.predict(X_val)
val_score = rmsle(np.expm1(y_val), np.expm1(y_pred_val))  # transform
back the exponent
print("Baseline RMSLE on validation:", val_score)
```

```
Baseline RMSLE on validation: 1.0480830223536157
```

```
!pip install xgboost --upgrade
```

```
Requirement already satisfied: xgboost in
/usr/local/lib/python3.11/dist-packages (2.1.4)
Collecting xgboost
  Downloading xgboost-3.0.0-py3-none-
manylinux_2_28_x86_64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
Downloading xgboost-3.0.0-py3-none-manylinux_2_28_x86_64.whl (253.9
MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 253.9/253.9 MB 1.7 MB/s eta
0:00:00
pting uninstall: xgboost
    Found existing installation: xgboost 2.1.4
    Uninstalling xgboost-2.1.4:
      Successfully uninstalled xgboost-2.1.4
Successfully installed xgboost-3.0.0
```

```python
def objective(trial):
    # Hyperparameters
    params = {
        'verbosity': 0,
        'objective': 'reg:squarederror',
        'random_state': 42,
        'tree_method': 'auto',
        'learning_rate': trial.suggest_float('learning_rate', 0.01,
0.3),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree',
0.5, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0, 10),
        'reg_lambda': trial.suggest_float('reg_lambda', 1, 10)
    }
    # Use n_estimators as num_boost_round
    num_boost_round = trial.suggest_int('n_estimators', 100, 1000)

    # Using K-Fold for evaluation
```

```python
    kf = KFold(n_splits=3, shuffle=True, random_state=42)
    rmsle_scores = []
    for train_idx, val_idx in kf.split(X, y):
        X_tr, X_vl = X.iloc[train_idx], X.iloc[val_idx]
        y_tr, y_vl = y.iloc[train_idx], y.iloc[val_idx]

        # Convert training and validation data to DMatrix
        dtrain = xgb.DMatrix(X_tr, label=y_tr)
        dvalid = xgb.DMatrix(X_vl, label=y_vl)

        evals = [(dvalid, 'eval')]
        model = xgb.train(
            params,
            dtrain,
            num_boost_round=num_boost_round,
            evals=evals,
            early_stopping_rounds=50,
            verbose_eval=False
        )

        # Use iteration_range to limit predictions to the best
iteration found
        preds = model.predict(dvalid, iteration_range=(0,
model.best_iteration))
        fold_score = rmsle(np.expm1(y_vl), np.expm1(preds))
        rmsle_scores.append(fold_score)

    return np.mean(rmsle_scores)

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=20, show_progress_bar=True)

print("Best RMSLE:", study.best_value)
print("Best parameters:", study.best_params)
```

```
[I 2025-03-30 10:00:34,934] A new study created in memory with name:
no-name-7bde1c7e-5789-4325-beee-222e37ddccb8
```


```
{"model_id":"85f6af606ddc4261bfec3d7a582b171f","version_major":2,"vers
ion_minor":0}
```


```
[I 2025-03-30 10:02:12,492] Trial 0 finished with value:
1.048235334532411 and parameters: {'learning_rate':
0.23203082447710996, 'max_depth': 9, 'subsample': 0.6237072460344051,
'colsample_bytree': 0.5068556910345523, 'reg_alpha':
9.694805650195118, 'reg_lambda': 9.176105542581137, 'n_estimators':
849}. Best is trial 0 with value: 1.048235334532411.
[I 2025-03-30 10:12:40,315] Trial 1 finished with value:
1.0456073913314434 and parameters: {'learning_rate':
0.012024093904223494, 'max_depth': 7, 'subsample': 0.7838765874041749,
'colsample_bytree': 0.927482895613863, 'reg_alpha': 8.468488987720114,
```

```
'reg_lambda': 5.9574121167245595, 'n_estimators': 781}. Best is trial
1 with value: 1.0456073913314434.
[I 2025-03-30 10:18:06,733] Trial 2 finished with value:
1.048234865828527 and parameters: {'learning_rate':
0.023763144827529996, 'max_depth': 4, 'subsample': 0.7803521610097062,
'colsample_bytree': 0.942639837613334, 'reg_alpha': 9.453407232200286,
'reg_lambda': 2.535377425075684, 'n_estimators': 626}. Best is trial 1
with value: 1.0456073913314434.
[I 2025-03-30 10:19:26,434] Trial 3 finished with value:
1.0470734203585879 and parameters: {'learning_rate':
0.18019822443841535, 'max_depth': 9, 'subsample': 0.5436187735708686,
'colsample_bytree': 0.946885752362935, 'reg_alpha': 3.193267372447232,
'reg_lambda': 5.334110751630746, 'n_estimators': 803}. Best is trial 1
with value: 1.0456073913314434.
[I 2025-03-30 10:20:30,549] Trial 4 finished with value:
1.0464858733921305 and parameters: {'learning_rate':
0.24999666168009047, 'max_depth': 7, 'subsample': 0.5597546939460907,
'colsample_bytree': 0.916850944702005, 'reg_alpha': 8.34825108425107,
'reg_lambda': 1.153524522309962, 'n_estimators': 691}. Best is trial 1
with value: 1.0456073913314434.
[I 2025-03-30 10:23:02,278] Trial 5 finished with value:
1.0465248412629464 and parameters: {'learning_rate':
0.06357908253575201, 'max_depth': 11, 'subsample': 0.679039081531783,
'colsample_bytree': 0.9538516040129981, 'reg_alpha':
8.652871321468215, 'reg_lambda': 9.68303359835292, 'n_estimators':
321}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:24:56,979] Trial 6 finished with value:
1.0458570713313275 and parameters: {'learning_rate':
0.07053035441183415, 'max_depth': 8, 'subsample': 0.7158665166968257,
'colsample_bytree': 0.9722505575668283, 'reg_alpha':
1.2459883024076768, 'reg_lambda': 3.138091674567455, 'n_estimators':
814}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:27:47,520] Trial 7 finished with value:
1.0478220612675233 and parameters: {'learning_rate':
0.06796855171253151, 'max_depth': 11, 'subsample': 0.546028109325027,
'colsample_bytree': 0.7164881637425577, 'reg_alpha':
1.5053414339104276, 'reg_lambda': 5.426929591020721, 'n_estimators':
526}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:32:15,479] Trial 8 finished with value:
1.0469702675464239 and parameters: {'learning_rate':
0.08598792989950434, 'max_depth': 5, 'subsample': 0.9564136245249774,
'colsample_bytree': 0.5658787030607828, 'reg_alpha':
2.2629905576625253, 'reg_lambda': 3.3263837168112746, 'n_estimators':
747}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:36:07,511] Trial 9 finished with value:
1.0458441272641086 and parameters: {'learning_rate':
0.04838978907413369, 'max_depth': 8, 'subsample': 0.9871120886630691,
'colsample_bytree': 0.6893068673792815, 'reg_alpha':
9.115914411188532, 'reg_lambda': 6.401201878227335, 'n_estimators':
```

```
413}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:37:46,945] Trial 10 finished with value:
1.046218372993381 and parameters: {'learning_rate':
0.13714799216623308, 'max_depth': 6, 'subsample': 0.8392960689422029,
'colsample_bytree': 0.8362506237981608, 'reg_alpha':
6.132332497369323, 'reg_lambda': 7.424639994175929, 'n_estimators':
985}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:40:52,419] Trial 11 finished with value:
1.055181390798077 and parameters: {'learning_rate':
0.01285841117610031, 'max_depth': 3, 'subsample': 0.9747603625470846,
'colsample_bytree': 0.700362994042501, 'reg_alpha': 6.536145000208948,
'reg_lambda': 7.058111333087767, 'n_estimators': 401}. Best is trial 1
with value: 1.0456073913314434.
[I 2025-03-30 10:42:20,870] Trial 12 finished with value:
1.0459827325503557 and parameters: {'learning_rate':
0.13841133743558234, 'max_depth': 7, 'subsample': 0.8618346734370944,
'colsample_bytree': 0.7926867746201864, 'reg_alpha':
7.444487075212015, 'reg_lambda': 7.170024748936444, 'n_estimators':
488}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:43:35,298] Trial 13 finished with value:
1.0478364767005817 and parameters: {'learning_rate':
0.29365327575982425, 'max_depth': 9, 'subsample': 0.898560469232555,
'colsample_bytree': 0.6406921754941046, 'reg_alpha':
4.6292371137070365, 'reg_lambda': 6.232080613215508, 'n_estimators':
245}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:45:02,242] Trial 14 finished with value:
1.0576211209920547 and parameters: {'learning_rate':
0.010544276887474412, 'max_depth': 6, 'subsample': 0.7659018365060182,
'colsample_bytree': 0.8538530944014209, 'reg_alpha':
4.740964824805478, 'reg_lambda': 4.282946154230269, 'n_estimators':
105}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:47:42,824] Trial 15 finished with value:
1.0479770161607158 and parameters: {'learning_rate':
0.10798981037893024, 'max_depth': 12, 'subsample': 0.9986452270831843,
'colsample_bytree': 0.6736039325527651, 'reg_alpha':
7.655256533688579, 'reg_lambda': 8.914545927769565, 'n_estimators':
991}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:50:43,999] Trial 16 finished with value:
1.0456632983808518 and parameters: {'learning_rate':
0.046057970644333174, 'max_depth': 8, 'subsample': 0.908832120930388,
'colsample_bytree': 0.7746269758670896, 'reg_alpha':
9.937760023662808, 'reg_lambda': 4.8275081558038515, 'n_estimators':
606}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 10:52:39,463] Trial 17 finished with value:
1.0466210777034446 and parameters: {'learning_rate':
0.10638156278270505, 'max_depth': 10, 'subsample': 0.8092263683656474,
'colsample_bytree': 0.7779898949353766, 'reg_alpha':
9.980914512276595, 'reg_lambda': 4.431020860167472, 'n_estimators':
639}. Best is trial 1 with value: 1.0456073913314434.
```

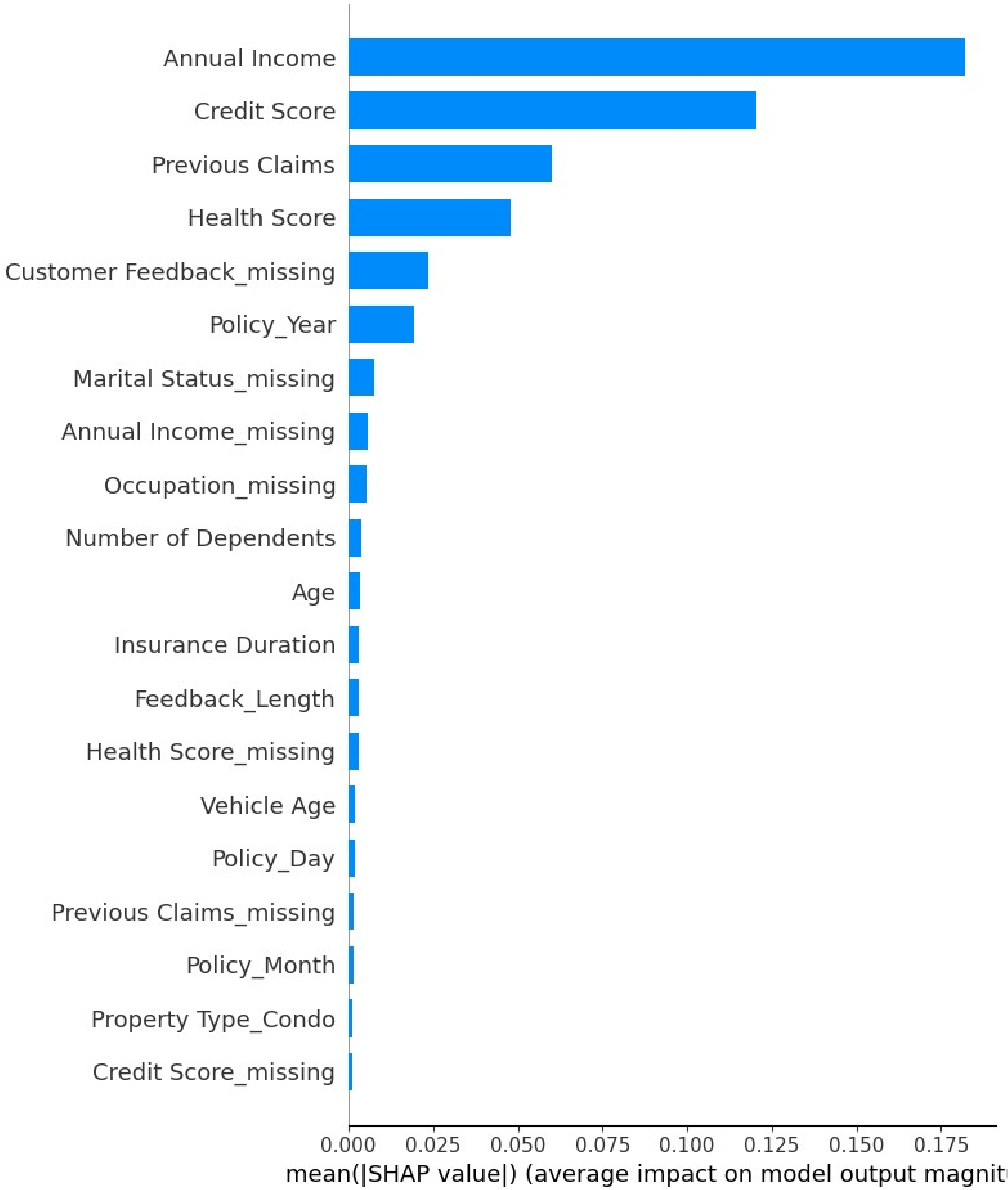
```
[I 2025-03-30 10:54:01,911] Trial 18 finished with value:
1.0464050146719268 and parameters: {'learning_rate':
0.17687102757282283, 'max_depth': 6, 'subsample': 0.8622727171392076,
'colsample_bytree': 0.8738283891439347, 'reg_alpha':
5.924727455904282, 'reg_lambda': 8.061904461166883, 'n_estimators':
604}. Best is trial 1 with value: 1.0456073913314434.
[I 2025-03-30 11:01:31,407] Trial 19 finished with value:
1.046518616938356 and parameters: {'learning_rate':
0.039323576925330006, 'max_depth': 5, 'subsample': 0.9061660030338359,
'colsample_bytree': 0.9961203860573464, 'reg_alpha':
7.371904712908288, 'reg_lambda': 4.628354788050437, 'n_estimators':
846}. Best is trial 1 with value: 1.0456073913314434.
Best RMSLE: 1.0456073913314434
Best parameters: {'learning_rate': 0.012024093904223494, 'max_depth':
7, 'subsample': 0.7838765874041749, 'colsample_bytree':
0.927482895613863, 'reg_alpha': 8.468488987720114, 'reg_lambda':
5.9574121167245595, 'n_estimators': 781}
```

```
best_params = study.best_params
final_model = xgb.XGBRegressor(**best_params, random_state=42)
final_model.fit(X, y)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.927482895613863, device=None,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=None,
grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.012024093904223494, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=781, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

```
# ==========================================
# 9. Model Evaluation using RMSLE on Validation
# ==========================================
# Already done cross-validation;
# If we have a separate test set (we do - but no target), we will just
generate predictions:
# We'll trust cross-validation results. For final submission on
Kaggle:
y_pred_test = final_model.predict(X_test)
# Remember to invert the log transform for predictions
y_pred_test = np.expm1(y_pred_test)
```

```
# ==========================================
# 10. Explainability with SHAP
# ==========================================
# Use a subset of data to speed up SHAP
explainer = shap.TreeExplainer(final_model)
shap_values = explainer.shap_values(X.sample(1000, random_state=42))
# Global Feature Importance
shap.summary_plot(shap_values, X.sample(1000, random_state=42),
plot_type='bar')
```




```
shap.summary_plot(shap_values, X.sample(1000, random_state=42))
```

Annual Income
Credit Score
Previous Claims
Health Score
Customer Feedback_missing
Policy_Year
Marital Status_missing
Annual Income_missing
Occupation_missing
Number of Dependents
Age
Insurance Duration
Feedback_Length
Health Score_missing
Vehicle Age
Policy_Day
Previous Claims_missing
Policy_Month
Property Type_Condo
Credit Score_missing
−0.6 −0.4 −0.2 0.0 0.2 0.4 0.6
SHAP value (impact on model output)
High
Low
Feature value