# Smart Premium Prediction – End-to-End Project Documentation

## 1. Objective

The goal of the Smart Premium Prediction project is to build a machine learning model that accurately predicts a user's insurance premium amount based on their health and personal attributes. The final model is integrated into a Streamlit web application to enable user interaction and real-time predictions.

## 2. Data Preprocessing and Cleaning

The raw dataset includes health, demographic, and policy-related fields. Data cleaning involves:  
- Handling missing values using median imputation for numeric columns and mode imputation for categorical columns.  
- Removing outliers using IQR-based filtering or domain-specific rules.  
- Converting date columns to datetime and extracting useful features like policy duration.  
- Encoding categorical variables (like Gender, Smoker, Diabetes) using One-Hot Encoding or Label Encoding.

## 3. ML Pipeline Class (SmartPremiumPipeline)

A custom ML pipeline class is created to handle preprocessing, training, evaluation, and prediction.  
- The `fit()` method handles preprocessing, model training, evaluation metrics calculation, and MLflow logging.  
- The `predict()` method ensures the same preprocessing steps are applied to new data before making predictions.  
- The class supports switching between different models: Random Forest, XGBoost, Decision Tree, and Linear Regression.  
- Model performance is evaluated using RMSE, MAE, and R² metrics.

## 4. Model Comparison and Selection

Four different models were evaluated on the dataset:  
- Random Forest: R² = 0.04  
- XGBoost: R² = 0.03  
- Decision Tree: R² = 0.01  
- Linear Regression: R² = 0.003  
Although all models performed poorly, Random Forest had the highest score and was chosen for further tuning.

## 5. Hyperparameter Tuning

Hyperparameter tuning was performed using both GridSearchCV and RandomizedSearchCV separately on RandomForestRegressor.  
- RandomizedSearchCV was preferred for large hyperparameter spaces due to better efficiency.  
- Best parameters were identified and used for model retraining.

## 6. Model Saving and Logging

Top-performing models (RandomForest and XGBoost) were saved separately using `joblib` in a 'saved\_models' directory.  
- MLflow was used to log parameters, metrics, and models under custom run names.  
- Each model's run was given a clear and identifiable name using `run\_name=f"{model\_name}\_Run"`.

## 7. Streamlit Application

A Streamlit UI was created for users to input their health information and get predicted premium amounts.  
- Users fill out a form with inputs like Age, Gender, Credit Score, BMI, Smoker, Diabetes, and Policy Start Date.  
- Input data is converted into a DataFrame and passed to the pipeline's `predict()` method.  
- Predictions are shown rounded to two decimal places using `round(prediction[0], 2)`.  
- A 'Cancel' button was added to the form UI to allow users to cancel and see a cancellation message.

## 8. Importing and Usage Instructions

- The pipeline class was imported using: `from smart\_premium\_pipeline import SmartPremiumPipeline`  
- Saved models were loaded using `joblib.load('saved\_models/RandomForest\_model.pkl')`  
- The notebook file (`smart\_premium\_Analysis.ipynb`) was converted into a Python module for use in Streamlit.

## 9. Challenges and Solutions

- Encountered MLflow server port errors (address already in use) and resolved by changing the port or killing the process.  
- Handled inconsistencies in model prediction output and ensured rounding worked properly.  
- Managed Streamlit form layout issues and added right-aligned cancel button with conditional logic.

## 10. Future Enhancements

- Improve feature engineering using domain knowledge.  
- Apply advanced ML algorithms like CatBoost or LightGBM.  
- Enable user authentication in the Streamlit app for secure access.