DEPARTMENT OF DECISION AND COMPUTING SCIENCES

17MDC95 – MINOR PROJECT – DECISION TOOL DEVELOPMENT

DECISION SUPPORT SYSTEM FOR INSURANCE: OPTIMIZING OPERATIONS AND ENHANCING CUSTOMER ENGAGEMENT

PROJECT REPORT

SUBMITTED BY

2033016 – KAAVIASUDHAN V S

2033037 – THILAK P L

SUBMITTED TO

DR. RADHAMANI V

Assistant Professor

Department of Computing

COIMBATORE INSTITUTE OF TECHNOLOGY

(Government Aided Autonomous Institution)

Coimbatore – 641014

TABLE OF CONTENTS

CHAPTER I
INTRODUCTION

The insurance industry is evolving rapidly due to technological advancements and rising customer expectations. To remain competitive, companies need to improve operational efficiency and address challenges like risk management, fraud detection, and customer satisfaction. This project aims to develop an **Information Decision Support System (IDSS)** to help insurers make better decisions, streamline processes, and enhance customer experiences through advanced data analytics and real-time insights.

The project includes several modules, starting with the **Premium Prediction Module**, which uses machine learning to forecast premiums based on customer data and market trends. This helps insurers align pricing strategies with demand, ensuring they remain competitive while maximizing profitability. The **Risk Management Module** assesses policyholder risk by evaluating various factors such as claims history and customer behavior, enabling better control over high-risk clients to minimize potential losses and protect the insurer's financial stability.

The **Fraud Detection Module** identifies suspicious claims through predictive algorithms that analyze patterns in claims data, helping reduce financial loss from fraudulent activities and safeguarding the integrity of the insurance process. The **Policy Recommendation Module** assists in selecting tailored policies based on customer needs and preferences, improving satisfaction and renewal rates while enhancing the overall customer experience through personalized service.

The **Customer Engagement Module** integrates AI-powered virtual assistants, such as Insurance GPT, to provide real-time answers to customer queries, improving interaction and support while allowing for a seamless user experience. The **Sales and Market Impact Module** analyzes the impact of different policies on market share and sales trends, helping insurers refine their strategies and make data-driven decisions that capitalize on emerging market opportunities.

CHAPTER II

PROBLEM STATEMENTS

It is commonly acknowledged that the insurance industry is a very competitive one with a number of existing negative boundaries such as client service tailoring, the strategy of setting up the premiums, and lowering the operational risk. Some of the major challenges are:

1.1 Inconsistent Premium Predictions:

Many insurance companies have inadequacies in estimating premium rates which make them overprice their products or set prices that are too low. This affects the overall net income and customers as well.

1.2 Inefficient Risk Management:

It is necessary to address problems such as fraud and policy management, albeit most of these companies do not have the necessary resources for fast fraud investigation and risk control.

1.3 Customer Engagement Challenges:

Providing adequate and satisfying coverage to clients by suggesting generalized policies recommended rather than those that are individually catered to the clients is one of the main issues that the insurers encounter.

In order to address these issues, this project suggests an integrated DSS which encompasses a number of tools including prognostics modeling and data analysis in both decision making and operation efficiency in these areas.

CHAPTER III
OBJECTIVES

The aim of the unmet needs analysis in Decision Support System (DSS) for Insurance Company is to support the decision making and improve the company's performance in the areas discussed below:

3.1 Sales Forecasting: Enables projection of future sales performance to sharpen the optimal sales strategies aimed at managing revenue collections.

3.2 Fraud Detection: Strategies employed in identification and prevention of activities imperilling the insurance business.

3.3 Risk Assessment: Identification and control of risks arising from various categories of offered insurance products in order to reduce losses.

3.4 Policy Recommendations: Recommend appropriate insurance contracts to individuals in relation to their profile in order to enhance customer experience.

3.5 Policy Renewals Analysis: Studied and dissected renewal patterns of clients for enhanced client retention strategies and effectiveness of the renewed processes.

3.6 Policy Premium Prediction: Enable prediction on policy premiums that would guide users on their choices of insurance coverage.

3.7 Insurance Query Handling: This tool uses advanced AI to help answer your questions about various insurance policies, claims, and general topics. Think of it as a smart assistant that provides accurate, up-to-date information about insurance products and services. This not only boosts user engagement but also helps you make better decisions.

3.8 Channel Prediction: This feature identifies the best marketing channels to optimize campaign performance. It considers factors like the target audience, type of policy, budget, and data from previous campaigns. By leveraging machine learning, it helps select channels that reach more people and drive more sign-ups, making your marketing strategies more effective.

The overall ambition of DSS is to combine these functionalities of the system to the user and also for supporting managerial decisions hence achieving better performance of the organization and serving the customer more efficiently.

CHAPTER IV
SCOPE

The scope of the Decision Support System (DSS) for Insurance Company encompasses the following areas:

4.1 Sales Forecasting: The system will focus on predicting future sales trends for various insurance products, leveraging historical sales data and market trends to provide actionable insights for optimizing sales strategies.

4.2 Fraud Detection: The DSS will incorporate advanced analytics and machine learning techniques to detect and prevent fraudulent activities within the insurance processes, safeguarding the company's operations and financial integrity.

4.3 Risk Assessment: The system will assess potential risks associated with different insurance products by analyzing historical data, market conditions, and policyholder information, enabling proactive risk management.

4.4 Policy Recommendations: The DSS will offer personalized policy recommendations based on user-specific inputs and preferences, aiming to enhance customer satisfaction by providing tailored insurance solutions.

4.5 Policy Renewals Analysis: The system will analyze renewal data to identify patterns and trends, assisting in the development of strategies to improve customer retention and streamline the renewal process.

4.6 Policy Premium Prediction: The DSS will predict annual premiums for various insurance policies using user-specific data and predictive modelling, helping customers make informed decisions about their insurance coverage.

The project is specifically tailored to Insurance Company's operations, focusing on its diverse range of insurance products, including life, health, and motor insurance, within the Indian market. The system will be designed to integrate with existing infrastructure and data sources, providing a comprehensive solution to meet the company's needs for enhanced decision-making and operational efficiency.

CHAPTER V

PROPOSED WORK

It will execute the project in the following phases:

5.1 Requirement Analysis:

Work closely with stakeholders to identify system requirements and ensure that the DSS functionalities are aligned with the strategic goals of the insurance industry.

5.2 Data Collection and Integration:

Gather and preprocess data from multiple insurance-related sources, such as customer profiles, claims, policy records, and market trends, to ensure accurate analysis and model training.

5.3 System Development:

Develop key DSS components for predicting premiums, detecting fraud, managing risks, and recommending policies. Leverage Python, Flask, and machine learning libraries to create an intuitive interface and ensure smooth system performance.

5.4 Testing and Integration:

Conduct rigorous testing to validate model accuracy and integrate the DSS seamlessly with existing insurance platforms for efficient operation.

5.5 Implementation and Training:

Roll out the DSS and provide comprehensive training to users, enabling them to effectively utilize the system in decision-making processes.

5.6 Monitoring and Continuous Improvement:

Regularly monitor the system's performance and gather user feedback to continuously improve and adapt the DSS, ensuring its ongoing relevance and effectiveness.

CHAPTER VI

DATASET WITH EXPLORATORY DATA ANALYSIS (EDA)

6.1 Sale forecasting Insurance:

The dataset contains the following columns:

- Date: The specific day a sale was recorded.
- Policy_Type: The category of insurance policy sold.
- Year: The year the sales were logged.
- Month: The month during which the sales took place.
- Premium_Rate: The price charged for the insurance policy.
- Total_Investment_Cover: The total coverage amount provided by the policy.
- Customer_Acquisition: The number of new customers gained.
- Customer_Retention: The count of returning customers.
- Operational_Impact: The effect of operations on sales performance.
- Market_Share_Impact: The influence of the policy on market share.
- Renewal_Rates: The percentage of policies that were renewed.
- Sales: The total quantity of policies sold.

```
<bound method DataFrame.info of           Date            Policy_Type  Year  Month  Premium_Rate  \
0    2010-01-31        Health Insurance  2010      1   3807.947177
1    2010-01-31          Life Insurance  2010      1   9512.071633
2    2010-01-31  Motor Vehicle Insurance  2010      1   7346.740024
3    2010-01-31          Home Insurance  2010      1   6026.718994
4    2010-02-28        Health Insurance  2010      2   1644.584540
..          ...                     ...   ...    ...           ...
711  2024-10-31          Home Insurance  2024     10   5996.823532
712  2024-11-30        Health Insurance  2024     11   4768.604267
713  2024-11-30          Life Insurance  2024     11   4177.225050
714  2024-11-30  Motor Vehicle Insurance  2024     11   3553.795839
715  2024-11-30          Home Insurance  2024     11   9302.338528

     Total_Investment_Cover  Customer_Acquisition  Customer_Retention  \
0             832313.213710                   178            0.505449
1             965376.641560                   919            0.930426
2             133054.251251                   428            0.831879
3             733558.800452                   294            0.658963
4             938957.052253                   852            0.633441
..                      ...                   ...                 ...
711           952293.667569                   385            0.528106
712           642728.738437                   621            0.897559
713           869239.111575                   435            0.701744
714           460192.457078                   408            0.729695
715           520440.068293                   483            0.781967

     Operational_Impact  Market_Share_Impact  Renewal_Rates          Sales
0              9.337443             3.077988       0.671102  415210.946990
1              1.172045             1.302226       0.473359  415633.544263
2              5.291591             0.382888       0.505216  356147.466803
3              7.189498             2.016061       0.410668   57173.681811
4              7.504363             1.247597       0.775726  420232.661643
..                  ...                  ...            ...            ...
711            3.173832             0.512226       0.485138  244665.650921
712            4.188442             2.026209       0.724173  398037.152882
713            1.948109             0.591203       0.630001  290362.007054
714            3.001252             0.181303       0.671941  174324.370155
715            5.673021             3.342422       0.607404  405939.478123

[716 rows x 12 columns]>
```

Figure 6.1.1 - Sale forecasting dataset descriptions

6.2 Fraud Detection

The dataset includes the following columns:

- Policy_ID: Unique identifier for each policy.
- Policy_Type: The type of insurance policy involved.
- Customer_ID: Unique identifier for each customer.
- Annual_Premium: The yearly premium paid by the customer.
- Claims_Made: The number of claims filed by the customer.
- Total_Claim_Amount: Total amount claimed by the customer.
- Last_Claim_Amount: The amount claimed in the last instance.
- Claim_Status: Status of the claim (e.g., Approved, Rejected).
- Risk_Score: A score assessing the likelihood of fraud.

```
<bound method DataFrame.info of       Policy_ID       Policy_Type Customer_ID  Annual_Premium  Claims_Made  \
0       POL1000             Motor   CUST10000       31938.320            4
1       POL1001  Personal Accident   CUST10001        5968.090            2
2       POL1002            Health   CUST10002       48676.010            1
3       POL1003            Travel   CUST10003       23263.000            0
4       POL1004              Life   CUST10004       27209.545            3
...         ...               ...         ...             ...          ...
11995  POL11995              Life   CUST13995        7740.730            0
11996  POL11996              Home   CUST13996       21420.110            3
11997  POL11997      Agricultural   CUST13997        5582.960            1
11998  POL11998  Personal Accident   CUST13998       36470.290            3
11999  POL11999              Life   CUST13999       14642.240            2

       Total_Claim_Amount  Last_Claim_Amount        Claim_Status   Risk_Score
0               187303.38       12915.786979              Normal    33.365414
1               115362.12       19196.106864              Normal    13.341944
2               139872.16       32094.272025              Normal    19.942710
3                76549.51       32388.576636  Claim_Without_Count     0.000000
4               117513.39       25473.705196              Normal    24.908471
...                   ...                ...                 ...          ...
11995           108420.37       44095.923437  Claim_Without_Count     0.000000
11996           133423.80       19378.438651              Normal    30.475399
11997           109173.72           0.000000              Normal   124.916639
11998            60588.22       17598.898860              Normal    23.933658
11999           193403.90       12837.705776              Normal  1384.165167

[12000 rows x 9 columns]>
```

Figure 6.2.1 - Fraud Detection dataset descriptions

6.3 Medical Cost prediction:

The dataset includes the following columns:

- age: The age of the customer.
- sex: The gender of the customer.
- bmi: The customer's body mass index.
- children: The number of children the customer has.
- smoker: Whether the customer is a smoker (Yes/No).
- region: The region where the customer lives.
- charges: The total medical charges incurred.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Figure 6.3.1 – Medical cost Prediction dataset descriptions

6.4 Policy Recommendation:

The dataset includes the following columns:

- Policy Name: The name of the insurance policy.
- Customer Age: The age of the customer.
- Occupation: The customer's job or occupation.
- Income: The customer’s income level.
- Education: The educational background of the customer.
- Marital Status: Whether the customer is married or not.
- Tenure (Years): Number of years the customer has held a policy.
- Premium (INR): The amount of premium paid in Indian Rupees.
- Coverage (INR): The amount of coverage provided in Indian Rupees.
- Family Size: The number of family members.
- Gender: The customer’s gender.
- Region: The geographical area the customer is from.
- Policy Type: The type of insurance policy recommended.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Policy Name     5000 non-null   object
 1   Customer Age    5000 non-null   int64 
 2   Occupation      5000 non-null   object
 3   Income          5000 non-null   int64 
 4   Education       5000 non-null   object
 5   Marital Status  5000 non-null   object
 6   Tenure (Years)  5000 non-null   int64 
 7   Premium (INR)   5000 non-null   int64 
 8   Coverage (INR)  5000 non-null   int64 
 9   Family Size     5000 non-null   int64 
 10  Gender          5000 non-null   object
 11  Region          5000 non-null   object
 12  Policy Type     5000 non-null   object
dtypes: int64(6), object(7)
memory usage: 507.9+ KB
```

Figure 6.4.1 – Policy Recommendation dataset descriptions

6.5 Risk Management

The dataset includes the following columns:

- Policy_ID: Unique identifier for each policy.
- Policy_Type: Type of insurance policy held by the customer.
- Customer_ID: Unique identifier for each customer.
- Customer_Age: The age of the customer.
- Annual_Premium: The yearly premium paid for the policy.
- Policy_Status: The current status of the policy (e.g., Active, Lapsed).
- Risk_Score: A score indicating the risk level of the policyholder.
- Is_High_Value_Customer: Whether the customer is considered high-value (1 = Yes, 0 = No).
- Life_Stage: The customer's stage in life (e.g., Young Adult, Senior).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12000 entries, 0 to 11999
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Policy_ID               12000 non-null  object 
 1   Policy_Type             12000 non-null  object 
 2   Customer_ID             12000 non-null  object 
 3   Customer_Age            12000 non-null  int64  
 4   Annual_Premium          12000 non-null  float64
 5   Policy_Status           12000 non-null  object 
 6   Risk_Score              12000 non-null  float64
 7   Is_High_Value_Customer  12000 non-null  int64  
 8   Life_Stage              12000 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 843.9+ KB
```

Figure 6.5.1 – Risk Management dataset descriptions

6.6 EDA OUTCOME:

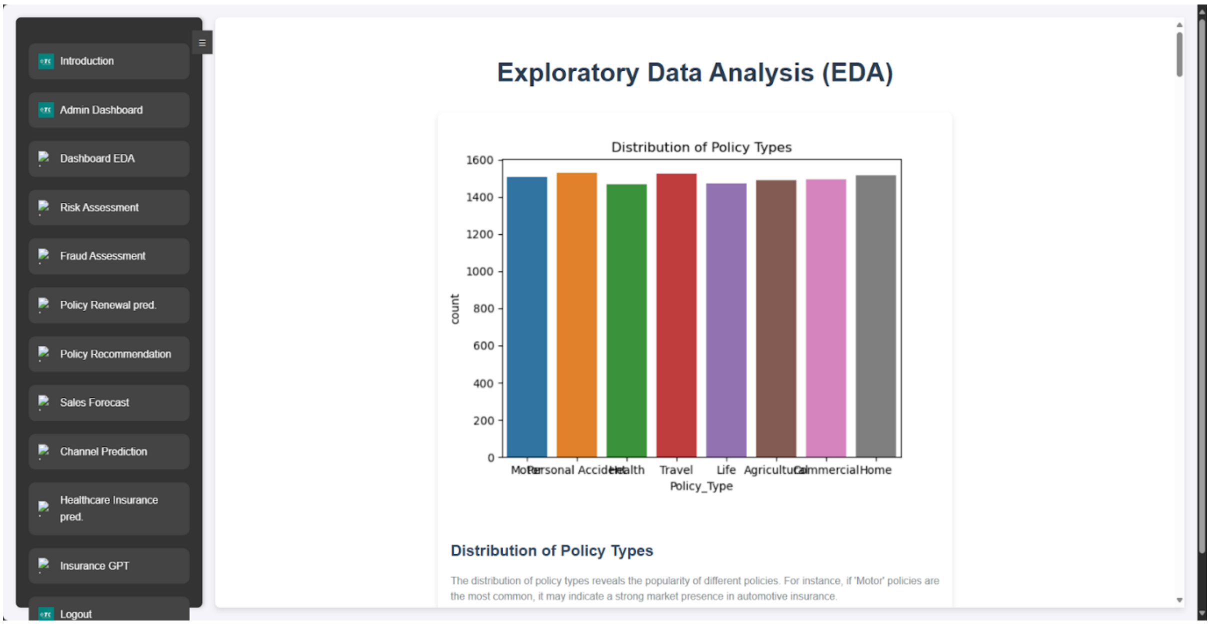

Figure 6.6.1 – Distribution of Policy Types

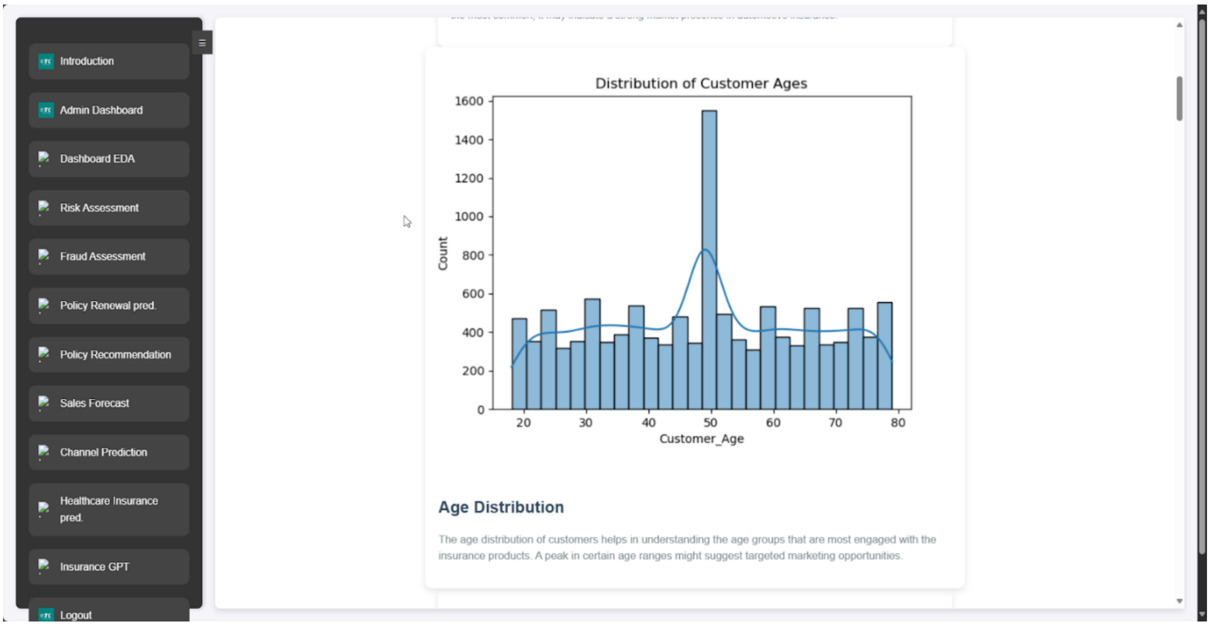

Figure 6.6.2 – Distribution of Age

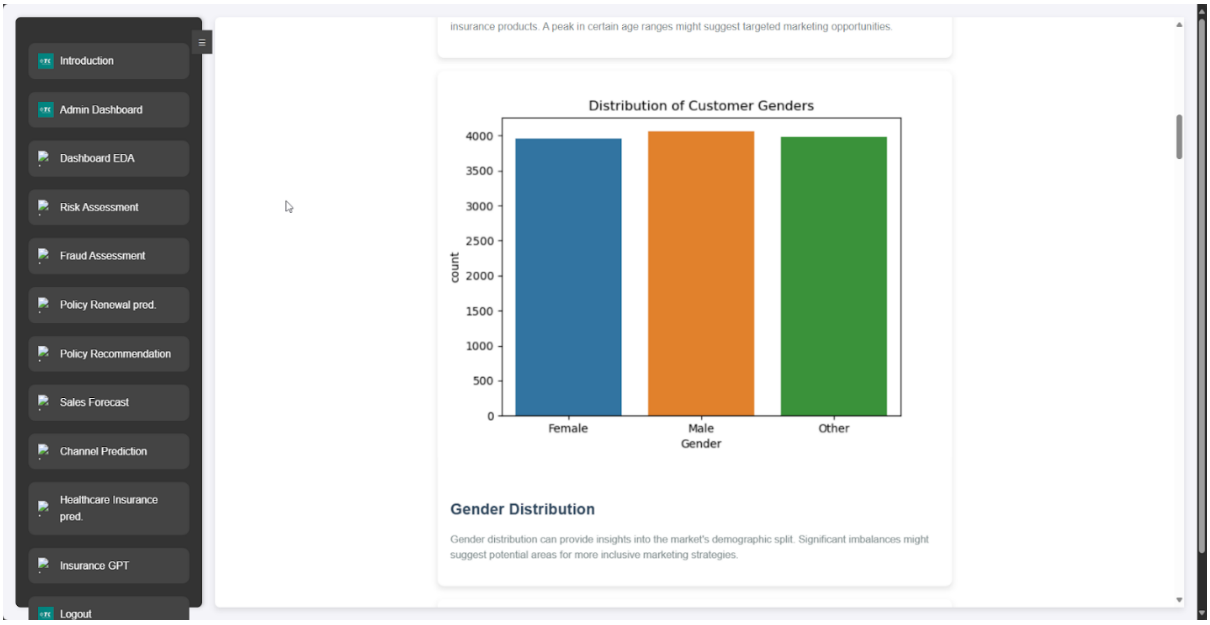

Figure 6.6.3 – Distribution of Gender

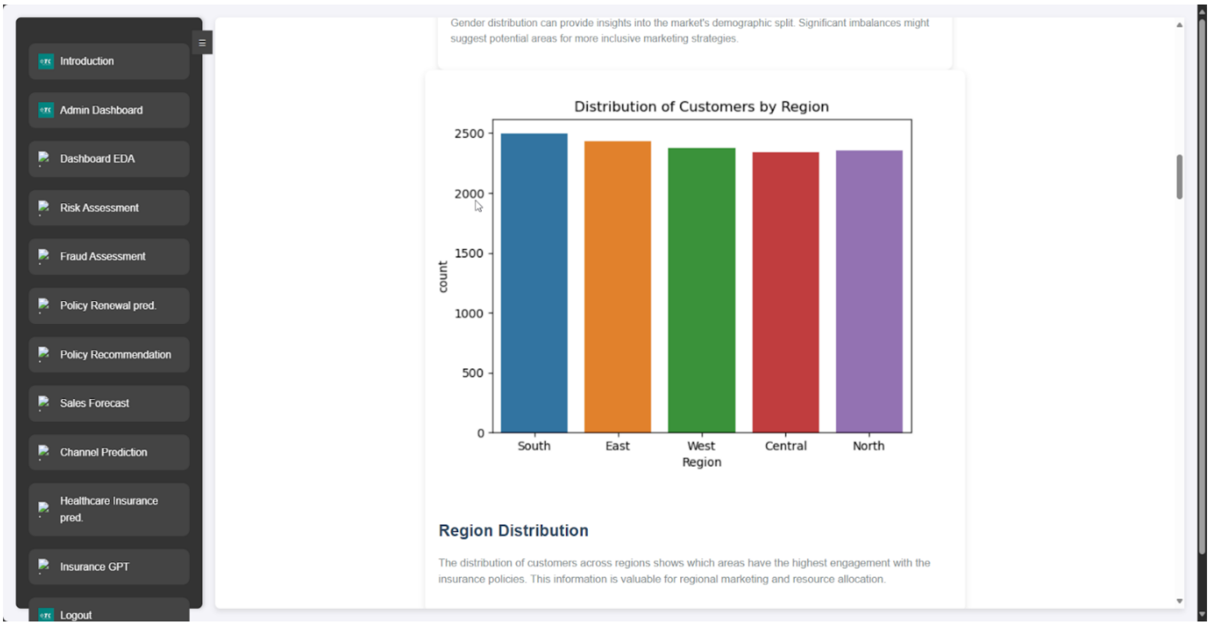

Figure 6.6.4 – Distribution of Region

Figure 6.6.5 – Trend Analysis of Policy Types based on Year

Figure 6.6.6 – Distribution of Policy Type vs Marital Status

CHAPTER 7

TOOLS AND TECHNIQUES

7.1 flask_pymongo:

- Integrates Flask with MongoDB, allowing you to easily perform database operations directly within Flask routes.

7.2 dotenv:

- Automatically loads environment variables from a `.env` file, making configuration settings secure and easy to manage without hardcoding them into the application.

7.3 flask_mail:

- Provides a simple interface to send emails asynchronously in Flask apps, commonly used for user notifications, password resets, etc.

7.4 matplotlib:

- A versatile library for creating a wide range of static and interactive plots and visualizations, including bar charts, line plots, scatter plots, and more.

7.5 seaborn:

- Builds on top of Matplotlib to provide aesthetically pleasing and easy-to-create statistical graphics, simplifying complex visualizations like heatmaps or regression plots.

7.6 sklearn.ensemble (RandomForestClassifier, RandomForestRegressor):

- Implements ensemble machine learning models that combine multiple decision trees to improve accuracy in classification (RandomForestClassifier) and regression (RandomForestRegressor) tasks.

7.7 flask (Blueprint, render_template, redirect, url_for, session):

- Offers a modular approach to organizing routes and views in Flask apps while handling template rendering, URL management, and session storage.

7.8 bson.objectid (ObjectId):

- Manages MongoDB's unique document identifiers, allowing you to interact with and reference specific database entries.

7.9 pymongo (Mongo Client):

- Allows connection and interaction with MongoDB databases from Python, enabling CRUD operations, querying, and more advanced database handling.

7.10 secrets:

- Generates cryptographically strong random values, often used for generating secure tokens, session keys, or passwords.

These tools and techniques are integral to developing a comprehensive Decision Support System for Insurance Company, facilitating efficient data processing, model building, and user interface design.

CHAPTER 8

METHODOLOGIES

8.1 Data Visualization:

- Utilizes libraries like Matplotlib, Seaborn, and Plotly to visually represent data.
- Techniques include bar plots, line charts, and annotations to highlight trends and key insights.

8.2 Data Filtering and Aggregation:

- Filters data based on user inputs such as date ranges, product categories, and brands.
- Aggregates sales data by categories like date, brand, and product category to derive meaningful insights.

8.3 Data Preprocessing:

- Involves removing irrelevant columns, handling missing values, encoding categorical variables, and normalizing numerical features for better model performance.

8.4 Feature Engineering:

- Creates new features such as Duration and Budget Efficiency to provide additional insights and enhance model accuracy.

8.5 Actionable Dynamic Insights:

- Generates insights dynamically based on quantiles and averages, identifying underperforming or overperforming campaigns, channels, or target audiences for improved decision-making.

8.6 Trends Over Time Analysis:

- Tracks and visualizes metrics (e.g., Revenue Generated, Budget) over time, identifying patterns and trends that inform strategy.

8.7 Label Encoding:

- Converts categorical variables into numerical form, enabling machine learning models to process the data effectively.

8.8 Standard Scaling:

- Normalizes features so they have a mean of 0 and a standard deviation of 1, which is important for algorithms like Random Forest.

8.9 Univariate and Bivariate Analysis:

- Univariate Analysis examines the distribution of each variable.

- Bivariate Analysis explores relationships between two variables, often using visualizations like box plots.

8.10 Correlation Analysis:

- Determines relationships between variables, with a heatmap used to visualize the correlation matrix of numerical features.

8.11 Premium Prediction:

- Uses regression models like Linear Regression and XGBoost to predict policy premiums based on customer data.

8.12 Fraud Detection:

- Implements classification techniques such as Random Forest to flag suspicious activities in claims data.

8.13 Customer Segmentation:

- Applies KMeans Clustering to group customers by attributes like income, age, and risk level, optimizing targeted marketing efforts.

8.14 Risk Assessment:

- Builds predictive models to evaluate risks associated with policyholders and adjust premium pricing accordingly.

CHAPTER 9

ALGORITHMS

9.1 Random Forest:

- Used for: Healthcare Insurance, Policy Recommendation, Channel Prediction.
- Description: This is an ensemble model that builds multiple decision trees and combines their output to improve accuracy and prevent overfitting. It's particularly good for classification and regression tasks.

9.2 Logistic Regression:

- Used for: Fraud Detection, Policy Renewal Prediction.
- Description: This model is used for binary classification. It predicts the probability of a binary outcome (e.g., fraud or not, renewal or not) using a linear equation and a logistic function.

9.3 Support Vector Classifier (SVC):

- Used for: Policy Renewal Prediction.
- Description: SVC is a powerful classification algorithm that works by finding the hyperplane that best separates different classes. It's useful when classes are not linearly separable.

9.4 ARIMA (Autoregressive Integrated Moving Average):

- Used for: Sales Forecasting.
- Description: ARIMA is a time series model that captures patterns in data over time. It's useful for forecasting future values based on historical data trends.

9.5 Linear Regression:

- Used for: Risk Assessment.
- Description: This model predicts a continuous target variable (e.g., risk score) based on one or more input features by fitting a straight line through the data.

These models work together to provide predictions for different aspects of the insurance management system.

CHAPTER

INTRODUCTION

TITLE: Decision Support System for Insurance: Optimizing Operations and Enhancing Customer Engagement

ABSTRACT:

This paper presents the development of an integrated Decision Support System (DSS) for Insurance Company, designed to optimize operations and enhance customer engagement. The DSS leverages modern technologies and advanced data analytics to address critical operational areas in the insurance sector. It includes modules for sales forecasting, fraud detection, risk assessment, policy recommendations, policy renewals analysis, premium predictions, channel prediction, healthcare insurance predictions, and an AI-powered assistant for user queries. The system utilizes predictive models such as Random Forest, Logistic Regression, ARIMA, and Support Vector Classifier to provide data-driven insights and actionable decisions. This DSS aims to improve operational efficiency, customer satisfaction, and financial stability by providing a comprehensive solution for Insurance Company.

Keyword: Operational Efficiency, Customer Engagement, Insurance Sector, Business Process Optimization, Decision Support Systems, Predictive Analytics, Machine Learning, Data-Driven Decision Making, Insurance Business Intelligence

10 ALGORITHMS:

11 MODULE:

11.1 Welcome Hub (Introduction)

- Purpose: Provides an overview of the system and introduces users to the insurance management platform.
- Offers quick links and a summary of key features to guide users through the system.

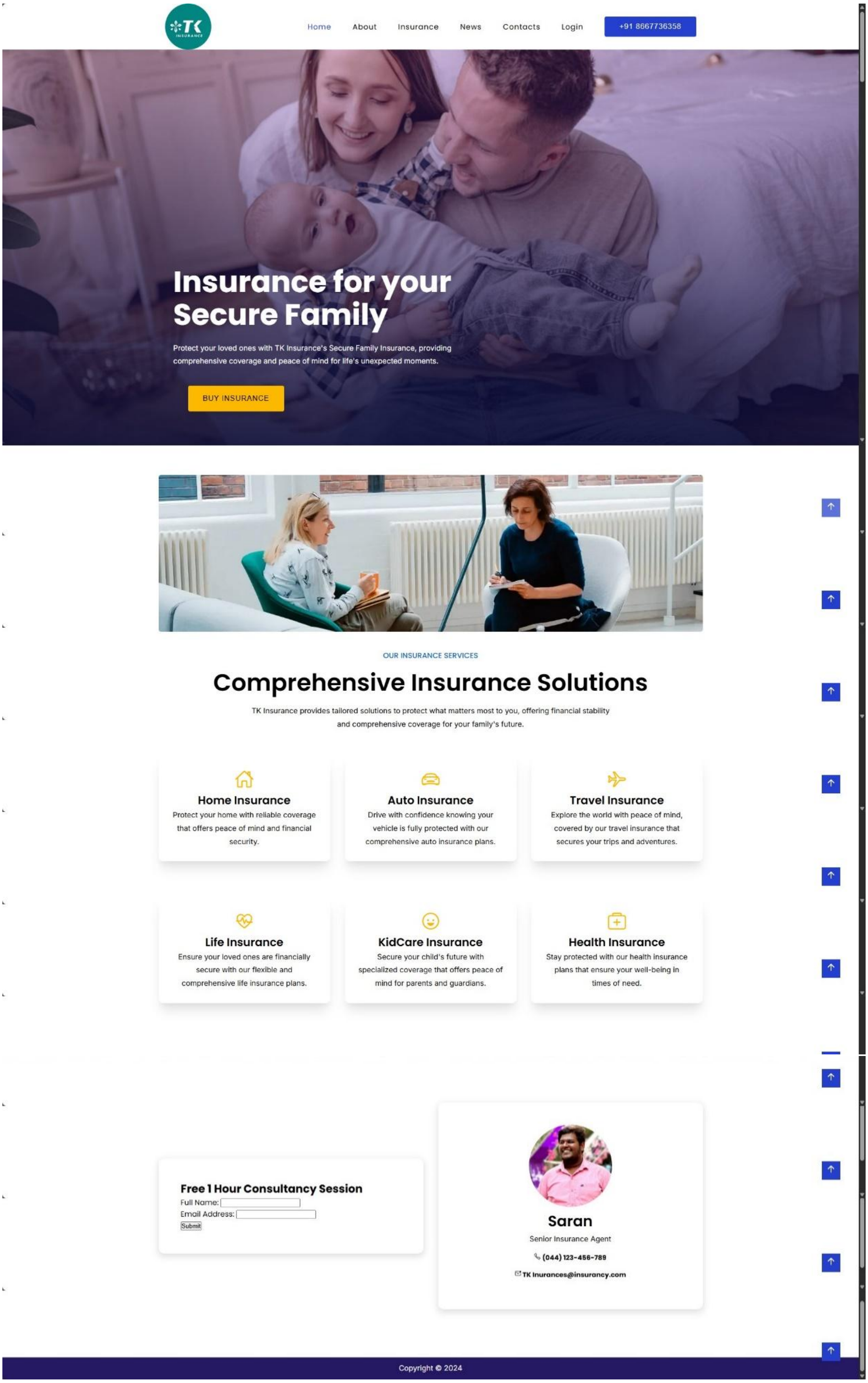
TK
Home
About
Insurance
News
Contacts
Login
+91 8667736358
Insurance for your Secure Family
Protect your loved ones with TK Insurance's Secure Family Insurance, providing comprehensive coverage and peace of mind for life's unexpected moments.
BUY INSURANCE
OUR INSURANCE SERVICES
Comprehensive Insurance Solutions
TK Insurance provides tailored solutions to protect what matters most to you, offering financial stability and comprehensive coverage for your family's future.
Home Insurance
Protect your home with reliable coverage that offers peace of mind and financial security.
Auto Insurance
Drive with confidence knowing your vehicle is fully protected with our comprehensive auto insurance plans.
Travel Insurance
Explore the world with peace of mind, covered by our travel insurance that secures your trips and adventures.
Life Insurance
Ensure your loved ones are financially secure with our flexible and comprehensive life insurance plans.
KidCare Insurance
Secure your child's future with specialized coverage that offers peace of mind for parents and guardians.
Health Insurance
Stay protected with our health insurance plans that ensure your well-being in times of need.
Free 1 Hour Consultancy Session
Full Name:
Email Address:
Submit
Saran
Senior Insurance Agent
(044) 123-456-789
TK Inurances@insurancy.com
Copyright © 2024

Figure 10.1 – Welcome Hub

11.2 Control Center (Admin Dashboard)

- Purpose: Admins can monitor and manage users, system operations.
- Enables access to administrative tasks, settings, and data control.

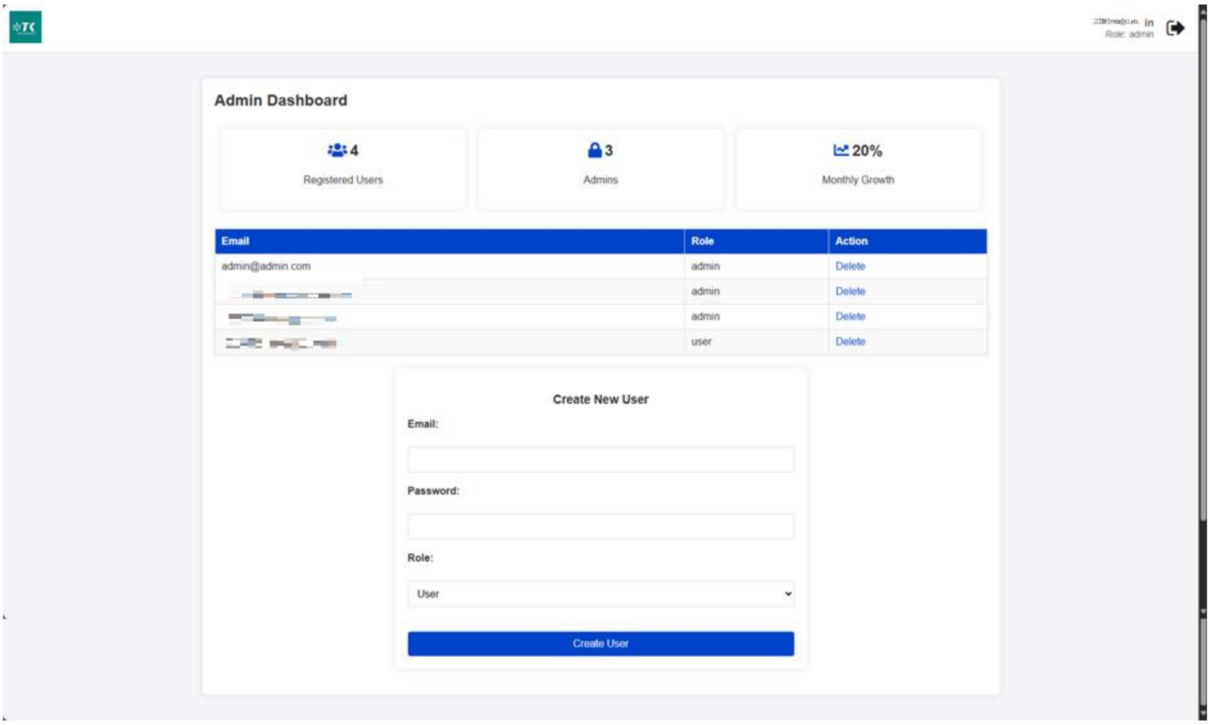

Figure 10.2 - Control Center

11.3 Data Explorer (Explore Dashboard)

Purpose: Visualizes key insurance metrics, customer insights, and policy trends. Allows users to analyze historical data and generate custom reports.

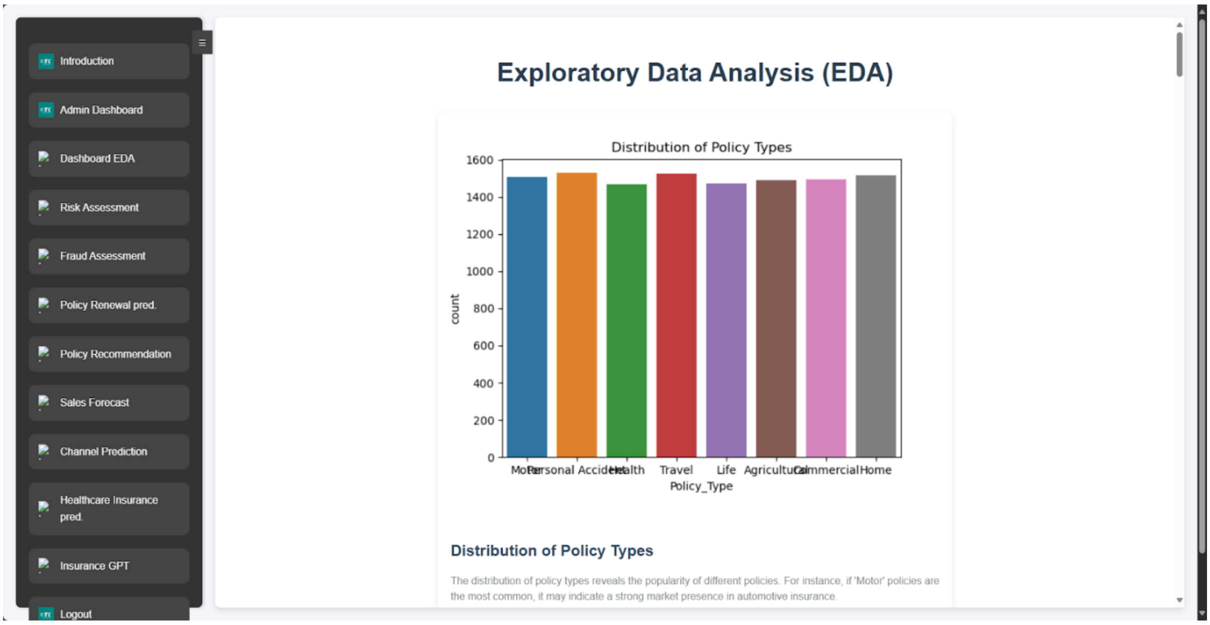

Figure 10.3.1 – EDA (1)

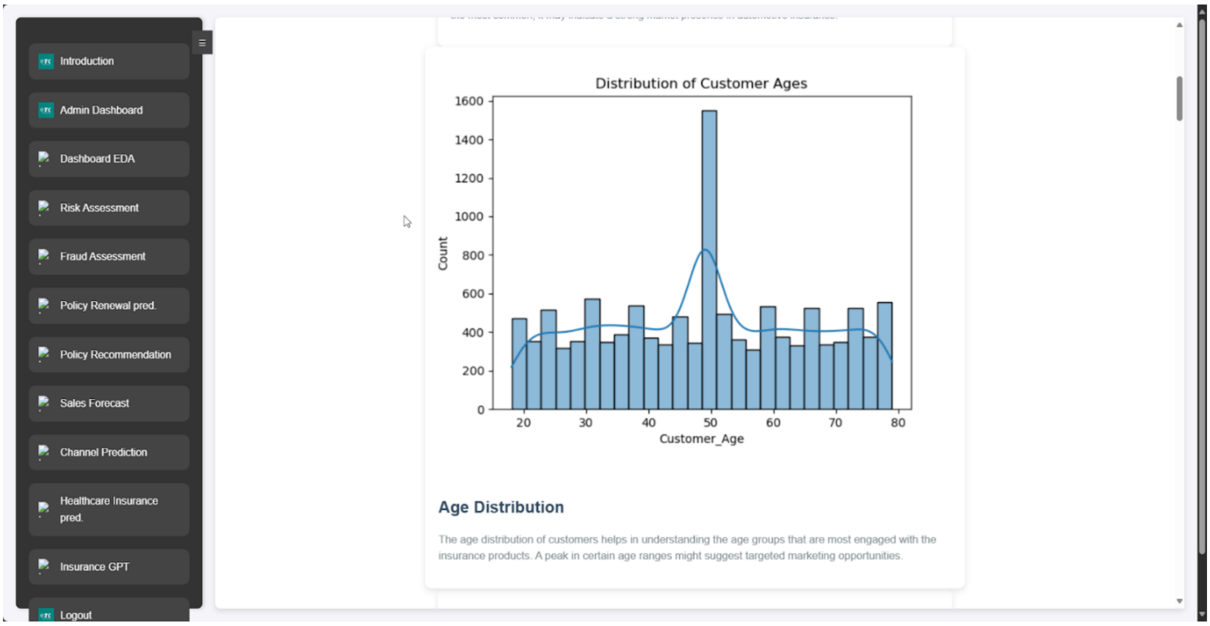

Figure 10.3.2 – EDA (2)

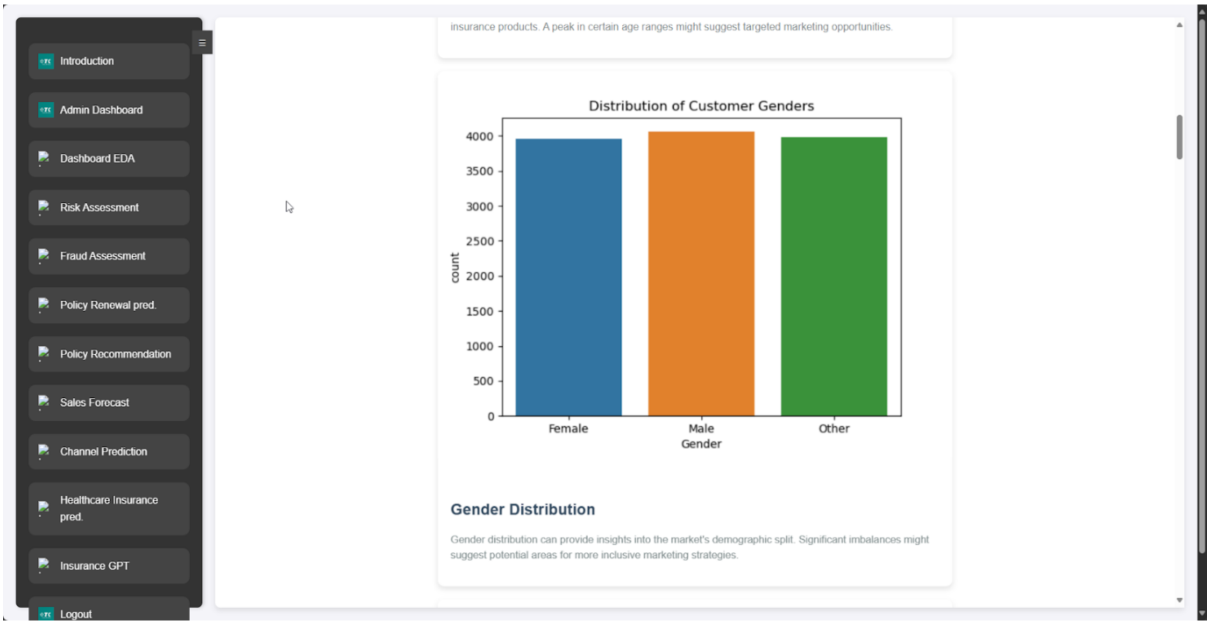

Figure 10.3.3 – EDA (3)

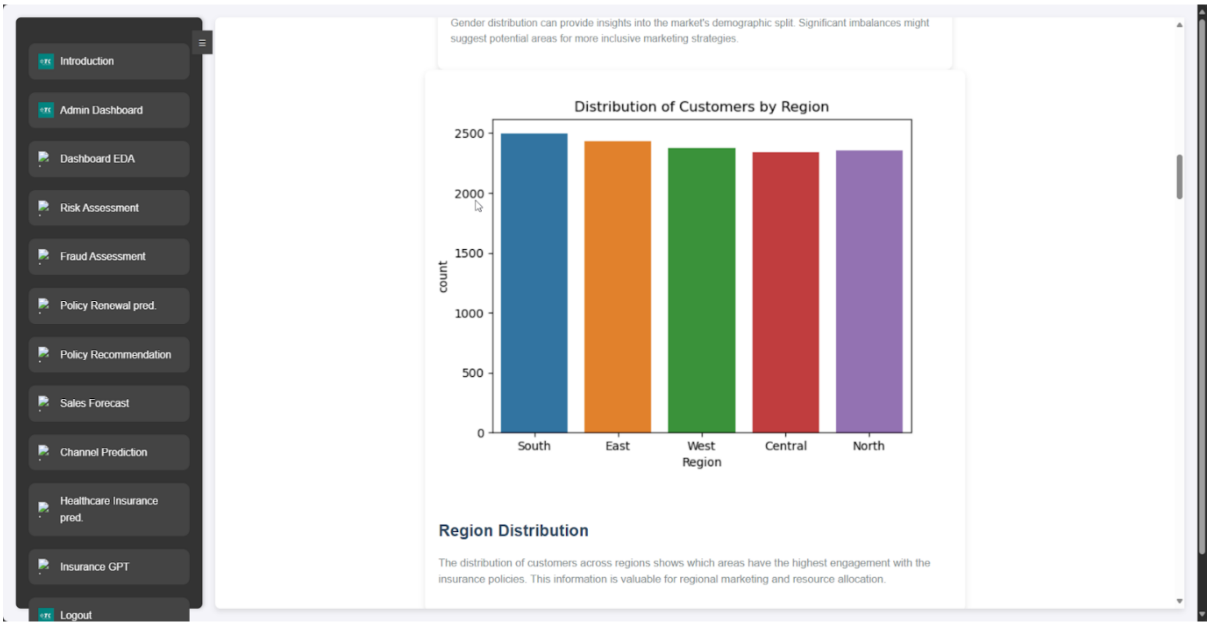

Figure 10.3.4 – EDA (4)

11.4 Risk Radar (Risk Assessment)

- Purpose: Predicts the level of risk associated with individual insurance policies.
- Helps insurers assess and mitigate potential risks in policies using data-driven models.

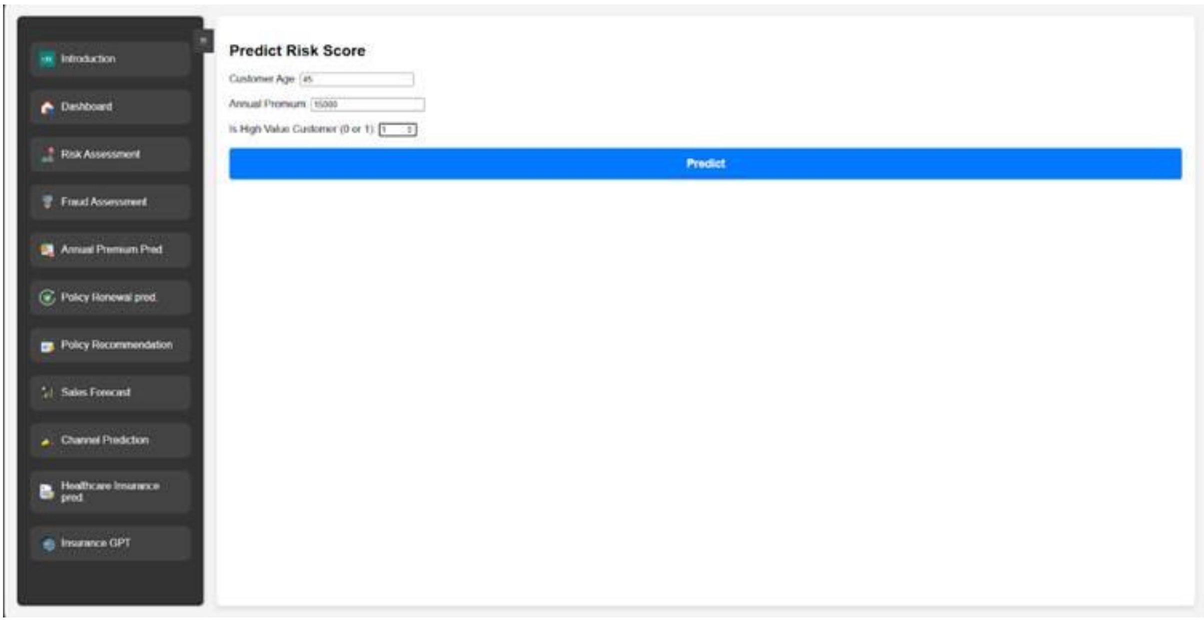

Figure 10.4.1 – Risk Radar

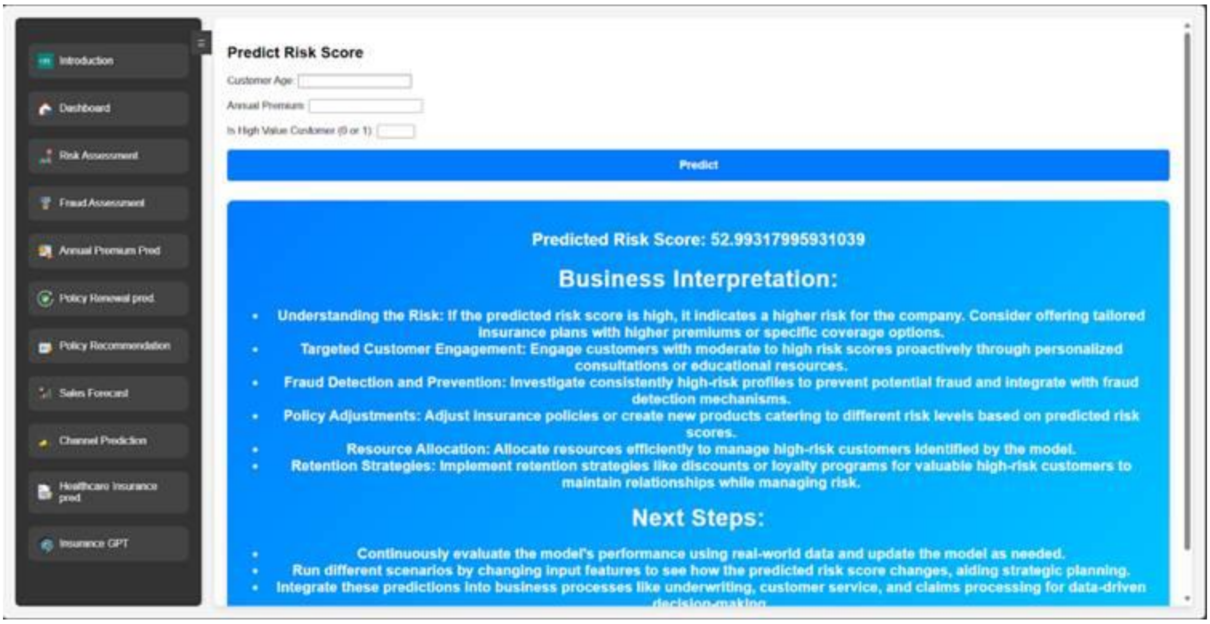

Figure 10.4.2 – Risk Radar

This both above figures 10.4.1 & 10.4.2 explains that evaluates potential risks associated with different insurance products using historical data and statistical models. Helps in managing and mitigating risks to ensure financial stability and protect against losses.

Model Comparison:

Name of the Model	**Accuracy Rate (R^2 Score)**
Random-Forest	78%
Gradient-Boosting	85%
Linear-Regression	89%

Table: 10.4.1

The above table 10.4.1 provides a structured comparison, highlighting the performance and accuracy of various machine learning models applied to the 10.4 modules. For instance, the **Linear Regression** model, with an accuracy rate of **89%**, is selected for implementation due to its superior performance, ensuring optimal predictions and reliability in the module.

11.5 Fraud Shield (Fraud Assessment)

- Purpose: Detects potentially fraudulent activities in insurance claims or policies.
- Uses predictive algorithms to flag suspicious behaviours, enhancing security.

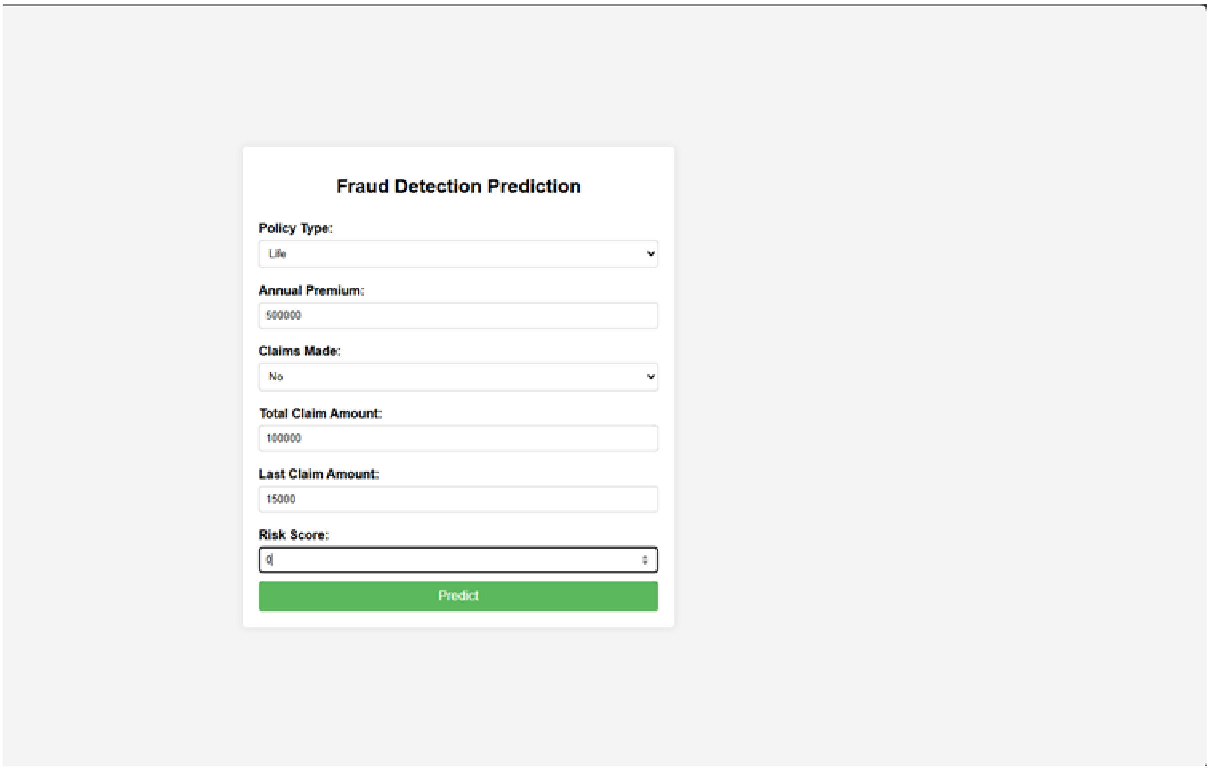

Figure 10.5.1 – Fraud shield

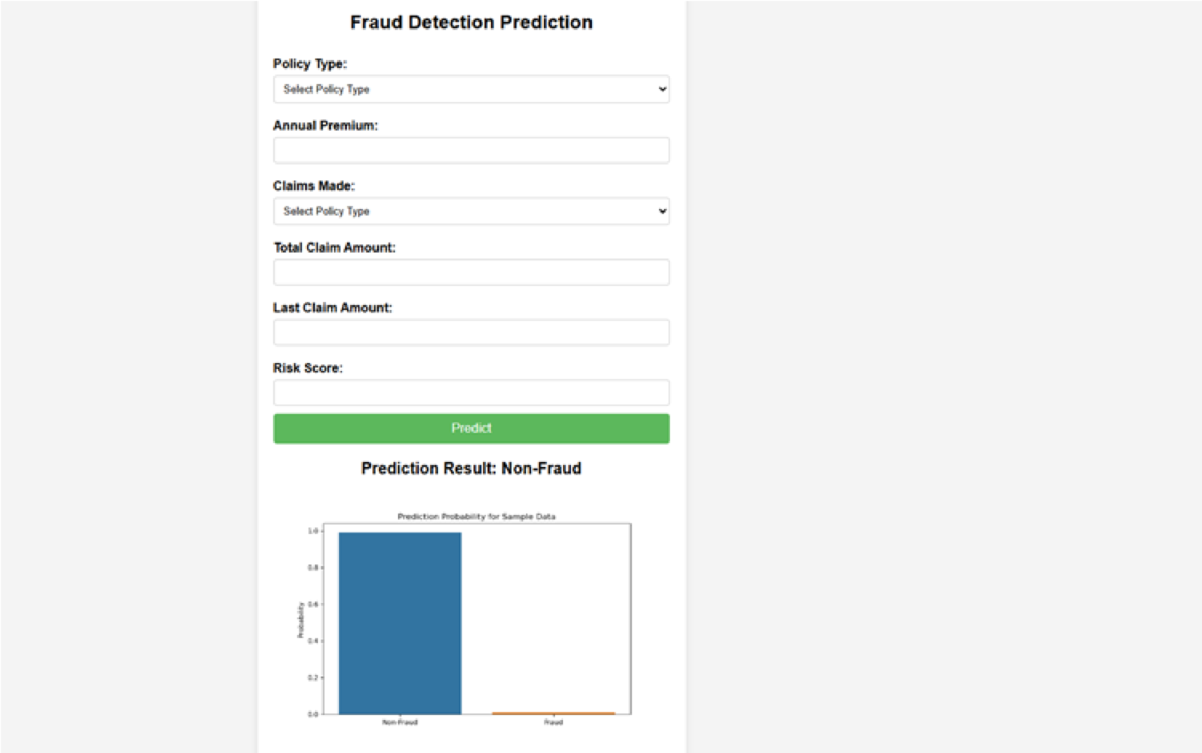

Figure 10.5.2 – Fraud shield

The above image explains the Detects and prevents fraudulent activities by Analyzing patterns and anomalies in insurance claims and transactions. Ensures the integrity of the company's operations and reduces financial losses due to fraud.

Model Comparison:

Name of the Model	**Accuracy Rate (R2 Score)**
Random-Forest	73%
Logistic-Regression	85%
Linear-Regression	82%

Table: 10.5.1

The above table 10.5.1 provides a structured comparison, highlighting the performance and accuracy of various machine learning models applied to the 10.5 modules. For instance, the **Logistic Regression** model, with an accuracy rate of **85%**, is selected for implementation due to its superior performance, ensuring optimal predictions and reliability in the module.

11.6 Renewal Predictor (Policy Renewal Prediction)

- Purpose: Forecasts the likelihood of policy renewals for customers.
- Aids insurers in identifying customers at risk of not renewing and planning retention strategies.

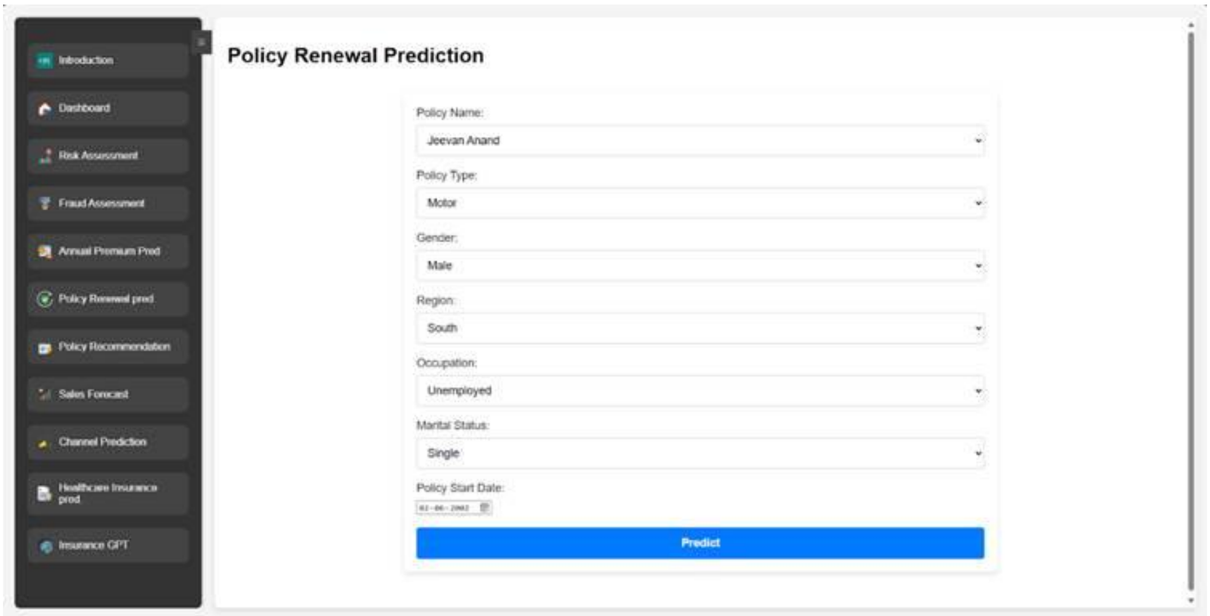

Figure 10.6.1 – Renewal Predictor

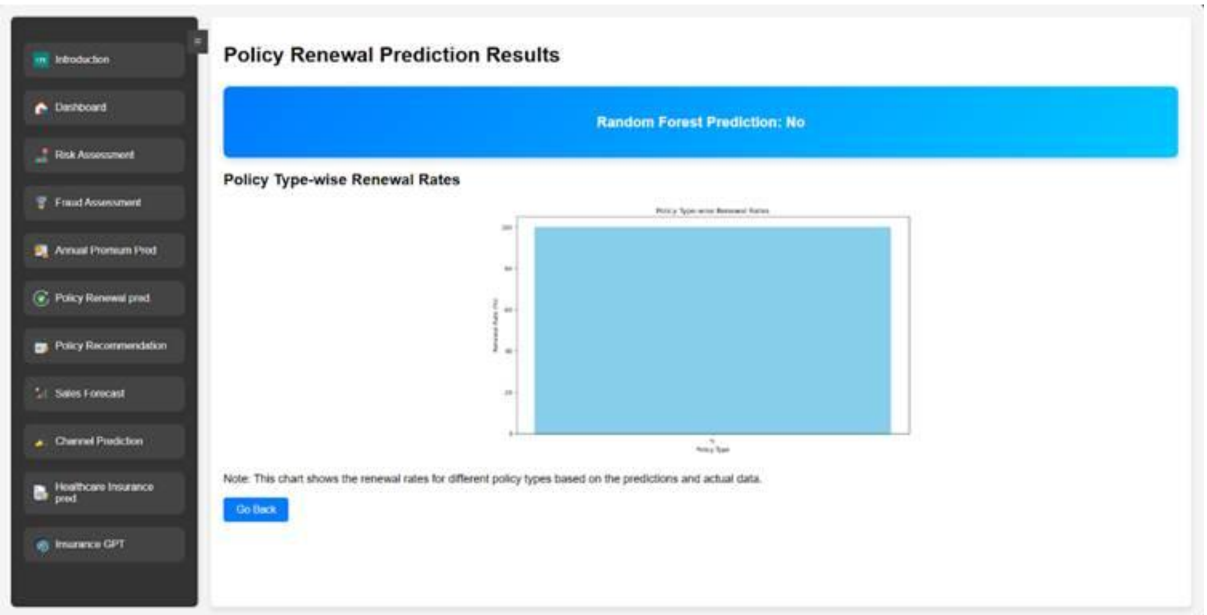

Figure 10.6.2 – Renewal Predictor

The above both image 10.6.1 and 10.6.2 explains the Analyze patterns and trends in policy renewals to forecast future renewal rates. Assists in developing strategies to improve customer retention and streamline the renewal process.

Model Comparison:

Name of the Model	**Accuracy Rate (R2 Score)**
Support Vector Machine	95%
Logistic-Regression	71%
Random Forest	83%

Table: 10.6.1

The above table 10.6.1 provides a structured comparison, highlighting the performance and accuracy of various machine learning models applied to the 10.6 modules. For instance, the **Support Vector Machine** model, with an accuracy rate of **95%**, is selected for implementation due to its superior performance, ensuring optimal predictions and reliability in the module.

11.7 Smart Policy Advisor (Policy Recommendation)

- Purpose: Recommends tailored policies to customers based on their needs and history.
- Leverages customer data and machine learning to optimize policy offerings.

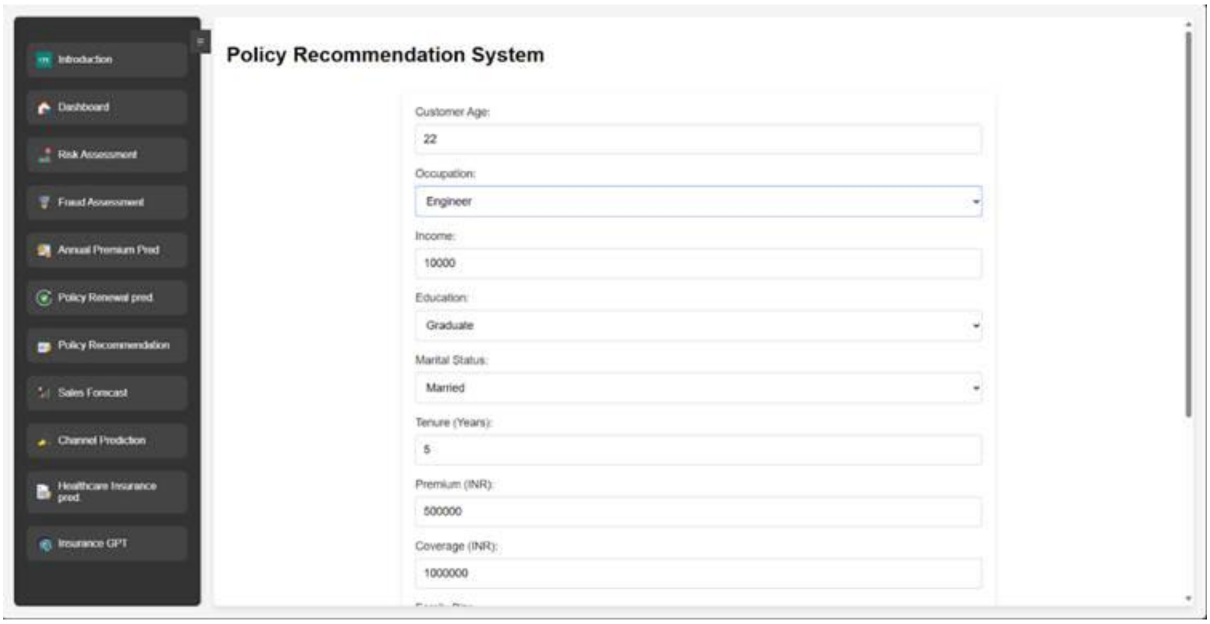

Figure 10.7.1 – Smart Policy Advisor

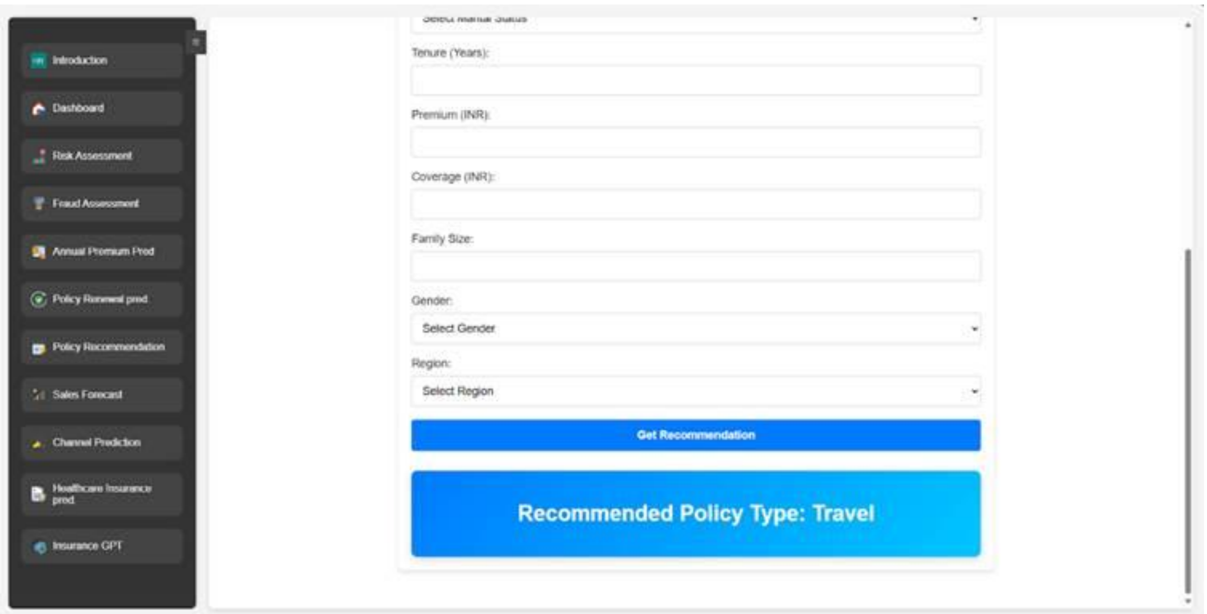

Figure 10.7.2 – Smart Policy Advisor outcome result

The above give figure 10.7.1 & 10.7.2 explains provides personalized insurance policy suggestions based on individual customer profiles, including age, income, and health status. Utilizes advanced algorithms to match users with suitable policies, enhancing their decision-making experience.

Model Comparison:

Name of the Model	Accuracy Rate (R^2 Score)
Support Vector Machine	69%
Logistic-Regression	74%
Random Forest	85%

Table: 10.7.1

The above shown table 10.7.1 provides a structured comparison, highlighting the performance and accuracy of various machine learning models applied to the 10.7 modules. For instance, the **Random Forest** model, with an accuracy rate of **85%**, is selected for implementation due to its superior performance, ensuring optimal predictions and reliability in the module.

11.8 Sales Vision (Sales Forecast)

- Purpose: Projects future sales trend and helps in strategic decision-making.
- Uses ARIMA time series modeling to forecast revenue and sales performance.

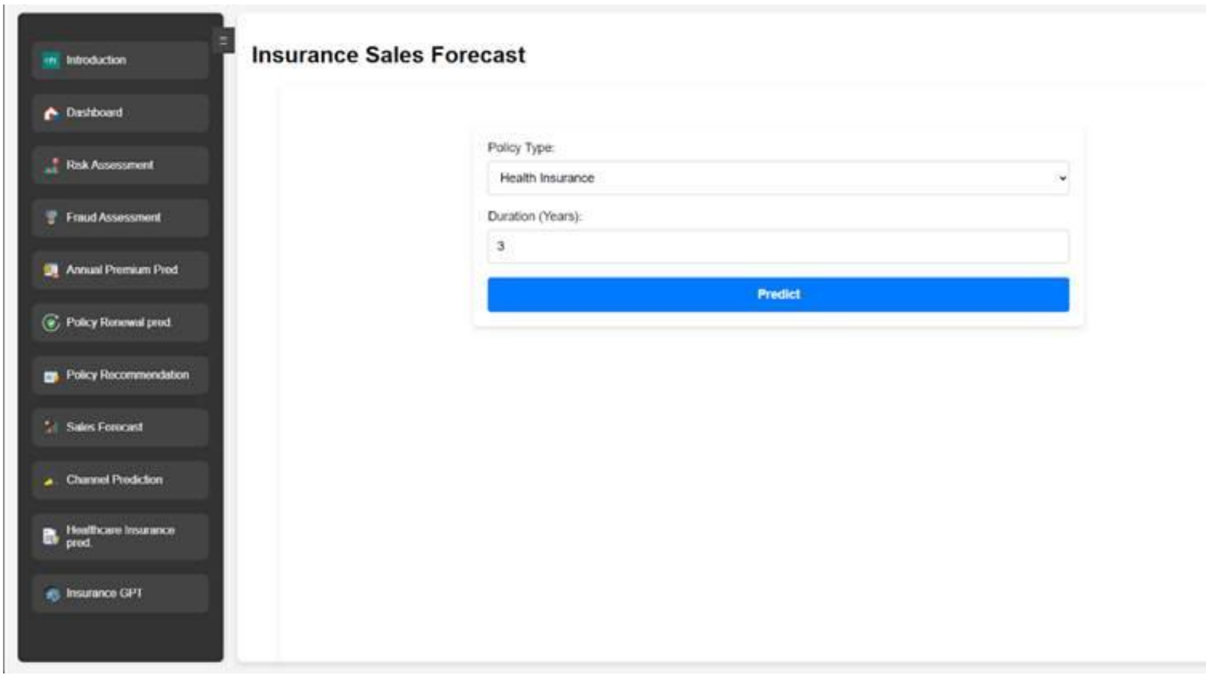

Figure – 10.8.1 – Sales Vision

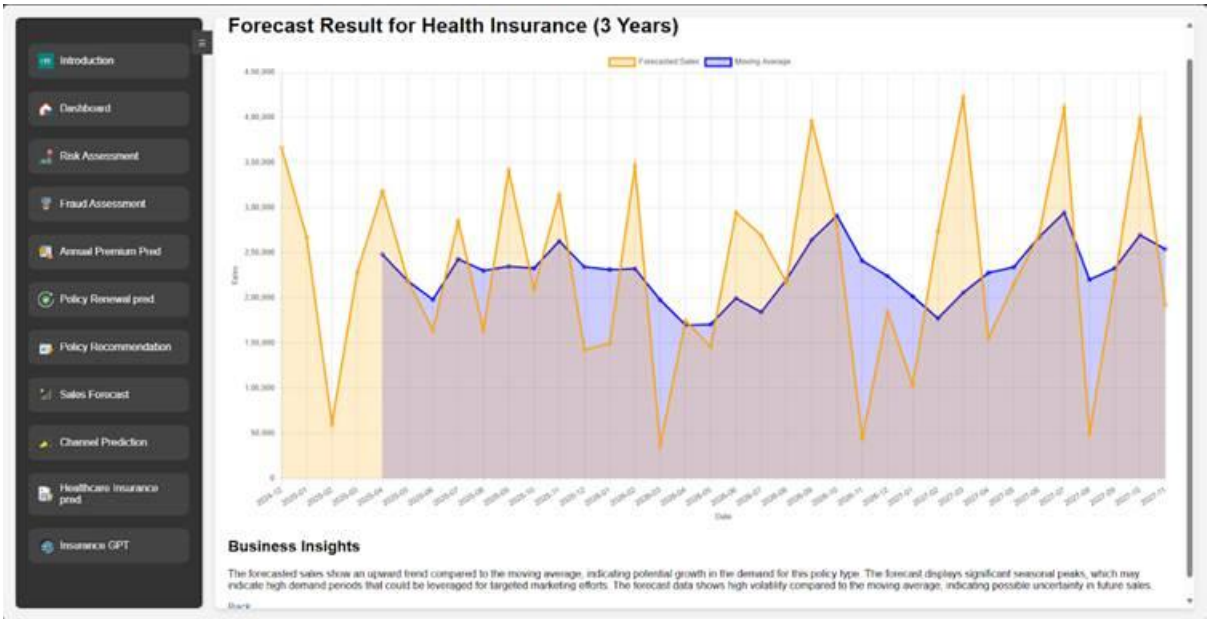

Figure – 10.8.2 – Sales Vision outcome result

The above figure 10.8.1 and 10.8.2 explains the Predicts future sales trends for various insurance products using historical data and market analysis. Supports strategic planning by providing insights into expected sales performance and demand.

11.9 Channel Optimizer (Channel Prediction)

- Purpose: Predicts the most effective marketing channels for customer engagement.
- Enhances marketing efforts by identifying the best channels for target audiences.

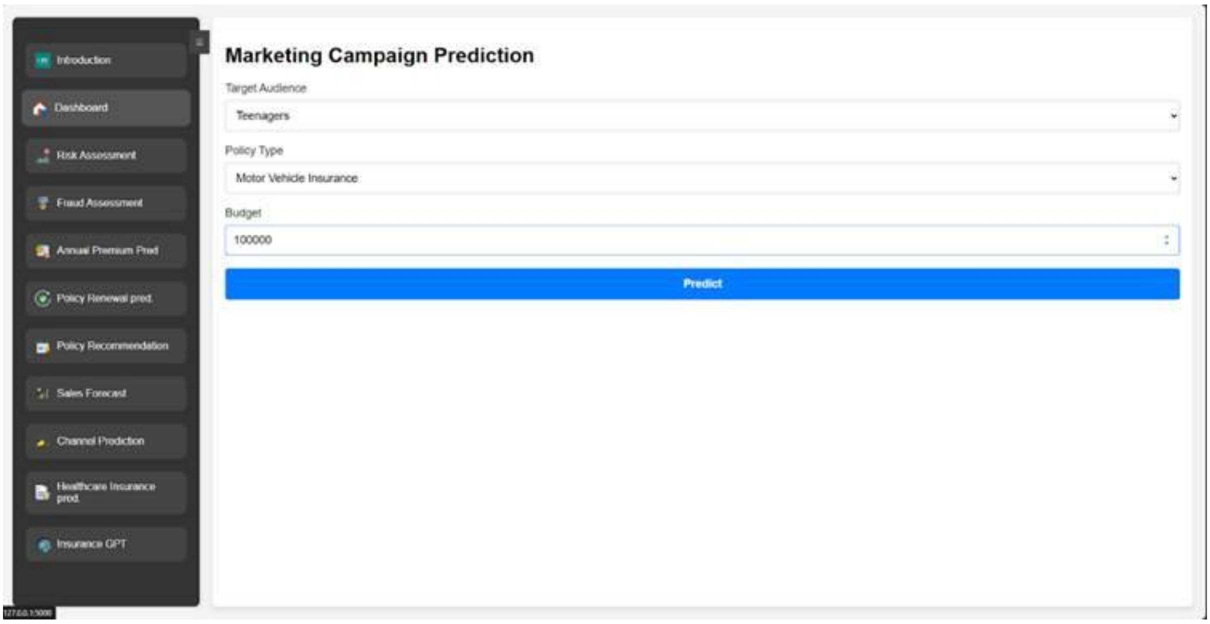

Figure 10.9.1 – Channel Optimizer

Figure 10.9.2 – Channel Optimizer outcome result

The above Figure 10.9.1 and 10.9.2 explains the Forecasts the effectiveness of marketing campaigns by analysing past campaign data and market trends. Aids in optimizing future marketing strategies to maximize engagement and conversion rates.

Model Comparison:

Name of the Model	Accuracy Rate (R2 Score)
Random Forest	85%
XGBoost-Regressor	81%
Logistic-Regressor	79%

Table: 10.9.1

The above table 10.9.1 provides a structured comparison, highlighting the performance and accuracy of various machine learning models applied to the 10.9 modules. For instance, the **Random Forest** model, with an accuracy rate of **85%**, is selected for implementation due to its superior performance, ensuring optimal predictions and reliability in the module.

11.10 Health-Cost Estimator (Healthcare Insurance Predictions)

- Purpose: Predicts medical insurance premiums based on user demographics and behavior.
- Helps customers estimate insurance costs for personalized healthcare plans.

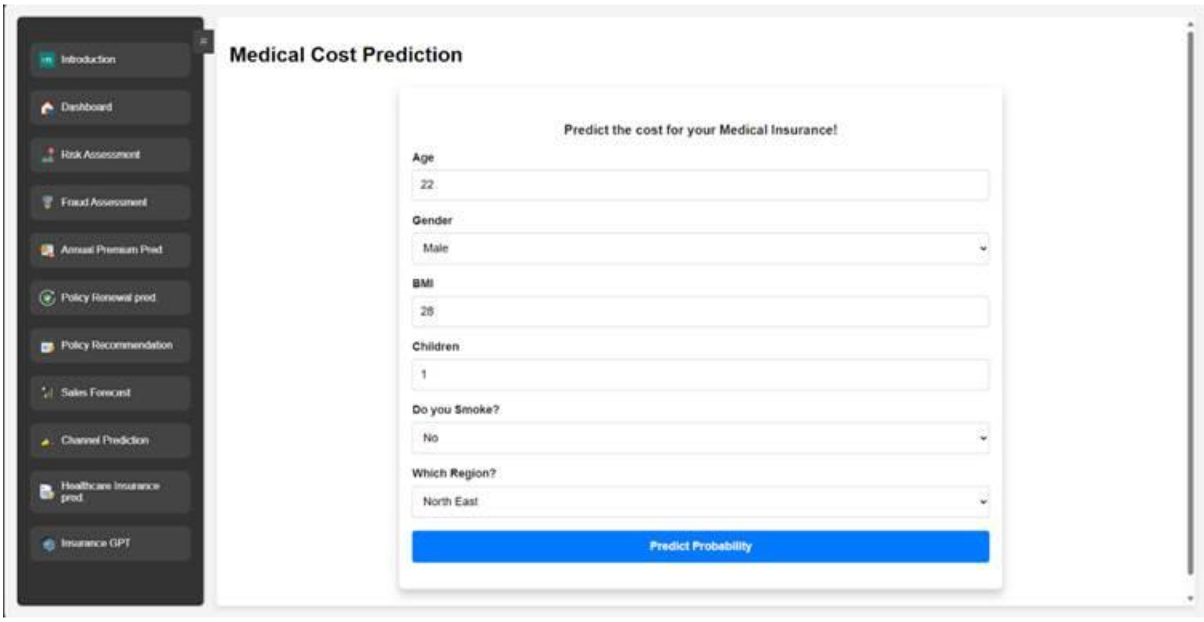

Figure 10.10.1 – Health / Medical cost Estimator

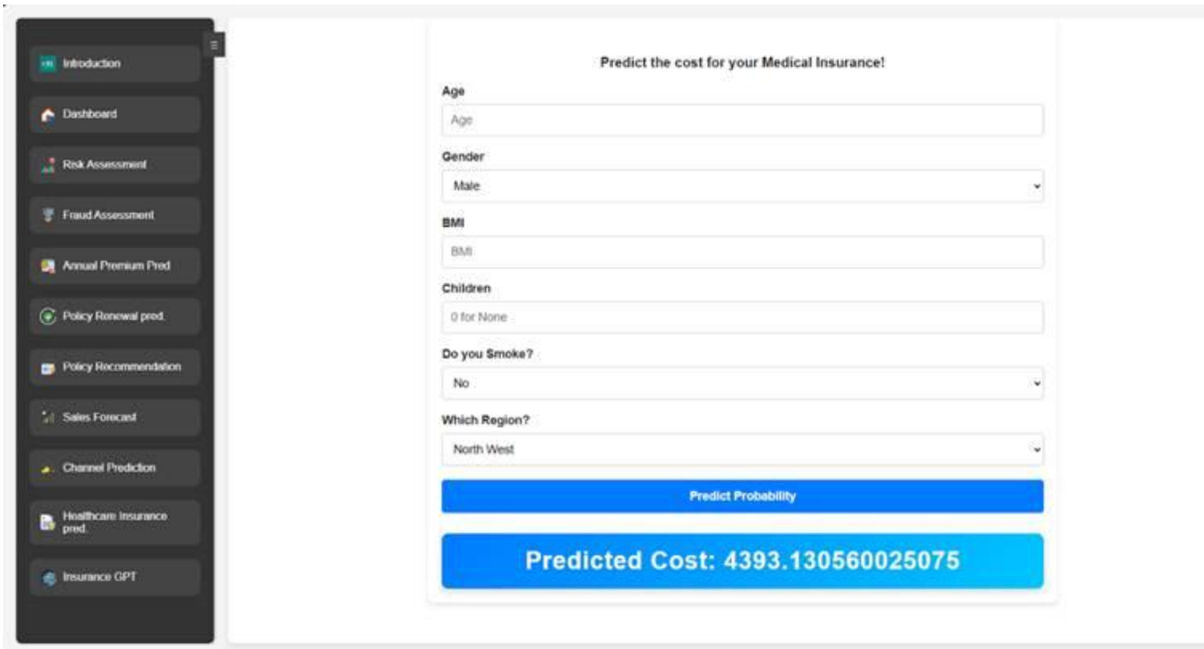

Figure 10.10.2 – Health / Medical cost Estimator outcome results

The image represents a system that forecasts the yearly cost of healthcare insurance based on user-specific information like age, health status, and coverage requirements. It aids customers in estimating their expected expenses, allowing them to make well-informed decisions about their insurance plans.

Model Comparison:

Name of the Model	Accuracy Rate (R2 Score)
Linear-Regression	78%
Support Vector Machine-Reg.	84 %
Random Forest Regressor	88 %

Table 10.10.1: Machine Learning Model Performance Overview

Table 10.10.1 outlines a comparative analysis of various machine learning models applied to the 10.10 modules. Among them, the Random Forest model stands out with an 88% accuracy rate, making it the preferred choice for implementation. Its exceptional performance ensures reliable predictions and consistency in the module's operations.

10.11 Insurance AI Assistant (Insurance GPT)

- **Objective:** This AI-driven virtual assistant is built to respond to user inquiries related to insurance.
- **Functionality:** It uses AI to provide real-time answers to frequently asked questions and detailed information regarding insurance policies.

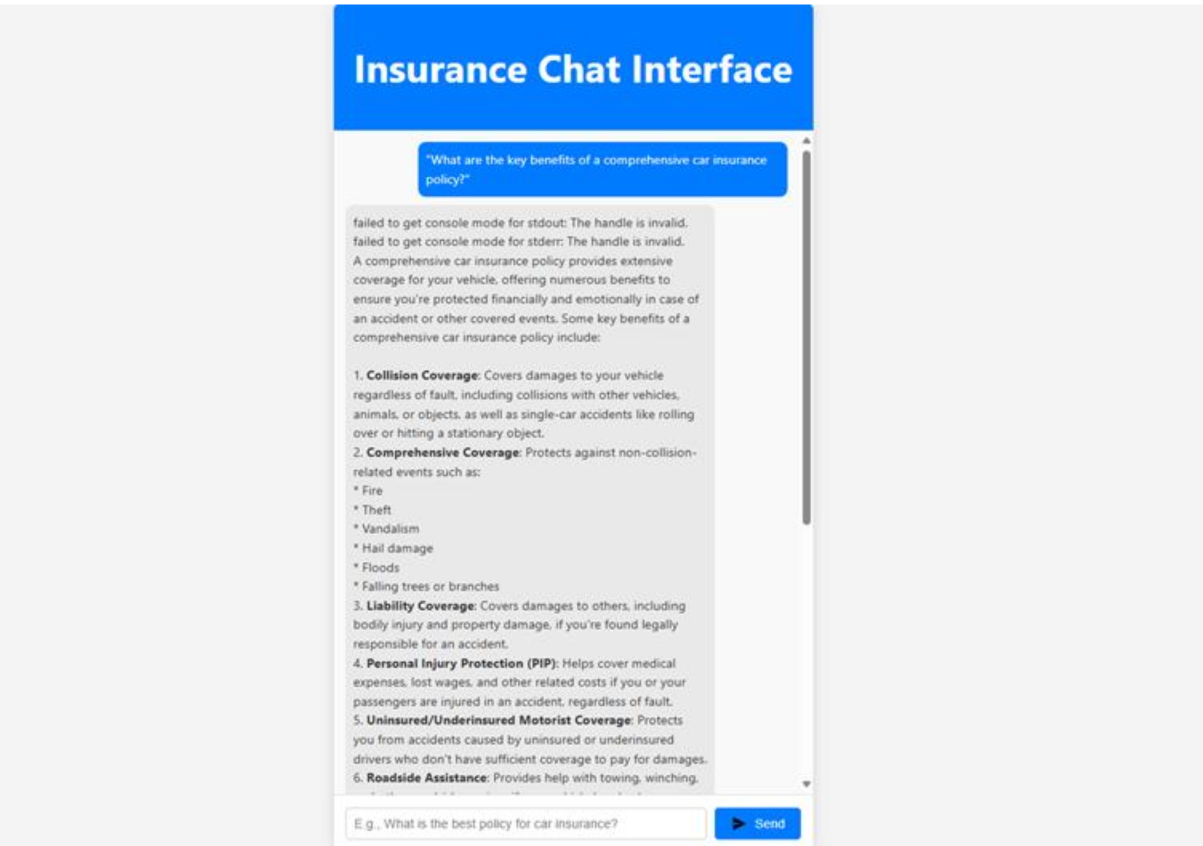

Figure 10.11.1: Insurance GPT Model Overview

Figure 10.11.1 displays the architecture of the Insurance GPT Model, a sophisticated language model designed to address user queries about insurance. It facilitates dynamic interaction by delivering accurate and timely responses on various insurance-related topics, policy details, and coverage options, enhancing user engagement through personalized communication.

12 SOUCRE CODE:

Main Folder: ./app

./app/static

./app/static/**style.css**

```
/*--------------------- Google fonts ------------------- */
@import
url('https://fonts.googleapis.com/css2?family=Inter:wght@100;200;300;400;500;600;700;800;900&
family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,
300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap');


/* --------------------- CSS Global variables ------------*/
:root
{
	/*------------ Colors -------------*/
	--primary-color: #963cdd;     /*---- Purple ----*/
	--secondary-color: #1668b8;   /*---- Blue ------*/
	--third-color: #000000;       /*----- Black ------*/
	--fourth-color: #ffffff;      /*---- White ------*/
	--blue-color: #2540ce;    /*---- Button Blue -----*/
	--yellow-color: #fcb900;  /*---- Button Yello ----*/


	/*------------ Font Size --------------*/
	--heading-font-size: 1.5rem;
	--paragraph-font-size: 1rem;
}

*
{
	margin: 0;
	padding: 0;
	box-sizing: border-box;
}

html
{
	font-size: 16px;
}

body
{
	font-family: 'Poppins', sans-serif;
}

.container
{
	padding: 2rem;
}

.heading
{
	font-size: 2.2rem;
	font-weight: 600;
	line-height: 1.2;
	padding: 1rem 0;
}

.sub-heading
{
	color: var(--secondary-color);
	font-size: 1rem;
	font-weight: 500;
	text-transform: uppercase;
}

img
```

```
{
	max-width: 100%;
	height: auto;
}

.brand img {
	width: 100px; /* Adjust size as needed */
	height: 100px;
	border-radius: 50%; /* Make the image round */
	object-fit: cover; /* Ensure the image fits the circular shape */
	display: block;
}

h1, h2, h3, h4, h5, h6
{
	font-family: 'Poppins', sans-serif;
}

p
{
	font-family: 'Inter', sans-serif;
}

.white
{
	color: #ffffff;
}

.para-line
{
	font-size: 1rem;
	line-height: 1.5;
}

.btn
{
	padding: 0.8rem 2rem;
	border: none;
	border-radius: 4px;
	cursor: pointer;
}

.btn a
{
	font-size: 1rem;
	font-weight: 500;
	text-transform: uppercase;
	text-decoration: none;
	letter-spacing: 1px;
}


/*------------------------- Menu style ------------------------*/
.menu-container
{
  width: 1152px;
  max-width: 90%;
  margin: 0 auto;
}

.nav-wrapper
{
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.nav-wrapper ul.nav-list
{
  list-style-type: none;
  display: flex;
  align-items: center;
  gap: 16px;
}
```

```css
.nav-wrapper ul.nav-list li
{
  margin-left: 30px;
  padding: 20px 0;
  position: relative;
}

.nav-wrapper ul.nav-list li a
{
  color: var(--third-color);
  text-decoration: none;
  letter-spacing: 1px;
  transition: all .5s ease-in-out;
}

.nav-wrapper ul.nav-list li a:hover, .nav-wrapper ul.nav-list li.active a
{
  color: var(--blue-color);
}

nav ul.dropdown-list
{
  list-style-type: none;
  display: block;
  background: whitesmoke;
  padding: 6px 16px;
  position: absolute;
  width: max-content;
  z-index: 9999;
  left: 50%;
  transform: translateX(-50%);
  opacity: 0;
  pointer-events: none;
}

.nav-wrapper ul.dropdown-list li
{
  margin-left: 0;
  padding: 5px 0;
}

.nav-wrapper ul.dropdown-list li a
{
  color: var(--third-color);
}

.nav-wrapper ul.nav-list li:hover .dropdown-list
{
  opacity: 1;
  pointer-events: auto;
  animation: moveUp .5s ease-in-out forwards;
}


.nav-wrapper .nav-list li .btn a
{
        color: var(--fourth-color);
}

.nav-wrapper .nav-list li .btn:hover a
{
        color: var(--third-color);
}

@keyframes moveUp
{
  0%
  {
    opacity: 0;
    transform: translateX(-50%) translateY(50px);
  }
  100%
  {
```

```
    opacity: 1;
    transform: translateX(-50%) translateY(20px);
  }
}

.hamberger
{
  display: none;
}

.mobile .hamberger
{
  display: flex;
  flex-direction: column;
  padding: 20px 0;
  cursor: pointer;
}

.mobile .hamberger span
{
  background: var(--third-color);
  width: 28px;
  height: 2px;
  margin-bottom: 8px;
}

.mobile ul.nav-list
{
  background: -webkit-linear-gradient(45deg, #f5f6fa, #dcdde1);
  background: linear-gradient(45deg, #f5f6fa, #dcdde1);
  position: fixed;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  padding-top: 80px;
  opacity: 0;
  pointer-events: none;
  transition: All .3s ease-in-out;
}

.hamberger, .brand
{
  z-index: 9999;
}

.mobile ul.nav-list.open
{
  opacity: 1;
  pointer-events: auto;
  z-index: 999;
  overflow-y: auto;
}


.mobile .hamberger span
{
  transform-origin: left;
  transition: all .3s ease-in-out;
}

.mobile ul.nav-list li a
{
  font-size: 20px;
}

.mobile ul.dropdown-list
{
  position: relative;
  background: transparent;
  text-align: center;
  height: 0;
```

```
  overflow-y: hidden;
  transition: opacity 1s ease-in-out;
  padding-top: 0;
}

.mobile .nav-wrapper ul li:hover .dropdown-list
{
  height: max-content;
  padding-top: 6px;
}

.mobile ul.nav-list li
{
  margin-left: 0;
  text-align: center;
}

.mobile .nav-wrapper ul.dropdown-list li a
{
  color: #7f8fa6;
}

.mobile .nav-wrapper ul.dropdown-list li a:hover
{
	color: var(--third-color);
}
/*-------------------------- Menu style -------------------------*/


/*----------- Scroll to top --------------*/
#topBtn
{
  position: fixed;
  bottom: 40px;
  right: 40px;
  font-size: 22px;
  width: 40px;
  height: 40px;
  background: var(--blue-color);
  color: #white;
  border: none;
  cursor: pointer;
  display: none;
}

#topBtn ion-icon
{
	color: #fff;
}
/*----------- Scroll to top --------------*/

/*-------- Blue button ---------*/
.btn-blue
{
	background: var(--blue-color);
	transition: 0.3s ease-in-out;
}

.btn-blue:hover
{
	background: var(--yellow-color);
}

.btn-blue a
{
	color: var(--fourth-color);
}

.btn-blue:hover > a
{
	color: var(--third-color);
}

.btn-blue
```

```
{
        background: var(--blue-color);
        transition: 0.3s ease-in-out;
}

.btn-blue:hover
{
        background: var(--yellow-color);
}

.btn-blue a
{
        color: var(--fourth-color);
}

.btn-blue:hover > a
{
        color: var(--third-color);
}

/*---------- Yellow btn ----------*/
.btn-yellow
{
        background: var(--yellow-color);
        transition: 0.3s ease-in-out;
}

.btn-yellow:hover
{
        background: var(--blue-color);
}

.btn-yellow > a
{
        color: #000;
}

.btn-yellow:hover > a
{
        color: var(--fourth-color);
}

.btn-yellow
{
        background: var(--yellow-color);
        transition: 0.3s ease-in-out;
}

.btn-yellow:hover
{
        background: var(--blue-color);
}

.btn-yellow a
{
        color: var(--third-color);
}

.btn-yellow:hover > a
{
        color: var(--fourth-color);
}

/*------------ Full width button ------------*/
.btn-full-w
{
        padding: 1.2rem 2rem;
        display: block;
        width: 100%;
        box-shadow: rgba(0, 0, 0, 0.15) 0px 5px 15px 0px;
}
```

```css
/*----------------- Hero section styling ----------------*/
.hero
{
        background: linear-gradient(rgba(150, 60, 221, 0.4), rgba(22, 104, 184, 0.9)),
url("../img/hero-bg.jpg");
        background-position: center top;
        background-repeat: no-repeat;
        background-size: cover;
}

.hero .hero-container
{
        width: 100%;
        height: 90vh;
        display: flex;
        justify-content: flex-start;
        align-items: flex-end;
}

.hero-container .row > .col
{
        display: flex;
        flex-direction: column;
        gap: 1.4rem;
}

.hero-content
{
        padding: 0 2rem 3.6rem 2rem;
}

.hero-heading
{
        font-size: 2.3rem;
        line-height: 1.1;
}

.inner-row .inner-col
{
        margin: 1rem 0;
}
/*----------------- Hero section styling ----------------*/


/*----------------- Why Us section styling ---------------*/
.why-us
{
        background-image: linear-gradient(
          155deg,
          hsl(215deg 100% 98%) 0%,
          hsl(215deg 100% 98%) 30%,
          hsl(215deg 100% 98%) 38%,
          hsl(215deg 100% 98%) 43%,
          hsl(215deg 100% 98%) 47%,
          hsl(215deg 100% 98%) 48%,
          hsl(215deg 100% 98%) 50%,
          hsl(215deg 100% 98%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 50%,
          hsl(215deg 100% 99%) 51%,
          hsl(215deg 100% 100%) 53%,
          hsl(215deg 100% 100%) 55%,
          hsl(215deg 100% 100%) 60%,
          hsl(215deg 100% 100%) 68%,
          hsl(0deg 0% 100%) 96%
        );
}
```

```
.why-us-col ion-icon
{
        font-size: 2rem;
        color: var(--fourth-color);
        background: var(--yellow-color);
        padding: 1rem;
        border: none;
        border-radius: 50px;
}

.why-us-highlight-heading
{
        font-size: 1.4rem;
        font-weight: 600;
}

.lead-form
{
        border: none;
        border-radius: 12px;
        margin: 2.5rem 0;
        padding: 3rem 1.6rem;
        box-shadow: rgba(0, 0, 0, 0.15) 0px 5px 15px 0px;
}

.input-field
{
        display: flex;
        flex-direction: column;
        margin: 1rem 0;
}

.input-field label
{
        font-size: 0.8rem;
        font-weight: 600;
        margin: 0.3rem 0;
        text-transform: uppercase;
}

.input-field input
{
        font-size: 1rem;
        border: none;
        border-radius: 5px;
        padding: 1rem;
        background: #f3f8ff;
}
/*------------------ Why Us section styling ----------------*/



/*------------------ Services section styling ----------------*/
.service-img
{
        border-radius: 5px;
        margin: 0 0 2rem 0;
}

.our-services .services
{
        margin: 2rem 0;
}

.services .service
{
        border: none;
        border-radius: 10px;
        margin: 1.5rem 0;
        padding: 1.8rem;
        box-shadow: rgba(0, 0, 0, 0.1) 0px 20px 25px -5px, rgba(0, 0, 0, 0.04) 0px 10px 10px -
5px;
}
```

```css
.services .service:hover
{
	box-shadow: rgba(0, 0, 0, 0.25) 0px 25px 50px -12px;
}

.service ion-icon
{
	color: var(--yellow-color);
	font-size: 2.6rem;
}

.service .service-heading
{
	font-weight: 600;
}
/*----------------- Services section styling ---------------*/



/*----------------- Overline section styling ---------------*/
.overline
{
	background: linear-gradient(rgba(10, 17, 79, 0.9), rgba(10, 17, 79, 0.9)), 
url("../img/hero-bg.jpg");
	background-position: center top;
	background-repeat: no-repeat;
	background-size: cover;
	text-align: center;
}

.insurance-policies
{
	border-radius: 6px;
}

.overlines .row .col
{
	margin: 2.6rem 0;
}

.overlines .row .col ion-icon
{
	font-size: 2.6rem;
}

.overline-heading
{
	color: var(--yellow-color);
	font-size: 1.3rem;
	font-weight: 600;
}
/*----------------- Overline section styling ---------------*/


/*----------------- About us section styling ---------------*/
.about-highlights
{
	margin: 2rem 0;
}

.about-highlight-line
{
	display: flex;
	align-items: center;
	gap: 0.5rem;
	margin: 1rem 0;
}

.about-highlight-line ion-icon
{
	color: var(--yellow-color);
	font-size: 1.6rem;
}
```

```
.highlight-line-heading
{
        font-size: 1rem;
        font-weight: 600;
}

.about-img
{
        border-radius: 6px;
        margin-top: 4rem;
}

.partners
{
        margin: 2rem 0;
}
/*----------------- About us section styling ---------------*/



/*----------------- Testimonial section styling --------------*/
.testimonial
{
        background: linear-gradient(rgba(10, 17, 79, 0.9), rgba(10, 17, 79, 0.9)),
url("../img/hero-bg.jpg");
        background-position: center top;
        background-repeat: no-repeat;
        background-size: cover;
        text-align: center;
}

.testimonial-profile
{
        display: flex;
        justify-content: center;
        align-items: center;
        gap: 1rem;
        margin: 1rem;
}

.profile-img
{
        border-radius: 50px;
}

.client-name
{
        font-size: 1rem;
        font-weight: 600;
        text-align: left;
}

.client-location
{
        text-align: left;
}

.stars ion-icon
{
        color: var(--yellow-color);
}
/*----------------- Testimonial section styling --------------*/


/*----------------- Agent card styling -----------------------*/
.agent-card
{
        text-align: center;
}

.agent-img
{
        border-radius: 50%;
}
```

```
.agent-name
{
	font-size: 2rem;
	font-weight: 600;
}

.agent-number, .agent-email
{
	margin: 1rem 0;
	font-size: 0.9rem;
}
/*----------------- Agent card styling -----------------------*/


/*----------------- Footer styling ------------------*/
footer
{
	background: #251963;
	display: flex;
	justify-content: center;
	align-items: center;
	padding: 0.6rem 0;
}
/*----------------- Footer styling ------------------*/





/*--------------------------------------------------------------*/
/*-------------------- Desktop Screen Styling ------------------*/
/*--------------------------------------------------------------*/
@media screen and (min-width: 789px)
{
	.container, .hero-container
	{
		max-width: 1180px;
		margin: 0 auto;
	}

	.container
	{
		padding: 4rem 0;
	}

	.heading
	{
		font-size: 3.2rem;
	}

	.para-line
	{
		line-height: 1.8;
	}

	.sub-heading
	{
		font-size: 1rem;
	}

	.row
	{
		display: flex;
		flex-direction: row;
		justify-content: space-between;
		align-items: center;
		gap: 2rem;
	}

	.row .col
	{
		width: 100%;
```

```
	}

	.inner-row
	{
		display: flex;
		gap: 2rem;
	}


	/*-------------- Hero section styling ------------------*/
	.hero-container .row > .hero-content
	{
		width: 140%;
	}

	.hero-heading
	{
		font-size: 4rem;
	}

	.hero
	{
		background: linear-gradient(rgba(150, 60, 221, 0.2), rgba(13, 14, 56, 0.9)),
url("../img/hero-bg.jpg");
		background-position: center top;
		background-repeat: no-repeat;
		background-size: cover;
	}
	/*-------------- Hero section styling ------------------*/


	/*-------------- Why us styling ---------------------*/
	.why-us .container .row
	{
		gap: 5rem;
	}

	.why-us .container .row .why-us-content
	{
		width: 150%;
	}

	.why-us-content .inner-row
	{
		margin-top: 2rem;
	}

	.lead-form
	{
		background: var(--fourth-color);
		padding: 3rem !important;
	}

	.input-field
	{
		margin: 1.5rem 0;
	}
	/*-------------- Why us styling ---------------------*/


	/*-------------- Services section styling --------------*/
	.our-services
	{
		text-align: center;
	}

	.our-services .container .head-desc
	{
		max-width: 66%;
		margin: 0 auto;
	}

	.services, .partners-grid
	{
```

```
                display: grid;
                grid-template-rows: repeat(2, 1fr);
                grid-template-columns: repeat(3, 1fr);
                grid-gap: 2rem;
        }
        /*-------------- Services section styling --------------*/



        /*-------------- About section styling ------------------*/
        .about .container .row
        {
                gap: 4rem;
        }

        .partners
        {
                margin-top: 4rem;
        }
        /*-------------- About section styling ------------------*/


        /*-------------- Testimonial section styling ---------------*/
        .testimonial .container .para-line
        {
                max-width: 840px;
                margin: 0 auto;
                font-size: 1.1rem;
        }

        .testimonial
        {
                padding: 4rem 0;
        }
        /*-------------- Testimonial section styling ---------------*/
}
```

./app/static/Userstyle.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
    box-sizing: border-box;
}

.container {
    display: flex;
    height: calc(100vh - 40px); /* Adjust for top and bottom margins */
    margin: 20px; /* Margin around the container */
}

.sidebar {
    width: 250px;
    background: #333;
    color: #fff;
    padding: 20px;
    box-shadow: 2px 0 5px rgba(0, 0, 0, 0.2);
    position: relative;
    transition: width 0.3s;
    border-radius: 8px;
    overflow-y: auto; /* Enable vertical scrolling */
}

.sidebar ul {
    list-style: none;
    padding: 0;
    margin: 0;
}

.sidebar.collapsed {
    width: 80px; /* Width when collapsed */
```

```
}

.sidebar.collapsed ul {
    overflow-y: hidden; /* Hide overflow when collapsed */
}

.sidebar .toggle-btn {
    position: absolute;
    top: 20px;
    right: -16px;
    background: #444;
    border: none;
    color: #fff;
    padding: 10px;
    cursor: pointer;
}

.sidebar ul li {
    margin: 20px 0;
}

.sidebar ul li a {
    color: #fff;
    text-decoration: none;
    display: flex;
    align-items: center;
    padding: 15px;
    border-radius: 10px;
    background: #444;
    transition: background 0.3s, transform 0.3s;
}

.sidebar ul li a:hover {
    background: #555;
    transform: scale(1.05);
}

.sidebar ul li img {
    width: 24px;
    height: 24px;
    margin-right: 10px;
}

.sidebar.collapsed li a .text {
    display: none;
}

.content {
    flex: 1;
    padding: 20px;
    background: #fff;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    margin-left: 20px; /* Space between sidebar and content */
    overflow: auto; /* Add scroll bars if content overflows */
    height: 100%; /* Ensure the content takes full height available */
    box-sizing: border-box; /* Include padding and border in element's total width and height
*/
}

.card-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 20px;
    background: #fff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
    transition: box-shadow 0.3s, transform 0.3s;
}

.card-container:hover {
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);
```

```
    transform: scale(1.02);
}

.card-header {
    font-size: 1.5em;
    margin-bottom: 15px;
    border-bottom: 2px solid #eee;
    padding-bottom: 10px;
}

.card-body {
    font-size: 1em;
    color: #666;
    margin-bottom: 15px;
}

.card-footer {
    text-align: right;
}

.button {
    display: inline-block;
    padding: 10px 20px;
    background: #007bff;
    color: #fff;
    text-decoration: none;
    border-radius: 5px;
    transition: background 0.3s;
}

.button:hover {
    background: #0056b3;
}

.table-wrapper {
    overflow-x: auto;
    margin-top: 20px;
}

table {
    width: 100%;
    border-collapse: collapse;
}

table th, table td {
    padding: 10px;
    border: 1px solid #ddd;
}

table th {
    background: #f4f4f4;
    text-align: left;
}

table tbody tr:nth-child(even) {
    background: #f9f9f9;
}

table tbody tr:hover {
    background: #f1f1f1;
}

/* Form-Controls */
/* Form Container */
.form-container {
    max-width: 60%;
    margin: 40px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    transition: box-shadow 0.3s ease, transform 0.3s ease;
}
```

```css
/* Hover Effect */
.form-container:hover {
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    transform: scale(1.02);
}

/* Subheading Style */
.subheading {
    font-size: 1.25em;
    color: #333;
    margin-bottom: 20px;
    text-align: center;
}

/* Form-Controls */
.form-group {
    margin-bottom: 20px;
}

.form-group label {
    display: block;
    font-size: 1.1em;
    margin-bottom: 8px;
    color: #333;
}

.form-group input[type="number"],
.form-group select {
    width: 100%;
    padding: 12px;
    font-size: 1.1em;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-sizing: border-box;
    transition: border-color 0.3s;
}

.form-group input[type="number"]:focus,
.form-group select:focus {
    border-color: #007bff;
    outline: none;
}

/* Button Styling */
button[type="submit"] {
    width: 100%;
    padding: 14px;
    font-size: 1.1em;
    font-weight: bold;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button[type="submit"]:hover {
    background-color: #0056b3;
}

/* Result Styling */
.result {
    margin-top: 30px;
    padding: 20px;
    text-align: center;
    background: linear-gradient(135deg, #007bff, #00c6ff);
    border-radius: 10px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
    color: #fff;
    font-size: 1.4em;
    font-weight: bold;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
```

```
/* Adding a hover effect */
.result:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 24px rgba(0, 0, 0, 0.2);
}

/* Styling the text within the result */
.result h4 {
    margin: 0;
    font-size: 1.6em;
    letter-spacing: 1px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1);
}

/* Tesgt */
.chart img {
    max-width: 100%; /* Ensure the image scales within its container */
    height: auto; /* Maintain aspect ratio */
    max-height: 400px; /* Adjust the maximum height of the chart */
    display: block;
    margin: 0 auto; /* Center the image horizontally */
}

.note {
    margin-top: 20px;
    font-size: 1.1em;
}

.chart-container {
    margin-top: 20px;
    position: relative;
    width: 100%;
    height: 400px; /* Adjust as needed */
}

/* Fraud Tips */
.fraud-tips {
    background-color: #f9d6d5;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
}

.fraud-tips h3 {
    color: #d9534f;
    font-size: 1.25em;
    margin-bottom: 10px;
}

.fraud-tips ul {
    list-style-type: disc;
    padding-left: 20px;
}

/* Improvement Tips */
.improvement-tips {
    background-color: #d5f9d6;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.improvement-tips h3 {
    color: #5bc0de;
    font-size: 1.25em;
    margin-bottom: 10px;
}

.improvement-tips ul {
    list-style-type: disc;
    padding-left: 20px;
}
```

Main Folder: ./app/templates

./app/templates/**channel_route.py**

```
import os
from flask import Blueprint, render_template, request, redirect, url_for, session
import pandas as pd
# import numpy as np
# import pickle
import io
import base64
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

channel_bp = Blueprint('channel', __name__)

# Construct the path to the dataset
dataset_path = os.path.join(os.path.dirname(__file__), 'data',
'updated_marketing_campaign_dataset.csv')
df = pd.read_csv(dataset_path)

possible_target_audiences = df['TargetAudience'].unique().tolist()
possible_policy_types = df['PolicyType'].unique().tolist()

le_target_audience = LabelEncoder()
le_target_audience.fit(possible_target_audiences)
le_policy_type = LabelEncoder()
le_policy_type.fit(possible_policy_types)
le_channel = LabelEncoder()
le_channel.fit(df['Channel'].unique())

# Encode the dataset
df['Channel'] = le_channel.transform(df['Channel'])
df['TargetAudience'] = le_target_audience.transform(df['TargetAudience'])
df['PolicyType'] = le_policy_type.transform(df['PolicyType'])

# Define input features and target variables
X = df[['TargetAudience', 'PolicyType', 'Budget']]
y_channel = df['Channel']
y_growth_rate = df['GrowthRate']
y_roi = df['ROI']
y_retention_rate = df['CustomerRetentionRate']

# Split the data
X_train, X_test, y_train_channel, y_test_channel = train_test_split(X, y_channel,
test_size=0.2, random_state=42)
_, _, y_train_growth_rate, y_test_growth_rate = train_test_split(X, y_growth_rate,
test_size=0.2, random_state=42)
_, _, y_train_roi, y_test_roi = train_test_split(X, y_roi, test_size=0.2, random_state=42)
_, _, y_train_retention_rate, y_test_retention_rate = train_test_split(X, y_retention_rate,
test_size=0.2, random_state=42)

# Initialize and train models
model_channel = RandomForestClassifier(n_estimators=100, random_state=42)
model_growth_rate = RandomForestRegressor(n_estimators=100, random_state=42)
model_roi = RandomForestRegressor(n_estimators=100, random_state=42)
model_retention_rate = RandomForestRegressor(n_estimators=100, random_state=42)

model_channel.fit(X_train, y_train_channel)
model_growth_rate.fit(X_train, y_train_growth_rate)
model_roi.fit(X_train, y_train_roi)
model_retention_rate.fit(X_train, y_train_retention_rate)

@channel_bp.route('/channelpred', methods=['GET', 'POST'])
def channelpredgwt():

    # Get the role from the session
```

```
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('main.login'))  # Redirect to login if no role is found

    if request.method == 'POST':
        target_audience = request.form.get('target_audience')
        policy_type = request.form.get('policy_type')
        budget = float(request.form.get('budget'))

        # Process the input
        example_input = {
            'TargetAudience': le_target_audience.transform([target_audience])[0],
            'PolicyType': le_policy_type.transform([policy_type])[0],
            'Budget': budget
        }

        input_df = pd.DataFrame([example_input])

        # Predict outcomes
        predicted_channel = le_channel.inverse_transform(model_channel.predict(input_df))
        predicted_growth_rate = model_growth_rate.predict(input_df)[0]
        predicted_roi = model_roi.predict(input_df)[0]
        predicted_retention_rate = model_retention_rate.predict(input_df)[0]

        # Create visualizations
        fig, axes = plt.subplots(1, 3, figsize=(18, 5))
        sns.barplot(x=['Growth Rate'], y=[predicted_growth_rate], ax=axes[0])
        axes[0].set_ylim(0, 20)
        axes[0].set_title('Predicted Growth Rate')

        sns.barplot(x=['ROI'], y=[predicted_roi], ax=axes[1])
        axes[1].set_ylim(0, 3)
        axes[1].set_title('Predicted ROI')

        sns.barplot(x=['Customer Retention Rate'], y=[predicted_retention_rate], ax=axes[2])
        axes[2].set_ylim(60, 100)
        axes[2].set_title('Predicted Customer Retention Rate')

        # Save plot to a BytesIO object
        img = io.BytesIO()
        plt.savefig(img, format='png')
        plt.close(fig)
        img.seek(0)
        plot_url = base64.b64encode(img.getvalue()).decode()

        # Interpretation note
        interpretation = (
            f"Based on the latest analysis, here’s what we predict for your marketing 
strategy:\n\n"
            f"<br><br><b>Marketing Channel:</b> Our model suggests that the most effective 
channel for your campaign is likely to be <b><i>{predicted_channel[0]}</i></b>. This choice 
could help you reach your target audience more effectively and drive better results.\n\n"
            f"<br><br><b>Growth Rate:</b> We anticipate a growth rate of approximately 
<b><i>{predicted_growth_rate:.2f}%</i></b>. This indicates a positive trend in your marketing 
efforts, with expected growth in your target metrics.\n\n"
            f"<br><br><b>Return on Investment (ROI):</b> The predicted ROI stands at 
<b><i>{predicted_roi:.2f}%</i></b>. This figure reflects the efficiency of your marketing 
investments and suggests a favorable return relative to your expenditures.\n\n"
            f"<br><br><b>Customer Retention Rate:</b> We forecast a customer retention rate 
of about <b><i>{predicted_retention_rate:.2f}%</i></b>. This high rate demonstrates strong 
customer loyalty and satisfaction, indicating that your marketing strategies are resonating 
well with your audience.\n\n"
            f"<br><br><i>Leveraging these insights will help you refine your approach and 
optimize your marketing strategies for even greater success.</i>"
        )

        return render_template('channelpred-mod/index.html', plot_url=plot_url, 
interpretation=interpretation, role=user_role)

    return render_template('channelpred-mod/index.html', plot_url=None, interpretation=None, 
role=user_role)
```

Folder Name: ./app/templates/**eda_route.py**

```
from flask import Blueprint, render_template, redirect, url_for, session
import pandas as pd
import seaborn as sns
import io
import base64

import matplotlib
matplotlib.use('Agg')  # Use a non-interactive backend
import matplotlib.pyplot as plt

# Create a blueprint
eda_bp = Blueprint('eda', __name__)

# Load and prepare the dataset
data = pd.read_csv('app/data/eda-data.csv')
data['Policy_Start_Date'] = pd.to_datetime(data['Policy_Start_Date'])
data['Policy_Start_Year'] = data['Policy_Start_Date'].dt.year

policy_trend = data.groupby(['Policy_Start_Year',
'Policy_Type']).size().reset_index(name='Policy_Count')

def plot_to_base64(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png')
    buf.seek(0)
    img_data = base64.b64encode(buf.getvalue()).decode('utf-8')
    plt.close(fig)  # Close the figure after saving
    return img_data

@eda_bp.route('/edat')
def edat():
    # Get the role from the session
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))  # Redirect to login if no role is found

    # Generate various plots and pass them to the template
    fig, ax = plt.subplots()
    sns.countplot(x='Policy_Type', data=data, ax=ax)
    policy_types_img = plot_to_base64(fig)
    plt.close(fig)

    # Distribution of policy types
    fig, ax = plt.subplots()
    sns.countplot(x='Policy_Type', data=data, ax=ax)
    ax.set_title('Distribution of Policy Types')
    policy_types_img = plot_to_base64(fig)
    plt.close(fig)

    # Age distribution
    fig, ax = plt.subplots()
    sns.histplot(data['Customer_Age'], kde=True, ax=ax)
    ax.set_title('Distribution of Customer Ages')
    age_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Gender distribution
    fig, ax = plt.subplots()
    sns.countplot(x='Gender', data=data, ax=ax)
    ax.set_title('Distribution of Customer Genders')
    gender_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Region distribution
    fig, ax = plt.subplots()
    sns.countplot(x='Region', data=data, ax=ax)
    ax.set_title('Distribution of Customers by Region')
    region_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Occupation distribution
    fig, ax = plt.subplots()
```

```
    sns.countplot(x='Occupation', data=data, ax=ax)
    ax.set_title('Distribution of Customers by Occupation')
    occupation_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Marital status distribution
    fig, ax = plt.subplots()
    sns.countplot(x='Marital_Status', data=data, ax=ax)
    ax.set_title('Distribution of Customers by Marital Status')
    marital_status_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Annual premium distribution
    fig, ax = plt.subplots()
    sns.histplot(data['Annual_Premium'], kde=True, ax=ax)
    ax.set_title('Distribution of Annual Premiums')
    annual_premium_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy start date distribution
    fig, ax = plt.subplots()
    sns.histplot(data['Policy_Start_Date'], kde=True, ax=ax)
    ax.set_title('Distribution of Policy Start Dates')
    start_date_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy tenure distribution
    fig, ax = plt.subplots()
    sns.histplot(data['Tenure'], kde=True, ax=ax)
    ax.set_title('Distribution of Policy Tenures')
    tenure_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Coverage amount distribution
    fig, ax = plt.subplots()
    sns.histplot(data['Coverage_Amount'], kde=True, ax=ax)
    ax.set_title('Distribution of Coverage Amounts')
    coverage_amount_dist_img = plot_to_base64(fig)
    plt.close(fig)

    # Trend analysis of policy types
    fig, ax = plt.subplots(figsize=(14, 7))
    sns.lineplot(data=policy_trend, x='Policy_Start_Year', y='Policy_Count',
hue='Policy_Type', marker='o', ax=ax)
    ax.set_title('Trend Analysis of Policy Types Based on Year')
    ax.set_xlabel('Year')
    ax.set_ylabel('Number of Policies')
    ax.legend(title='Policy Type', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xticks(rotation=45)
    trend_analysis_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy Type vs. Gender
    fig, ax = plt.subplots(figsize=(14, 7))
    sns.countplot(data=data, x='Policy_Type', hue='Gender', ax=ax)
    ax.set_title('Policy Type vs. Gender')
    policy_gender_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy Type vs. Region
    fig, ax = plt.subplots(figsize=(14, 7))
    sns.countplot(data=data, x='Policy_Type', hue='Region', ax=ax)
    ax.set_title('Policy Type vs. Region')
    policy_region_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy Type vs. Occupation
    fig, ax = plt.subplots(figsize=(14, 7))
    sns.countplot(data=data, x='Policy_Type', hue='Occupation', ax=ax)
    ax.set_title('Policy Type vs. Occupation')
    policy_occupation_img = plot_to_base64(fig)
    plt.close(fig)

    # Policy Type vs. Marital Status
```

```
    fig, ax = plt.subplots(figsize=(14, 7))
    sns.countplot(data=data, x='Policy_Type', hue='Marital_Status', ax=ax)
    ax.set_title('Policy Type vs. Marital Status')
    policy_marital_status_img = plot_to_base64(fig)
    plt.close(fig)

    return render_template('eda_mod/index.html',
                           policy_types_img=policy_types_img,
                           age_dist_img=age_dist_img,
                           gender_dist_img=gender_dist_img,
                           region_dist_img=region_dist_img,
                           occupation_dist_img=occupation_dist_img,
                           marital_status_dist_img=marital_status_dist_img,
                           annual_premium_dist_img=annual_premium_dist_img,
                           start_date_dist_img=start_date_dist_img,
                           tenure_dist_img=tenure_dist_img,
                           coverage_amount_dist_img=coverage_amount_dist_img,
                           trend_analysis_img=trend_analysis_img,
                           policy_gender_img=policy_gender_img,
                           policy_region_img=policy_region_img,
                           policy_occupation_img=policy_occupation_img,
                           policy_marital_status_img=policy_marital_status_img,
                           insights=generate_insights(), role=user_role)

def generate_insights():
    insights = {}

    # Insights for distribution of policy types
    insights['policy_types'] = (
        "The distribution of policy types reveals the popularity of different policies. "
        "For instance, if 'Motor' policies are the most common, it may indicate a strong
market presence in automotive insurance."
    )

    # Insights for age distribution
    insights['age_dist'] = (
        "The age distribution of customers helps in understanding the age groups that are
most engaged with the insurance products. "
        "A peak in certain age ranges might suggest targeted marketing opportunities."
    )

    # Insights for gender distribution
    insights['gender_dist'] = (
        "Gender distribution can provide insights into the market's demographic split. "
        "Significant imbalances might suggest potential areas for more inclusive marketing
strategies."
    )

    # Insights for region distribution
    insights['region_dist'] = (
        "The distribution of customers across regions shows which areas have the highest
engagement with the insurance policies. "
        "This information is valuable for regional marketing and resource allocation."
    )

    # Insights for occupation distribution
    insights['occupation_dist'] = (
        "Understanding the distribution of customers by occupation can highlight which
professional groups are more likely to purchase insurance. "
        "This can guide targeted product offerings."
    )

    # Insights for marital status distribution
    insights['marital_status_dist'] = (
        "Marital status distribution provides insights into customer life stages, which can
influence insurance needs and preferences."
    )

    # Insights for annual premium distribution
    insights['annual_premium_dist'] = (
        "The distribution of annual premiums shows the range of spending by customers. "
        "A high concentration in certain ranges could indicate price sensitivity or premium
affordability."
    )
```

```
    # Insights for policy start date distribution
    insights['start_date_dist'] = (
        "The distribution of policy start dates can help in understanding seasonality trends
and planning for policy renewals."
    )

    # Insights for policy tenure distribution
    insights['tenure_dist'] = (
        "Policy tenure distribution indicates how long customers typically stay with the
insurance provider. "
        "Longer tenures might suggest higher customer satisfaction and loyalty."
    )

    # Insights for coverage amount distribution
    insights['coverage_amount_dist'] = (
        "Coverage amount distribution helps in understanding the value of policies held by
customers. "
        "Higher coverage amounts may suggest a more affluent customer base or higher risk
coverage."
    )

    # Insights for trend analysis
    insights['trend_analysis'] = (
        "Trend analysis of policy types over the years can show how customer preferences have
evolved. "
        "For example, an increasing trend in 'Health' policies might indicate growing health
consciousness among customers."
    )

    return insights
```

Folder Name: ./app/templates/**forms.py**

```
from flask_wtf import FlaskForm
from wtforms import StringField, EmailField, SubmitField
from wtforms.validators import DataRequired, Email

class ConsultancyForm(FlaskForm):
    name = StringField('Full Name', validators=[DataRequired()])
    email = EmailField('Email Address', validators=[DataRequired(), Email()])
    submit = SubmitField('Get A Quote')
```

Folder Name: ./app/templates/**fraud_route.py**

```
from flask import Blueprint, render_template, request, redirect, url_for, session
import pickle
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

# Define the fraud blueprint
fraud_bp = Blueprint('fraud', __name__, url_prefix='/fraud')

# Load the saved RandomForest model and encoders
fraud_model_path = 'app/models/fraud_mod/RandomForest_model.pkl'
fraud_encoder_path = 'app/models/fraud_mod/label_encoders.pkl'

with open(fraud_model_path, 'rb') as f:
    fraud_rf_model = pickle.load(f)

with open(fraud_encoder_path, 'rb') as f:
    fraud_label_encoders = pickle.load(f)

# Fraud Prediction Route
@fraud_bp.route('/fraudpred', methods=['GET', 'POST'])
def fraudpredt():
```

```
    # Get the role from the session
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))  # Redirect to login if no role is found

    if request.method == 'POST':
        # Retrieve form data
        policy_type = int(request.form['policy_type'])
        annual_premium = float(request.form['annual_premium'])
        claims_made = int(request.form['claims_made'])
        total_claim_amount = float(request.form['total_claim_amount'])
        last_claim_amount = float(request.form['last_claim_amount'])
        risk_score = float(request.form['risk_score'])

        # Prepare the sample data
        sample_data = {
            'Policy_Type': policy_type,
            'Annual_Premium': annual_premium,
            'Claims_Made': claims_made,
            'Total_Claim_Amount': total_claim_amount,
            'Last_Claim_Amount': last_claim_amount,
            'Risk_Score': risk_score
        }
        sample_df = pd.DataFrame([sample_data])

        # Predict using the loaded model
        predicted = fraud_rf_model.predict(sample_df)
        predicted_proba = fraud_rf_model.predict_proba(sample_df)

        # Map the prediction back to the original label names
        fraud_mapping = {0: 'Non-Fraud', 1: 'Fraud'}
        predicted_label = fraud_mapping[predicted[0]]

        # Save the plot as an image
        plt.figure(figsize=(8, 5))
        sns.barplot(x=['Non-Fraud', 'Fraud'], y=predicted_proba[0])
        plt.title('Prediction Probability for Sample Data')
        plt.ylabel('Probability')
        plot_path = os.path.join('app', 'static', 'result_img',
'fraud_prediction_plot.png')  # Change to the correct path
        plt.savefig(plot_path)
        plt.close()

        return render_template('fraud-mod/index.html', predicted_label=predicted_label,
plot_path=plot_path, role=user_role)

    return render_template('fraud-mod/index.html', predicted_label=None, role=user_role)
```

Folder Name: ./app/templates/**healthcare_route.py**

```
import os
import pickle
import numpy as np
from flask import Blueprint, render_template, request, redirect, url_for, session

# Create a blueprint for healthcare predictions
health_bp = Blueprint('health', __name__)

# Load the trained model
model_path = os.path.join(os.path.dirname(__file__), 'models',
'healthcare_mod','rf_tuned.pkl')
with open(model_path, 'rb') as file:
    modelrf = pickle.load(file)

# Route for Annual Premium Predictions
@health_bp.route('/healthpred', methods=['GET', 'POST'])
def healthpredcost():

    # Get the role from the session
    user_role = session.get('role')
```

```
    if not user_role:
        return redirect(url_for('main.login'))  # Redirect to login if no role is found

    prediction = None
    if request.method == 'POST':
        # Get input values from the form
        age = float(request.form['age'])
        gender = int(request.form['Gender'])
        bmi = float(request.form['bmi'])
        children = int(request.form['children'])
        smoker = int(request.form['smoker'])
        region = int(request.form['region'])

        # Prepare the input for the model
        input_features = np.array([[age, gender, bmi, children, smoker, region]])

        # Predict
        prediction = modelrf.predict(input_features)[0]

    return render_template('healthcare-mod/index.html', pred=prediction, role=user_role)
```

Folder Name: ./app/templates/**models.py**

```
# app/db.py
from flask_pymongo import PyMongo
from . import mongo

users_collection = mongo.db.users
```

Folder Name: ./app/templates/**policyrecommend_route.py**

```
# policyrec_routes.py
from flask import Blueprint, render_template, request, session, redirect, url_for
import pandas as pd
import pickle
import numpy as np
import os

policy_recommend_bp = Blueprint('policy_recommend', __name__)

# Load the model and label encoders
model_path = os.path.join('app', 'models', 'policyrecommend_mod', 'Recommend_model.pkl')
le_path = os.path.join('app', 'models', 'policyrecommend_mod',
'Recommend_label_encoders.pkl')
df_path = os.path.join('app', 'data', 'Policy-recommend.csv')

with open(model_path, 'rb') as model_file:
    policy_model = pickle.load(model_file)

with open(le_path, 'rb') as le_file:
    policy_label_encoders = pickle.load(le_file)

# Load dataset
df = pd.read_csv(df_path)

# Extract unique values for dropdowns
unique_values = {}
for column in ['Occupation', 'Education', 'Marital Status', 'Gender', 'Region']:
    unique_values[column] = df[column].unique()

@policy_recommend_bp.route('/policypred', methods=['GET', 'POST'])
def policypredt():
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Extract data from the form
```

```
        data = {
            'Customer Age': [int(request.form['customer_age'])],
            'Occupation': [request.form['occupation']],
            'Income': [int(request.form['income'])],
            'Education': [request.form['education']],
            'Marital Status': [request.form['marital_status']],
            'Tenure (Years)': [int(request.form['tenure'])],
            'Premium (INR)': [int(request.form['premium'])],
            'Coverage (INR)': [int(request.form['coverage'])],
            'Family Size': [int(request.form['family_size'])],
            'Gender': [request.form['gender']],
            'Region': [request.form['region']]
        }

        # Create DataFrame
        sample_data = pd.DataFrame(data)

        # Apply label encoding to categorical columns
        for column in ['Occupation', 'Education', 'Marital Status', 'Gender', 'Region']:
            sample_data[column] =
policy_label_encoders[column].transform(sample_data[column])

        # Predict the policy type
        predicted_policy_type = policy_model.predict(sample_data)
        predicted_policy_type = policy_label_encoders['Policy
Type'].inverse_transform(predicted_policy_type)

        return render_template('policyrecommend-mod/index.html', unique_values=unique_values,
prediction=predicted_policy_type[0], role=user_role)

    return render_template('policyrecommend-mod/index.html', unique_values=unique_values,
role=user_role)
```

Folder Name: ./app/templates/**policyrenewal_route.py**

```
import os
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from flask import Blueprint, render_template, redirect, url_for, request, session

policyrenewal_bp = Blueprint('policyrenewal', __name__)

# Load models and label encoders
model_path = os.path.join('app', 'models', 'renewal_mod')

with open(os.path.join(model_path, 'RandomForest_model.pkl'), 'rb') as f:
    rf_model = pickle.load(f)

with open(os.path.join(model_path, 'LogisticRegression_model.pkl'), 'rb') as f:
    lr_model = pickle.load(f)

with open(os.path.join(model_path, 'SVC_model.pkl'), 'rb') as f:
    svc_model = pickle.load(f)

with open(os.path.join(model_path, 'label_encoders.pkl'), 'rb') as f:
    label_encoders = pickle.load(f)

def preprocess_input(data):
    if 'Policy_Start_Date' in data.columns:
        data['Policy_Start_Year'] = pd.to_datetime(data['Policy_Start_Date']).dt.year
        data['Policy_Start_Year'] = data['Policy_Start_Year'].astype(int)
        data.drop(columns=['Policy_Start_Date'], errors='ignore', inplace=True)

    for column, le in label_encoders.items():
        if column in data.columns:
            data[column] = le.transform(data[column])

    return data

def plot_renewal_chart(X_test_with_predictions, data):
```

```
    X_test_with_predictions['Policy_Type'] = data.loc[X_test_with_predictions.index,
'Policy_Type']

    policy_type_renewals = X_test_with_predictions.groupby('Policy_Type').agg(
        Total_Count=('Actual', 'size'),
        Renewed_Count=('Actual', 'sum'),
        Predicted_Renewed_Count=('Predicted', 'sum')
    ).reset_index()

    policy_type_renewals['Renewal_Rate'] = policy_type_renewals['Renewed_Count'] /
policy_type_renewals['Total_Count'] * 100

    plt.figure(figsize=(10, 6))
    plt.bar(policy_type_renewals['Policy_Type'].astype(str),
policy_type_renewals['Renewal_Rate'], color='skyblue')
    plt.xlabel('Policy Type')
    plt.ylabel('Renewal Rate (%)')
    plt.title('Policy Type-wise Renewal Rates')
    plt.xticks(rotation=45)
    plt.tight_layout()

    chart_path = os.path.join('app', 'static', 'result_img', 'renewal_chart.png')
    plt.savefig(chart_path)
    plt.close()

    return chart_path

@policyrenewal_bp.route('/policyrenewal', methods=['GET', 'POST'])
def renewal():

    # Get the role from the session
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))  # Redirect to login if no role is found

    if request.method == 'POST':
        input_data = {
            'Policy_Name': [request.form['Policy_Name']],
            'Policy_Type': [request.form['Policy_Type']],
            'Gender': [request.form['Gender']],
            'Region': [request.form['Region']],
            'Occupation': [request.form['Occupation']],
            'Marital_Status': [request.form['Marital_Status']],
            'Policy_Start_Date': [request.form['Policy_Start_Date']]
        }

        df = pd.DataFrame(input_data)
        df = preprocess_input(df)

        rf_prediction = rf_model.predict(df)[0]
        lr_prediction = lr_model.predict(df)[0]
        svc_prediction = svc_model.predict(df)[0]

        X_test_with_predictions = df.copy()
        X_test_with_predictions['Actual'] = [1]  # Example: Assume actual renewal
        X_test_with_predictions['Predicted'] = [rf_prediction]

        chart_path = plot_renewal_chart(X_test_with_predictions, df)

        return render_template('policyrenewal-mod/result.html',
                               rf_prediction='Yes' if rf_prediction == 1 else 'No',
                               lr_prediction='Yes' if lr_prediction == 1 else 'No',
                               svc_prediction='Yes' if svc_prediction == 1 else 'No',
                               chart_path=chart_path, role=user_role)

    return render_template('policyrenewal-mod/index.html', role=user_role)
```

Folder Name: ./app/templates/**risk_route.py**

```
import pickle
import pandas as pd
```

```
from flask import Blueprint, render_template, redirect, url_for, session, request

# Create a Blueprint
risk_bp = Blueprint('risk', __name__)

# Load the trained model
with open('app/models/risk_mod/linear_regression_model.pkl', 'rb') as file:
    risk_model = pickle.load(file)

# Define the route for risk prediction
@risk_bp.route('/riskpred', methods=['GET', 'POST'])
def riskpredt():

    # Get the role from the session
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))  # Redirect to login if no role is found

    if request.method == 'POST':
        # Get input data from the form
        customer_age = int(request.form['Customer_Age'])
        annual_premium = float(request.form['Annual_Premium'])
        is_high_value_customer = int(request.form['Is_High_Value_Customer'])

        # Create DataFrame for model input
        input_data = pd.DataFrame({
            'Customer_Age': [customer_age],
            'Annual_Premium': [annual_premium],
            'Is_High_Value_Customer': [is_high_value_customer]
        })

        # Make prediction
        prediction = risk_model.predict(input_data)[0]

        # Render template with prediction
        return render_template('risk-mod/index.html', prediction=prediction, role=user_role)

    return render_template('risk-mod/index.html', prediction=None, role=user_role)
```

Folder Name: ./app/templates/**routes.py**

```
from flask import Blueprint, render_template, request, redirect, url_for, flash, session
import socket
from flask_mail import Mail, Message
from bson.objectid import ObjectId
from pymongo import MongoClient
from datetime import datetime
import os

from werkzeug.security import check_password_hash, generate_password_hash

from . import mongo, mail  # Import 'mail' from your '__init__.py'

# Create a Blueprint
main_bp = Blueprint('main', __name__)

# MongoDB Configuration
client = MongoClient(os.environ.get('MONGO_URI'))  # Get Mongo URI from .env
db = client['login_system']
users = db['users']  # Assuming 'users' is the collection name

# Decorator to restrict access based on user roles
def login_required(f):
    def wrapper(*args, **kwargs):
        if 'email' not in session:
            flash('Please log in first.', 'danger')
            return redirect(url_for('main.login'))
        return f(*args, **kwargs)
    wrapper.__name__ = f.__name__
    return wrapper
```

```
def role_required(role):
    def decorator(f):
        def wrapper(*args, **kwargs):
            user = users.find_one({"email": session['email']})
            if user and user['role'] != role:
                flash(f'Access denied for {role}s only.', 'danger')
                return redirect(url_for('main.dashboard'))
            return f(*args, **kwargs)
        wrapper.__name__ = f.__name__
        return wrapper
    return decorator

# Function to check if the system has internet connection
def check_internet_connection():
    try:
        # Check if we can resolve the host for internet connectivity
        socket.create_connection(("www.google.com", 80), 2)
        return True
    except OSError:
        return False

@main_bp.route('/')
def index():
    return render_template('guest/index.html')

@main_bp.route('/consultancy', methods=['POST'])
def consultancy():
    if request.method == 'POST':
        name = request.form.get('name')
        email = request.form.get('email')

        # Ensure mongo.db.consultancy is the correct collection
        mongo.db.consultancy.insert_one({
            'name': name,
            'email': email
        })

        return redirect(url_for('main.index'))

@main_bp.route('/about')
def about():
    return render_template('guest/about.html')

@main_bp.route('/insurance')
def insurance():
    return render_template('guest/insurance.html')

@main_bp.route('/news-Insurance')
def newsinsurance():
    return render_template('guest/news.html')

@main_bp.route('/contact')
def contact():
    return render_template('guest/contact.html')

# @main_bp.route('/login', methods=['GET', 'POST'])
# def login():
#     if request.method == 'POST':
#         email = request.form['email']
#         password = request.form['password']

#         user = users.find_one({"email": email})

#         if user and check_password_hash(user['password'], password):
#             session['email'] = email
#             session['role'] = user['role']
#             flash('Login successful!', 'success')
#             send_email(email, user['email'], "Login successful")
#             return redirect(url_for('main.dashboard'))
#         else:
#             flash('Invalid credentials, please try again.', 'danger')
#             send_email(email, email, "Login failed")
#             return redirect(url_for('main.login'))
```

```
#     return render_template('users/login.html')

@main_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Check for internet connection before allowing login
        if not check_internet_connection():
            flash('No internet connection. Please check your network and try again.',
'danger')
            return redirect(url_for('main.login'))  # Prevent login and reload login page

        # Proceed with login process if internet is available
        email = request.form['email']
        password = request.form['password']

        user = users.find_one({"email": email})

        if user and check_password_hash(user['password'], password):
            session['email'] = email
            session['role'] = user['role']
            flash('Login successful!', 'success')
            send_email(email, user['email'], "Login successful")
            return redirect(url_for('main.dashboard'))
        else:
            flash('Invalid credentials, please try again.', 'danger')
            send_email(email, email, "Login failed")
            return redirect(url_for('main.login'))

    return render_template('users/login.html')

@main_bp.route('/dashboard')
@login_required
def dashboard():
    role = session['role']
    return render_template('users/dashboard.html', role=role)

# Function to send emails
def send_email(to, username, status):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    html_body = f"""
    <p>Dear {username},</p>
    <p>Your login attempt was: <strong>{status}</strong></p>
    <p>Timestamp: {timestamp}</p>
    """

    msg = Message('Login Status', recipients=[to])
    msg.html = html_body
    mail.send(msg)

# Logout route
@main_bp.route('/logout')
@login_required
def logout():
    session.clear()
    flash('You have been logged out.', 'success')
    return redirect(url_for('main.login'))

# Admin panel
# Admin dashboard to manage users
@main_bp.route('/admin_dashboard')
@login_required
@role_required('admin')
def admin_dashboard():
    all_users = users.find()

    # Query MongoDB to get user count and admin count
    user_count = mongo.db.users.count_documents({})  # Count all users
    admin_count = mongo.db.users.count_documents({"role": "admin"})  # Count all admins
    monthly_growth = 20  # Replace with actual calculation logic if needed

    return render_template('admin/admin_dashboard.html', users=all_users,
user_count=user_count,
                           admin_count=admin_count, monthly_growth=monthly_growth)
```

```
# User Mngts:

# Admin can create new users
@main_bp.route('/create_user', methods=['POST'])
@login_required
@role_required('admin')
def create_user():
    email = request.form['email']
    password = request.form['password']
    role = request.form['role']

    # Check if the user already exists
    existing_user = mongo.db.users.find_one({'email': email})

    if existing_user:
        flash('User with this email already exists.', 'danger')
        return redirect(url_for('main.admin_dashboard'))

    # Prepare user data with a 'created_at' field
    new_user = {
        'email': email,
        'password': generate_password_hash(password),  # Hash the password
        'role': role,
        'created_at': datetime.utcnow()  # Add the current UTC time
    }

    # Insert the new user into the MongoDB collection
    mongo.db.users.insert_one(new_user)

    flash(f'User {email} created successfully.', 'success')
    return redirect(url_for('main.admin_dashboard'))

# Notes:
# admin@admin.com - admin123
# remaining user - 123

# Admin can delete users
@main_bp.route('/delete_user/<user_id>')
@login_required
@role_required('admin')
def delete_user(user_id):
    users.delete_one({'_id': ObjectId(user_id)})
    flash('User deleted successfully.', 'success')
    return redirect(url_for('main.admin_dashboard'))

# Content-Mngt:
@main_bp.route('/dashboard/introduction')
@login_required
def introduction():
    role = session['role']
    # user_roles = current_user.roles  # Assuming 'current_user' is from Flask-Login
    return render_template('content/dashindex.html', role=role)
```

Folder Name: ./app/templates/**sales_route.py**

```
# app/saleforecast_route.py
import os
import pickle
import pandas as pd
import numpy as np
from flask import Blueprint, render_template, request, redirect, url_for, session

saleforecast_bp = Blueprint('saleforecast', __name__)

# Load the ARIMA model
model_path = os.path.join('app', 'models', 'sales_mod', 'arima_model.pkl')
with open(model_path, 'rb') as file:
    model_fit = pickle.load(file)

# Load and preprocess the dataset
data_path = os.path.join('app', 'data', 'insurance_dataset_full_2010_to_2024.csv')
```

```
data = pd.read_csv(data_path)
data['Date'] = pd.to_datetime(data[['Year', 'Month']].assign(DAY=1))
data.set_index('Date', inplace=True)
data = data[data.index >= '2000-01-01']

def predict_sales(policy_type, duration_years, min_threshold=0.5, moving_avg_window=5):
    filtered_data = data[data['Policy_Type'] == policy_type]['Sales'].resample('M').sum()
    forecast_periods = duration_years * 12
    forecast = model_fit.predict(n_periods=forecast_periods)

    historical_std = filtered_data.std()
    random_variation = np.random.normal(0, historical_std, size=forecast_periods)
    forecast += random_variation

    alpha = 0.9
    forecast_smooth = np.array([forecast[0]])
    for i in range(1, len(forecast)):
        forecast_smooth = np.append(forecast_smooth, alpha * forecast[i] + (1 - alpha) *
forecast_smooth[-1])

    forecast_min = forecast_smooth.min()
    if forecast_min < min_threshold:
        adjustment_factor = min_threshold - forecast_min
        forecast_smooth += adjustment_factor

    forecast_series = pd.Series(forecast_smooth,
index=pd.date_range(start=filtered_data.index[-1], periods=forecast_periods + 1,
freq='M')[1:])
    moving_avg = forecast_series.rolling(window=moving_avg_window).mean()

    return forecast_series, moving_avg

def generate_interpretation(forecast_series, moving_avg):
    interpretation = []
    if forecast_series.mean() > moving_avg.mean():
        interpretation.append("The forecasted sales show an upward trend compared to the
moving average, indicating potential growth in demand.")
    else:
        interpretation.append("The forecasted sales are relatively flat compared to the
moving average, suggesting stable demand.")

    if forecast_series.max() > forecast_series.mean() * 1.5:
        interpretation.append("The forecast displays significant seasonal peaks, which may
indicate high demand periods for targeted marketing.")
    else:
        interpretation.append("There are no pronounced seasonal peaks, suggesting consistent
demand throughout the year.")

    if forecast_series.std() > moving_avg.std() * 1.5:
        interpretation.append("The forecast data shows high volatility, indicating possible
uncertainty in future sales.")
    else:
        interpretation.append("The forecast data shows lower volatility, suggesting more
predictable future sales.")

    return " ".join(interpretation)

@saleforecast_bp.route('/salesforecast', methods=['GET', 'POST'])
def salesforecast():

    # Get the role from the session
    user_role = session.get('role')
    if not user_role:
        return redirect(url_for('login'))  # Redirect to login if no role is found

    if request.method == 'POST':
        policy_type = request.form['policy_type']
        duration_years = int(request.form['duration_years'])

        forecast_series, moving_avg = predict_sales(policy_type, duration_years)

        forecast_data = {
            'dates': forecast_series.index.strftime('%Y-%m').tolist(),
            'values': forecast_series.tolist(),
```

```
            'moving_avg_dates': moving_avg.index.strftime('%Y-%m').tolist(),
            'moving_avg_values': moving_avg.tolist()
        }

        interpretation = generate_interpretation(forecast_series, moving_avg)

        return render_template('sales-mod/result.html', forecast_data=forecast_data,
policy_type=policy_type, duration_years=duration_years, interpretation=interpretation,
role=user_role)

    return render_template('sales-mod/index.html', role=user_role)
```

Folder Name: ./**instance**

```
from dotenv import load_dotenv
import os

load_dotenv()

class Config:
    MONGO_URI = os.environ.get('MONGO_URI') or 'mongodb://localhost:27017/mydatabase'
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'a_very_secure_key'
    SESSION_TYPE = 'filesystem'  # Or 'redis' if using a Redis session store
    SESSION_PERMANENT = True
    PERMANENT_SESSION_LIFETIME = 3600  # 1 hour

# You can also define different configurations for different environments (optional)
class DevelopmentConfig(Config):
    DEBUG = True

class ProductionConfig(Config):
    DEBUG = False
    SESSION_TYPE = 'redis'  # Example for using Redis in production

class TestingConfig(Config):
    TESTING = True
```

File Name: **.env**

```
MONGO_URI="mongodb://xyz-789/login_system"
DB_NAME="login_system"
EMAIL_USER="2033XXXmdcs@cit.edu.in"
EMAIL_PASS="asdfghjkllkj"
SECRET_KEY=""
```

Folder Name: **inapp.py**

```
from app import create_app

# Create the Flask application
app = create_app()

if __name__ == "__main__":
    # Run the app
    port = 5001  # You can change this to any port you prefer
    print(f"Application running on port {port}")
    app.run(debug=True, port=port)
```

13 CONCLUSION:

The Decision Support System (DSS) developed as part of this project is an invaluable tool for insurance companies, offering data-driven insights to resolve significant operational challenges. Its comprehensive feature set optimizes policy pricing, detects potential fraud, forecasts premium trends, assesses risks, and examines sales patterns. The integration of predictive algorithms such as Random Forest, Logistic Regression, and ARIMA allows insurers to make informed decisions, improving profitability and customer satisfaction.

By forecasting premium rates and sales trends, insurers can anticipate market demands, optimize resources, and adjust pricing to better meet customer needs. Risk assessment tools further enhance the ability to accurately evaluate policyholder risks, leading to better management of high-risk clients and strategies that minimize losses. Additionally, fraud detection features protect against fraudulent claims, reducing financial exposure.

In summary, this DSS not only streamlines insurance operations but also strengthens customer relations by offering tailored policy recommendations and competitive pricing. Its capacity to generate actionable insights, speed up decision-making processes, and adapt to changing market conditions provides insurance companies with a competitive advantage in the evolving industry.

14 **LIMITATION**:

12.1 Dependence on Data Quality: The performance and accuracy of the system are greatly affected by the quality, completeness, and availability of historical data. Any gaps or biases in this data can negatively impact the predictive capabilities of the models.

12.2 Challenges with Generalization: While the models perform well with current data, adapting to unforeseen market changes or customer behavior shifts could be problematic, affecting the system's long-term effectiveness.

12.3 Scalability Concerns: Handling large datasets or increased demand may result in performance slowdowns, requiring additional infrastructure or system enhancements to maintain efficiency.

12.4 Complexity for Non-Technical Users: For users without technical expertise, interpreting the model outputs could be challenging. This may necessitate further training or the development of simpler, user-friendly interfaces to promote broader usage.

12.5 Market-Specific Application: Currently, the system is designed for the Indian insurance market, and its scalability to international markets has not been tested.

15 FUTURE ENHANCEMENTS:

13.1 Enhancing Customer Engagement through Website Analytics: By integrating advanced website analytics, insurers can better understand customer behaviors and interactions, enabling them to offer more personalized products and improve overall user satisfaction.

13.2 Improved Response Time for Insurance GPT Bot: Enhancing the underlying algorithms and infrastructure of the Insurance GPT bot can reduce response times, improving customer satisfaction by delivering quicker, more accurate responses.

13.3 Broader Application Across Insurance Sectors: Expanding the system to include more insurance products—such as health, life, property, and auto insurance—will increase its versatility and applicability across various sectors of the insurance industry.

13.4 Refining Premium Predictions: Improving the accuracy of premium predictions by incorporating more advanced machine learning algorithms and integrating customer behavior and market trend data will lead to better pricing models.

13.5 Real-Time Data Processing: Incorporating real-time data processing capabilities will enable insurance companies to react faster to market trends, customer behaviors, and live claims data, providing more timely and accurate insights.

16 REFERENCES:

1. Ackman, R.C., Green, P.A.G., and Young, A.G. 1982. Estimating Claims Outstanding. General Insurance Monograph. Institute of Actuaries. https://actuaries.org/ASTIN/Colloquia/Bergen/Neuhaus.pdf
2. Actuarial Standards Board. 2011. Actuarial Standard of Practice No. 20: Discounting of Property/- Casualty Unpaid Claim Estimates. Available at: ASOP No. 20 - Discounting of Property/Casualty Unpaid Claim Estimates (Proposed Revision) - Actuarial Standards BoardActuarial Standards Board
3. AISAM-ACME. 2007. AISAM-ACME study on non-life long-tail liabilities. Reserve Risk and Risk Margin Assessment Under Solvency II. 17 October 2007. https://www.sciencedirect.com/science/article/abs/pii/S0167668709000638
4. Alai, D.H., Merz, M., and Wüthrich, M.V. 2009. The mean-square error of prediction in the Bornhuetter– Ferguson claims to reserve method. Annals of Actuarial Science, 4(1), 7–31. https://scholar.google.com/scholar_lookup?title=Mean-square+error+of+prediction+in+the+Bornhuetter%E2%80%93+Ferguson+claims+reserving+method&author=Alai+D.H.&author=Merz+M.&author=W%C3%BCthrich+M.V.&publication+year=2009
5. Alai, D.H., Merz, M., and Wüthrich, M.V. 2011. A mean-square error of prediction in the Bornhuetter– Ferguson claims reserving method: revisited. Annals of Actuarial Science, 5(01), 7–17. https://scholar.google.com/scholar_lookup?title=Mean-square+error+of+prediction+in+the+Bornhuetter%E2%80%93+Ferguson+claims+reserving+method%3A+revisited&author=Alai+D.H.&author=Merz+M.&author=W%C3%BCthrich+M.V.&publication+year=2011
6. A.M., Best. 2015. Best's Statement File: United Kingdom. Available at: http://www.ambest.com/sales/ statementuk
7. A.M., Best. 2016b. Best's Special Report: Asbestos and Environmental Liabilities. November 2016. https://news.ambest.com/newscontent.aspx?refnum=254978#:~:text=A%20new%20Best%E2%80%99s%20Market%20Segment%20Report

8. Antonio, K., and Beirlant, J. 2007. Actuarial statistics with generalized linear mixed models. Insurance: Mathematics and Economics, 40(1), 58–76. https://scholar.google.com/scholar_lookup?title=Actuarial+statistics+with+generalized+linear+mixed+models&author=Antonio+K.&author=Beirlant+J.&publication+year=2007
9. Antonio, K., and Beirlant, J. 2008. Issues in claims reserving and credibility: a semiparametric approach with mixed models. Journal of Risk and Insurance, 75(3), 643–676. https://scholar.google.com/scholar_lookup?title=Issues+in+claims+reserving+and+credibility%3A+a+semiparametric+approach+with+mixed+models&author=Antonio+K.&author=Beirlant+J.&publication+year=2008