Insurance Premium Predictor

1. Problem Statement

Insurance companies often struggle to predict how much premium to charge a new or existing customer. This is because premium amount depends on a wide variety of factors like age, income, credit score, driving habits, and more. An inaccurate prediction may lead to customer dissatisfaction, loss to the company, or increased risk.

This project aims to build a machine learning model that accurately predicts the premium amount based on a given set of customer attributes.

2. Aim

The main objective of this project is to develop and deploy a machine learning model that can predict the insurance premium amount for individual customers based on their demographic, financial, and behavioral attributes.

3. Project Overview

- **Type**: Regression Problem
- **Platform**: Kaggle competition dataset
- **Tech Stack**: Python, Scikit-learn, XGBoost, MLflow, Streamlit
- **Deployment**: Streamlit Cloud
- **Final Output**: Web application that accepts user inputs and predicts the insurance premium amount.

4. Dataset Information

The dataset provided includes three files:

- train.csv: Contains ~1.2 million records used for training.
- test.csv: For final model prediction.
- sample_submission.csv: Reference format for submission.

Features Include:

- **Numerical**: Age, Annual Income, Health Score, Vehicle Age, Credit Score, Insurance Duration, etc.
- **Categorical**: Gender, Marital Status, Education Level, Occupation, Policy Type, Feedback, etc.
- **Target**: Premium Amount

5. Approach

Step-by-step Procedure:

1. **Exploratory Data Analysis (EDA)**:
    - Identified missing values
    - Inspected distributions
    - Checked feature-target correlations
2. **Feature Engineering**:
    - Missing value imputation using SimpleImputer

    - Standard scaling for numerical features
    - One-hot encoding for categorical features
3. **Model Selection**:
    - Tried Linear Regression, Decision Tree, Random Forest, XGBoost
    - Used Scikit-learn Pipelines to streamline preprocessing and modeling
4. **Model Evaluation**:
    - Used RMSLE as the primary metric
    - Logged results using MLflow
5. **Model Logging and Registration**:
    - MLflow used to track parameters, metrics, and artifacts
    - XGBoost model was registered as the best model
6. **Streamlit App Development**:
    - Built an interactive UI using Streamlit
    - Deployed app on Streamlit Cloud

6. ML Models Used

- **Linear Regression**
- **Decision Tree Regressor**
- **Random Forest Regressor**
- **XGBoost Regressor** (best performing model)

7. Model Comparison

Model	RMSLE	RMSE	MAE	R^2 Score
Linear Regression	1.17	863.34	667.28	0.00
Decision Tree	1.15	848.96	643.66	0.04
Random Forest	1.17	855.92	660.87	0.02
XGBoost	1.15	847.06	646.53	0.04

8. Evaluation Metrics

- **RMSLE (Root Mean Squared Log Error)**: Preferred due to wide range of target values
- **RMSE**: Root Mean Squared Error
- **MAE**: Mean Absolute Error
- **R^2 Score**: Coefficient of Determination

9. My ML Workflow Strategy

Manual First, Then Automated

- I began by manually executing each ML step from EDA to prediction to deeply understand the data.
- Once confident, I transitioned to automated Scikit-learn Pipelines.

Key Steps:

- Feature engineering

- Preprocessing pipelines
- Cross-validation
- Hyperparameter tuning (basic)
- Evaluation metric logging

NOTE: The model developed and deployed here is a very weak model, meaning it's practically useless and predicts wrong predictions most of the time. The reason stems from the problem of the dataset and after searching through the internet and kaggle conpetitions, I could only find a notebook which has model metrics value of RMSLE = 1.05. This was the best I could find.

The link: https://www.kaggle.com/code/lekhatopil/predict-insurance-premium-eda-lightgbm#Predicting-on-the-Test-Dataset

As we all know, RMSLE should atleast between 0.6 to 0.9 for the model to be considered okay and usable. These are all I could find at the time due to time constraints.

This model should only be used for learning purposes and can be used as an example for developing a ML model and deploying it to cloud.

10. MLflow Integration

- All models, parameters, metrics, and artifacts were tracked with MLflow.
- Models were saved and registered using the MLflow model registry.
- Versioning helped retain the best model for deployment.

11. Streamlit App Deployment

- Final model was deployed as a web app via Streamlit Cloud
- End users can input details and get instant premium predictions

Deployed App Link:

https://insurance-premium-predictor-zqspzehfdujgzv56hj9qx8.streamlit.app/

12. Challenges Faced

1. **Storage Issues**:
    - Cleared unnecessary caches and old mlruns folders
2. **MLflow Registration & Deployment Errors**:
    - Conflicts in loading model version after system sleep.
    - Fixed by re-registering and re-deploying the model pipeline.
3. **Streamlit Deployment Errors**:
    - App hang issues due to model size and cloud limitations
    - Used joblib to manually pickle the final model for use in app.py
4. **Cross-Version Compatibility**:
    - Python version mismatches caused errors like _RemainderColsList
    - Resolved by ensuring the same scikit-learn version locally and in cloud
5. **Limited GitHub Upload Space**:
    - Pushed only main.ipynb, app.py, and requirements.txt

- Added a detailed README.md to explain the complete workflow

13. Files Included

- main.ipynb: Complete pipeline-based training and evaluation code
- app.py: Streamlit UI script
- requirements.txt: All Python packages used

14. How to Run the Project Locally

pip install -r requirements.txt

streamlit run app.py

15. Future Improvements

- Add more robust cross-validation (e.g., KFold, StratifiedKFold)
- Try other ensemble models like CatBoost or LightGBM
- Containerize with Docker for better reproducibility
- Deploy with cloud services like Heroku, AWS EC2, or Azure

16. Conclusion

This project demonstrates the complete life cycle of an ML project from EDA to deployment. It involved solving real-world issues like missing data, preprocessing, model tuning, and deployment challenges. The final Streamlit web app enables end users to predict insurance premiums interactively.

17. References

- Kaggle Competition Dataset
- Scikit-learn Documentation
- XGBoost Official Docs
- Streamlit Docs
- MLflow Tracking API

Prepared by:

Ishaaq M M

LinkedIn: https://www.linkedin.com/in/ishaaq-m-m

Github: https://github.com/Ishaaq09

Year: 2025