health-insurance-premium-inference

March 9, 2025

0.1 # Regression with an Insurance Dataset (Test Data Preprocessing + Inference)

Source: Kaggle Playground Prediction Competition

Course Title: DAMO-510-4: Winter 2025 Predictive Analytics

Professor Name: Professor Ali El-Sharif

Submission Date: March 9, 2025

Submitted By:

1. **Denisse C. Cortes (NF1007936)**
2. **Dev D. Rabadia (NF1005560)**
3. **Miko L. Tan (NF1008647)**
4. **Rosario D. Torres (NF1001385)**

The data model for this project is a regression-based model designed to predict insurance premium amounts. The model uses a variety of features, including numerical variables (such as age, annual income, and health score) and categorical variables (such as gender, marital status, and policy type) to estimate the target variable: the insurance premium amount. The dataset incorporates several features with skewed distributions and missing values, which will be handled through appropriate preprocessing techniques. Outliers in the dataset, such as those found in the "Previous Claims" feature, will also be addressed to improve model accuracy and robustness.

[1]:
```
import numpy as np
import pandas as pd
from scipy.stats import boxcox
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
import joblib
```

[2]:
```
# Load test dataset
# insurance_test_data = pd.read_csv("/kaggle/input/playground-series-s4e12/test.
 ↪csv")
insurance_test_data = pd.read_csv("kaggle/input/playground-series-s4e12/test.
 ↪csv")
```

[3]:
```
# Load the best trained model
# best_model = joblib.load("kaggle/input/hip-best-model/scikitlearn/xgb_rscv_v1.
 ↪pkl/1/health-insurance-premium-best-model.pkl")
best_model = joblib.load("kaggle/working/health-insurance-premium-best-model.
 ↪pkl")
```

[4]:
```
# #### Exploratory Data Analysis (EDA)
# missing_data = insurance_test_data.isnull().sum()
# missing_percentage = (missing_data / len(insurance_test_data)) * 100
# missing_info = pd.DataFrame({
#     'Missing Count': missing_data,
#     'Missing Percentage': missing_percentage,
# })
# def categorize_missing_data(percentage):
#     if percentage <= 5:
#         return 'Small (1-5%)'
#     elif 5 < percentage <= 20:
#         return 'Moderate (5-20%)'
#     elif 20 < percentage <= 40:
#         return 'High (20-40%)'
#     else:
#         return 'Very High (40%+)'
# missing_info['Classification'] = missing_info['Missing Percentage'].
 ↪apply(categorize_missing_data)
# missing_info = missing_info.sort_values(by='Missing Percentage',␣
 ↪ascending=False)

# #### Handling Missing Data (excluding Premium Amount)
# if missing_info.loc["Age"]["Missing Percentage"] < 5:
#     insurance_test_data.dropna(subset=["Age"], inplace=True)
# else:
#     insurance_test_data["Age"].fillna(insurance_test_data["Age"].median(),␣
 ↪inplace=True)
# No dropping of records rule for Test dataset
insurance_test_data.loc[:, "Age"] = insurance_test_data["Age"].
 ↪fillna(insurance_test_data["Age"].median())
insurance_test_data.loc[:, "Gender"] = insurance_test_data["Gender"].
 ↪fillna("Unknown")
# if missing_info.loc["Annual Income"]["Missing Percentage"] < 5:
#     insurance_test_data.dropna(subset=["Annual Income"], inplace=True)
# else:
#     insurance_test_data["Annual Income"].fillna(insurance_test_data["Annual␣
 ↪Income"].median(), inplace=True)
insurance_test_data.loc[:, "Annual Income"] = insurance_test_data["Annual␣
 ↪Income"].fillna(insurance_test_data["Annual Income"].median())
```

```
insurance_test_data.loc[:, "Marital Status"] = insurance_test_data["Marital␣
 ↪Status"].fillna("Single")
insurance_test_data.loc[:, "Number of Dependents"] =␣
 ↪insurance_test_data["Number of Dependents"].fillna(0)
insurance_test_data.loc[:, "Education Level"] = insurance_test_data["Education␣
 ↪Level"].fillna("Unknown")
insurance_test_data.loc[:, "Occupation"] = insurance_test_data["Occupation"].
 ↪fillna("Unemployed")
insurance_test_data.loc[:, "Health Score"] = insurance_test_data["Health␣
 ↪Score"].fillna(0)
insurance_test_data.loc[:, "Location"] = insurance_test_data["Location"].
 ↪fillna("Unknown")
insurance_test_data.loc[:, "Policy Type"] = insurance_test_data["Policy Type"].
 ↪fillna("Basic")
insurance_test_data.loc[:, "Previous Claims"] = insurance_test_data["Previous␣
 ↪Claims"].fillna(0)
insurance_test_data.loc[:, "Vehicle Age"] = insurance_test_data["Vehicle Age"].
 ↪fillna(0)
insurance_test_data.loc[:, "Credit Score"] = insurance_test_data["Credit␣
 ↪Score"].fillna(0)
insurance_test_data.loc[:, "Insurance Duration"] =␣
 ↪insurance_test_data["Insurance Duration"].fillna(0)
insurance_test_data.dropna(subset=["Policy Start Date"], inplace=True)
insurance_test_data.loc[:, "Customer Feedback"] = insurance_test_data["Customer␣
 ↪Feedback"].fillna("Not Provided")
insurance_test_data.loc[:, "Smoking Status"] = insurance_test_data["Smoking␣
 ↪Status"].fillna("No")
insurance_test_data.loc[:, "Exercise Frequency"] =␣
 ↪insurance_test_data["Exercise Frequency"].fillna(
    "Not Provided")
insurance_test_data.loc[:, "Property Type"] = insurance_test_data["Property␣
 ↪Type"].fillna("Not Provided")

# Feature Engineering & Transformation (excluding Premium Amount)
bins = [0, 5, 18, 25, 35, 45, 55, 65, float('inf')]  # Age bin edges
labels = ["High Risk (0-5) (Infants)", "Moderate Risk (6-18) (Children &␣
 ↪Adolescents)",
          "Low Risk (19-25) (Young Adults)",
          "Moderate-Low Risk (26-35) (Early Adulthood)", "Moderate Risk (36-45)␣
 ↪(Middle Adulthood)",
          "High Risk (46-55) (Mature Adults)", "Very High Risk (56-65)␣
 ↪(Pre-Retirement)",
          "Very High Risk (65+) (Seniors)"]
insurance_test_data["Age_Bin"] = pd.cut(insurance_test_data["Age"], bins=bins,␣
 ↪labels=labels, right=True)
age_bin_mapping = {
```

```
    "High Risk (0-5) (Infants)": 1,
    "Moderate Risk (6-18) (Children & Adolescents)": 2,
    "Low Risk (19-25) (Young Adults)": 3,
    "Moderate-Low Risk (26-35) (Early Adulthood)": 4,
    "Moderate Risk (36-45) (Middle Adulthood)": 5,
    "High Risk (46-55) (Mature Adults)": 6,
    "Very High Risk (56-65) (Pre-Retirement)": 7,
    "Very High Risk (65+) (Seniors)": 8
}
insurance_test_data["Age_Bin Numeric"] = insurance_test_data["Age_Bin"].
 ↪map(age_bin_mapping)
insurance_test_data["Age_Bin Numeric"] = insurance_test_data["Age_Bin Numeric"].
 ↪cat.codes

if "Gender" in insurance_test_data.columns:
    insurance_test_data = pd.get_dummies(insurance_test_data,␣
 ↪columns=["Gender"], drop_first=False)
if 'Gender_Male' in insurance_test_data.columns:
    insurance_test_data['Gender_Male'] = insurance_test_data['Gender_Male'].
 ↪astype(int)
if 'Gender_Female' in insurance_test_data.columns:
    insurance_test_data['Gender_Female'] = insurance_test_data['Gender_Female'].
 ↪astype(int)
if 'Gender_Unknown' in insurance_test_data.columns:
    insurance_test_data['Gender_Unknown'] =␣
 ↪insurance_test_data['Gender_Unknown'].astype(int)
income_bins = [0, 25000, 50000, 100000, 150000, float('inf')]  # Define bin␣
 ↪edges
income_labels = ["Low Income (0-25k)", "Lower-Middle Income (25k-50k)",
                 "Middle Income (50k-100k)", "Upper-Middle Income (100k-150k)",
                 "High Income (150k+)"]

insurance_test_data["Income_Bin"] = pd.cut(insurance_test_data["Annual␣
 ↪Income"], bins=income_bins,
                                           labels=income_labels, right=False)
income_bin_mapping = {
    "Low Income (0-25k)": 1,
    "Lower-Middle Income (25k-50k)": 2,
    "Middle Income (50k-100k)": 3,
    "Upper-Middle Income (100k-150k)": 4,
    "High Income (150k+)": 5
}
insurance_test_data["Income_Bin Numeric"] = insurance_test_data["Income_Bin"].
 ↪map(income_bin_mapping)
insurance_test_data["Income_Bin Numeric"] = insurance_test_data["Income_Bin␣
 ↪Numeric"].cat.codes
```

```
if "Marital Status" in insurance_test_data.columns:
    insurance_test_data = pd.get_dummies(insurance_test_data, columns=["Marital␣
 ↪Status"], drop_first=False)

label_encoder = LabelEncoder()
desired_order = ["Unknown", "High School", "Bachelor's", "Master's", "PhD"]
label_encoder.classes_ = np.array(desired_order)
insurance_test_data['Education Level_Encoded'] = label_encoder.
 ↪transform(insurance_test_data['Education Level'])

if "Occupation" in insurance_test_data.columns:
    insurance_test_data['Occupation'] = insurance_test_data['Occupation'].
 ↪replace({
        'Employed': 'Employed/Self-Employed',
        'Self-Employed': 'Employed/Self-Employed'
    })
    insurance_test_data = pd.get_dummies(insurance_test_data,␣
 ↪columns=['Occupation'])
if 'Occupation_Employed/Self-Employed' in insurance_test_data.columns:
    insurance_test_data['Occupation_Employed/Self-Employed'] =␣
 ↪insurance_test_data[
        'Occupation_Employed/Self-Employed'].astype(int)
if 'Occupation_Unemployed' in insurance_test_data.columns:
    insurance_test_data['Occupation_Unemployed'] =␣
 ↪insurance_test_data['Occupation_Unemployed'].astype(int)

if "Location" in insurance_test_data.columns:
    insurance_test_data = pd.get_dummies(insurance_test_data,␣
 ↪columns=["Location"], drop_first=False)
if 'Location_Suburban' in insurance_test_data.columns:
    insurance_test_data['Location_Suburban'] =␣
 ↪insurance_test_data['Location_Suburban'].astype(int)
if 'Location_Rural' in insurance_test_data.columns:
    insurance_test_data['Location_Rural'] =␣
 ↪insurance_test_data['Location_Rural'].astype(int)
if 'Location_Urban' in insurance_test_data.columns:
    insurance_test_data['Location_Urban'] =␣
 ↪insurance_test_data['Location_Urban'].astype(int)
if 'Location_Unknown' in insurance_test_data.columns:
    insurance_test_data['Location_Unknown'] =␣
 ↪insurance_test_data['Location_Unknown'].astype(int)

if "Policy Type" in insurance_test_data.columns:
    # Perform one-hot encoding for "Policy Type" if it hasn't been done already
```

```
    insurance_test_data = pd.get_dummies(insurance_test_data, columns=["Policy␣
 ↪Type"], drop_first=False)
if 'Policy Type_Basic' in insurance_test_data.columns:
    insurance_test_data['Policy Type_Basic'] = insurance_test_data['Policy␣
 ↪Type_Basic'].astype(int)
if 'Policy Type_Comprehensive' in insurance_test_data.columns:
    insurance_test_data['Policy Type_Comprehensive'] =␣
 ↪insurance_test_data['Policy Type_Comprehensive'].astype(
        int)
if 'Policy Type_Premium' in insurance_test_data.columns:
    insurance_test_data['Policy Type_Premium'] = insurance_test_data['Policy␣
 ↪Type_Premium'].astype(int)

skewed_features = ['Previous Claims']
insurance_test_data[skewed_features] = insurance_test_data[skewed_features].
 ↪apply(
    lambda x: x + 1 if (x <= 0).any() else x)
for col in skewed_features:
    insurance_test_data[col], _ = boxcox(insurance_test_data[col])

insurance_test_data['Policy Start Date'] = pd.
 ↪to_datetime(insurance_test_data['Policy Start Date'],
                                                    errors='coerce')
insurance_test_data['Policy Start Year'] = insurance_test_data['Policy Start␣
 ↪Date'].dt.year
insurance_test_data['Policy Start Month'] = insurance_test_data['Policy Start␣
 ↪Date'].dt.month
insurance_test_data['Years Since Start'] = (pd.to_datetime('today') -␣
 ↪insurance_test_data[
    'Policy Start Date']).dt.days / 365
insurance_test_data.drop(columns=["Policy Start Date"], inplace=True)

if "Customer Feedback" in insurance_test_data.columns:
    insurance_test_data = pd.get_dummies(insurance_test_data,␣
 ↪columns=["Customer Feedback"], drop_first=False)
if 'Customer Feedback_Poor' in insurance_test_data.columns:
    insurance_test_data['Customer Feedback_Poor'] =␣
 ↪insurance_test_data['Customer Feedback_Poor'].astype(int)
if 'Customer Feedback_Average' in insurance_test_data.columns:
    insurance_test_data['Customer Feedback_Average'] =␣
 ↪insurance_test_data['Customer Feedback_Average'].astype(
        int)
if 'Customer Feedback_Good' in insurance_test_data.columns:
    insurance_test_data['Customer Feedback_Good'] =␣
 ↪insurance_test_data['Customer Feedback_Good'].astype(int)
if 'Customer Feedback_Not Provided' in insurance_test_data.columns:
```

```
    insurance_test_data['Customer Feedback_Not Provided'] = insurance_test_data[
        'Customer Feedback_Not Provided'].astype(int)

insurance_test_data['Smoking Status'] = insurance_test_data['Smoking Status'].
 ↪map({'No': 0, 'Yes': 1})
label_encoder = LabelEncoder()
insurance_test_data['Smoking Status'] = label_encoder.
 ↪fit_transform(insurance_test_data['Smoking Status'])

label_encoder = LabelEncoder()
desired_order = ["Not Provided", "Rarely", "Daily", "Weekly", "Monthly"]
label_encoder.classes_ = np.array(desired_order)
insurance_test_data['Exercise Frequency Encoded'] = label_encoder.transform(
    insurance_test_data['Exercise Frequency'])

if "Property Type" in insurance_test_data.columns:
    insurance_test_data = pd.get_dummies(insurance_test_data,␣
 ↪columns=["Property Type"], drop_first=False)
if 'Property Type_House' in insurance_test_data.columns:
    insurance_test_data['Property Type_House'] = insurance_test_data['Property␣
 ↪Type_House'].astype(int)
if 'Property Type_Apartment' in insurance_test_data.columns:
    insurance_test_data['Property Type_Apartment'] =␣
 ↪insurance_test_data['Property Type_Apartment'].astype(int)
if 'Property Type_Condo' in insurance_test_data.columns:
    insurance_test_data['Property Type_Condo'] = insurance_test_data['Property␣
 ↪Type_Condo'].astype(int)
if 'Property Type_Unknown' in insurance_test_data.columns:
    insurance_test_data['Property Type_Unknown'] =␣
 ↪insurance_test_data['Property Type_Unknown'].astype(int)

# #### Handling Outliers (excluding Premium Amount)
insurance_test_data['Number of Dependents'] = np.where(
    insurance_test_data['Number of Dependents'] > 10, 10,
    insurance_test_data['Number of Dependents']
)

Q1 = insurance_test_data['Health Score'].quantile(0.25)
Q3 = insurance_test_data['Health Score'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
insurance_test_data = insurance_test_data[(insurance_test_data['Health Score']␣
 ↪>= lower_bound) &
                                          (insurance_test_data['Health␣
 ↪Score'] <= upper_bound)]
```

```
bins = [-1, 0, 2, 5, 9, float('inf')]  # Define the bin edges
labels = ['No Claims', 'Few Claims (1-2)', 'Moderate Claims (3-5)', 'High␣
 ↪Claims (6-9)',
          'Extreme Claims (9+)']  # Define the bin labels
insurance_test_data['Previous Claims_Bin'] = pd.
 ↪cut(insurance_test_data['Previous Claims'], bins=bins,
                                                    labels=labels)
previous_claims_bin_mapping = {
    "No Claims": 1,
    "Few Claims (1-2)": 2,
    "Moderate Claims (3-5)": 3,
    "High Claims (6-9)": 4,
    "Extreme Claims (9+)": 5
}
insurance_test_data["Previous Claims_Bin Numeric"] =␣
 ↪insurance_test_data["Previous Claims_Bin"].map(
    previous_claims_bin_mapping)
insurance_test_data["Previous Claims_Bin Numeric"] =␣
 ↪insurance_test_data["Previous Claims_Bin Numeric"].cat.codes

insurance_test_data['Vehicle Age'] = np.where(
    insurance_test_data['Vehicle Age'] > 15, 15,
    insurance_test_data['Vehicle Age']
)

Q1 = insurance_test_data['Credit Score'].quantile(0.25)
Q3 = insurance_test_data['Credit Score'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
insurance_test_data = insurance_test_data[(insurance_test_data['Credit Score']␣
 ↪>= lower_bound) &
                                          (insurance_test_data['Credit␣
 ↪Score'] <= upper_bound)]

bins = [-1, 1, 3, 8, float('inf')]  # Define the bin edges
labels = ['New Clients (0-1)', 'Repeat Clients (2-3)', 'Established Clients␣
 ↪(4-8)',
          'Very Loyal Clients (9+)']  # Define the bin labels
insurance_test_data['Insurance Duration_Bin'] = pd.
 ↪cut(insurance_test_data['Insurance Duration'], bins=bins,
                                                       labels=labels)
insurance_duration_bin_mapping = {
    "New Clients (0-1)": 1,
    "Repeat Clients (2-3)": 2,
```

```
    "Established Clients (4-8)": 3,
    "Very Loyal Clients (9+)": 4
}
insurance_test_data["Insurance Duration_Bin Numeric"] =␣
 ↪insurance_test_data["Insurance Duration_Bin"].map(
    insurance_duration_bin_mapping)
insurance_test_data["Insurance Duration_Bin Numeric"] = insurance_test_data[
    "Insurance Duration_Bin Numeric"].cat.codes
```

[5]:
```
# Function to revert Box-Cox transformation
def inv_boxcox_custom(y, lambda_value):
    """
    Inverse Box-Cox transformation
    y: transformed values
    lambda_value: the lambda used in the original Box-Cox transformation
    """
    if lambda_value == 0:
        return np.exp(y)  # For lambda=0, use exp to revert log transformation
    else:
        return (y * lambda_value + 1) ** (1 / lambda_value)  # For lambda!=0
```

[6]:
```
# Apply preprocessing
X_test = insurance_test_data[['Age', 'Annual Income', 'Previous Claims',␣
 ↪'Credit Score',
                              'Age_Bin Numeric', 'Income_Bin Numeric', 'Policy␣
 ↪Start Year',
                              'Years Since Start', 'Previous Claims_Bin Numeric']]

# Generate predictions
y_pred = best_model.predict(X_test)

# After prediction, revert the transformation on 'Premium Amount'
lambda_value = 0.406  # Replace with the actual lambda value used for Box-Cox␣
 ↪transformation
y_pred_reverted = inv_boxcox_custom(y_pred, lambda_value)

# Save submission file
submission = pd.DataFrame({"id": insurance_test_data["id"], "Premium Amount":␣
 ↪y_pred_reverted})
submission.to_csv("kaggle/working/submission.csv", index=False)

print("Submission file saved!")
```

```
Submission file saved!
```