# Customer Travel ML Churn Project

Ethan Ong Yi Shen(s4407636)

December 13, 2025

# Contents

# 1 Executive Summary

This project addresses a key challenge faced by the travel company: a 23% customer churn rate, meaning nearly one in five customers leave the service. Losing customers at this level affects revenue, raises marketing costs, and threatens long-term growth. The company currently has no predictive tool to identify customers at risk of leaving. To solve this, this project has developed a complete machine-learning pipeline to predict churn, explain the reasons behind it, and support targeted retention planning.

The analysis used the company's anonymized customer dataset, which includes demographic and behavioural factors such as Age, FrequentFlyer status, Income Class, ServicesOpted, social-media linkage, and hotel booking history. Four models were built — Logistic Regression, Random Forest, Gradient Boosting, and an Artificial Neural Network — using a consistent 80/20 stratified train–test split and a shared preprocessing pipeline. SHAP was used to explain why the model makes each prediction, giving clear visibility into churn drivers.

For the primary evaluation metric, F1-score was chosen over Recall. Using Recall alone would cause the system to flag too many customers as potential churners, even when they are not at risk. For a small travel company with a limited budget for customer retention, this would be difficult to manage. The F1-score is better because it balances how well the model finds true churners while keeping the number of false alerts under control.

Among all the models evaluated, the Random Forest model performed the best. Its F1-score was 75.5% and its Recall was 82.2%, both higher than the targets set at the start. With numbers at this level, the model can be used with a reasonable level of confidence when spotting customers who may leave.

A simple web prototype was also built to show how the model can be used. Users can run churn checks, look at SHAP explanations, and be prompted with suggested follow-up steps through the interface. This prototype can be expanded later and integrated into the company's CRM system.

# 2 Introduction

## 2.1 Problem Statement

Keeping customers in the travel sector is not easy. Travellers change companies easily due to pricing, service issues, and the number of options available. In our dataset, the churn rate is around 23%, so about one in five customers leave.

At present, the company has no predictive mechanism to identify customers at risk of leaving. As a result, retention efforts are reactive instead of proactive. The core problem this project aims to address can be summarised as follows:

- Who are the customers most likely to churn?
- What features are associated with those customers?
- What areas can the company work on to reduce churn?

By tackling these questions, the company can shift from broad marketing efforts to more focused actions based on actual customer data.

## 2.2 Project Goal and Approach

The aim of this project was to build a model that could sort customers into two groups: those likely to leave and those likely to stay. To do this, several models were trained on past customer records and then compared to see which one performed best.

The models selected were:

- Logistic Regression
- Random Forest
- Gradient Boosting
- Artificial Neural Network (ANN)

Each model followed the same training setup, using an 80/20 stratified split and the same preprocessing steps. Their performance was compared using the F1-score because the classes were uneven and both Precision and Recall were important.

To understand the final model's behaviour, SHAP was used. It showed how each feature influenced a customer's churn score and pointed out which factors were the most important. This information helped clarify how the model worked and allowed the Sales and Marketing team to see the reasons behind each customer's risk score.

A second part of the project was to create a deployable model that could later fit into the travel company's CRM system. A working prototype web application was built to demonstrate how the model could be used in practice by business users.

## 2.3 Business Value

Every technical project must ultimately translate into business value, and the churn-prediction work can deliver meaningful business value in several ways.

### Revenue Stabilisation

By identifying customers who are at risk of leaving, the company can step in early and use targeted retention measures to reduce customer loss and protect revenue.

### Strategic Insight

The combination of modelling and SHAP analysis revealed deeper behavioural patterns that were not fully visible in initial Exploratory Data Analysis (EDA). Age seemed to play only a small role at first, but the SHAP results showed otherwise. It turned out to be the second biggest driver, right after FrequentFlyer status. These insights help management refine loyalty strategies and customer engagement plans.

### Operational Efficiency

The model enables the company to focus retention efforts on high-risk customers instead of implementing expensive blanket marketing campaigns. It allows the Sales and Marketing teams to concentrate on customers who are more at risk and direct their efforts where they matter most.

The remainder of the report is structured as follows:

- Section 2: Business Understanding
- Section 3: Data Understanding
- Section 4: Data Preparation
- Section 5: Modelling
- Section 6: Evaluation
- Section 7: Deployment
- Section 8: Conclusion

# 3 Business Understanding

## 3.1 Business Objectives

The company wanted better insight into why customers were leaving and aimed to reduce its 23% churn rate. These aims formed the foundation of the project.

### Predictive Accuracy

Part of this work involved building a machine learning model that could hit at least a 75% F1-score and an 80% Recall. Setting these numbers helped balance accuracy with the need to avoid too many false positives.

### Driver Identification

Determine which customer attributes were the strongest indicators of churn. This included examining factors such as customer habits and their level of engagement to see what might be causing certain customers to churn.

### Strategic Support for Retention Planning

Provide useful data that would help the Sales and Marketing teams to design retention actions aimed at customers most at risk, instead of running broad, unfocused campaigns.

By setting these goals, the project delivered both a working predictive model and practical insights the team could act on.

## 3.2 Current Situation

The early look at the dataset indicated that the company served mostly a young group of customers within the ages of 27 to 38. The observed churn rate was 23%—equivalent to losing roughly one in every five customers.

In the travel industry, competition is intense, and online aggregators make switching between service providers extremely easy. With 23% of customers leaving, the company was facing a real loss, especially since attracting new customers tends to cost more than keeping the ones they already have. The company had no tool to help them predict

which customers were at risk, so the team relied on general retention campaigns. These campaigns were expensive and did not move the needle much. A shift toward using the available data in a more focused way became necessary.

Out of this context, two major needs emerged:

1. The ability to predict churn reliably.
2. Actionable insights to support targeted retention programs.

## 3.3 Project Plan

The CRISP–DM framework was used to manage the entire project lifecycle. The main areas of work were:

### Data Understanding

EDA was used to explore the dataset. This exercise examined how each feature was distributed and how different variables related to one another. Visual charts and tables were generated to provide insights on customers' behaviour.

### Data Preparation

Data had to be cleaned before modelling. This included checking for missing values and duplicates, and handling categorical and numeric fields. The steps had to be consistent and repeatable, with no data leakage.

### Modelling

The Logistic Regression model was used to form a baseline. More complex models were added (Random Forest, Gradient Boosting, and an ANN) to provide comparisons. For fair benchmarking, every model used the same preprocessing steps and the same stratified data split.

### Evaluation

The key metric for evaluation was the F1-score. Other metrics plus the confusion matrices were generated. The model with the highest F1-score was selected. SHAP was used to provide deeper insights on the churn drivers.

### Deployment

A prototype built in Flask was used to show how the model would function in practice, with plans for later integration into the company's CRM system. This approach kept the work aligned with the company's needs and the analytical objectives.

# 4 Data Understanding

## 4.1 Data Collection and Description

The Customer Travel Dataset contained the following information:

- Age
- FrequentFlyer Status
- AnnualIncome Class
- ServicesOpted
- AccountSyncedToSocialMedia

The target variable was:

- 1 (Churned)
- 0 (Stay)

The data indicated that customers were young (between the ages 27–38). The values in these columns were generalised and grouped into ranges—most likely anonymised. Because of this, there were 507 duplicates found. Removing duplicates was not necessary in data preparation because these were not considered bad data.

## 4.2 Exploratory Data Analysis (EDA)

EDA was used to better understand the data.

### Target Summary

| Churn | Count | Percentage | Churn_Label |
|---|---|---|---|
| 0 | 730 | 76.52 | No Churn |
| 1 | 224 | 23.48 | Churned |

Figure 1: Table showing the Overall Churn Rate

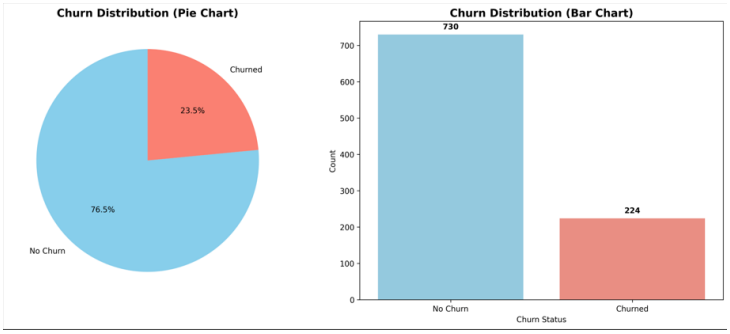



Figure 2: Chart showing the overall churn distribution

This showed that the Customer Travel Dataset was imbalanced. Most customers stayed and about 1 in 5 left. An imbalance like this would mean that metrics like Recall and F1-score would be more meaningful than accuracy.

Examining the numerical columns (Age and ServicesOpted)

### Age Group By Churn Rate

| AgeGroup | Count | Churn_Rate | Churned_Count | No_Churn_Count |
|---|---|---|---|---|
| 20-29 | 439 | 0.278 | 122 | 317 |
| 30-39 | 515 | 0.198 | 102 | 413 |
| 40-49 | 0 | | 0 | 0 |
| 50-59 | 0 | | 0 | 0 |

Figure 3: Table showing the Age Group By Churn Rates

| Target | Count | Mean | Median | Mode | Std | Min | Max | Q1 | Q3 |
|---|---|---|---|---|---|---|---|---|---|
| No Churn | 730 | 32.35 | 31 | 30 | 3.21 | 27 | 38 | 30 | 35 |
| Churned | 224 | 31.32 | 30 | 28 | 3.62 | 27 | 38 | 28 | 35 |

Figure 4: Table showing Age Statistics by Churn Rates

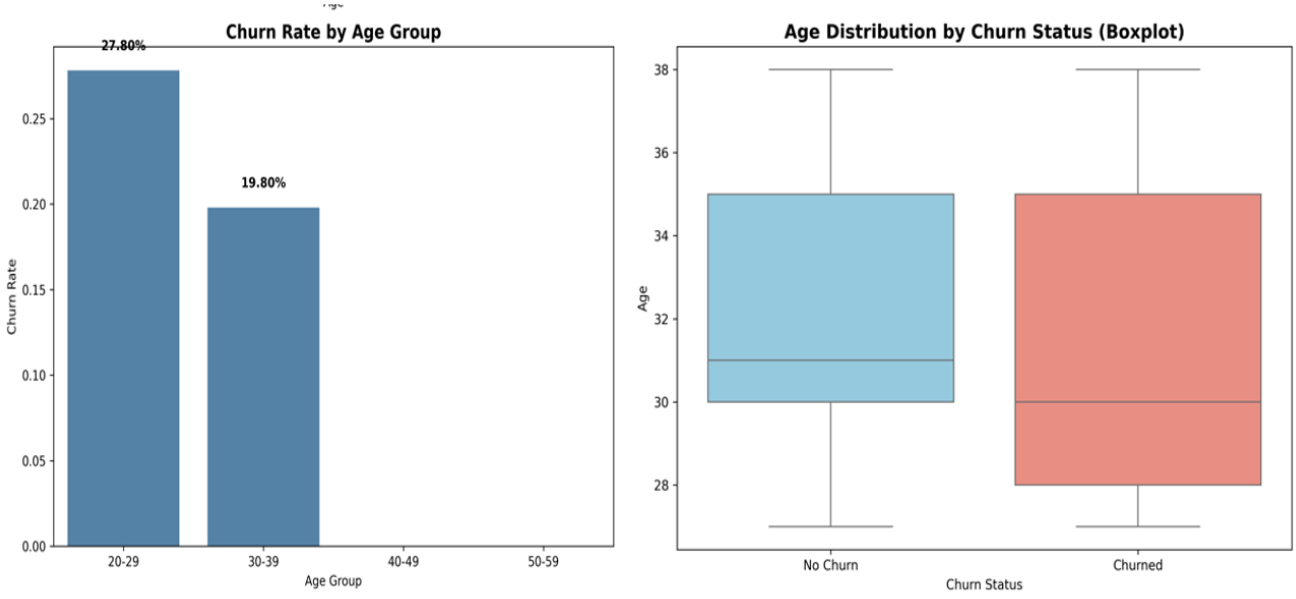



Figure 5: Histogram and Boxplot showing the Age distribution By Churn Rates

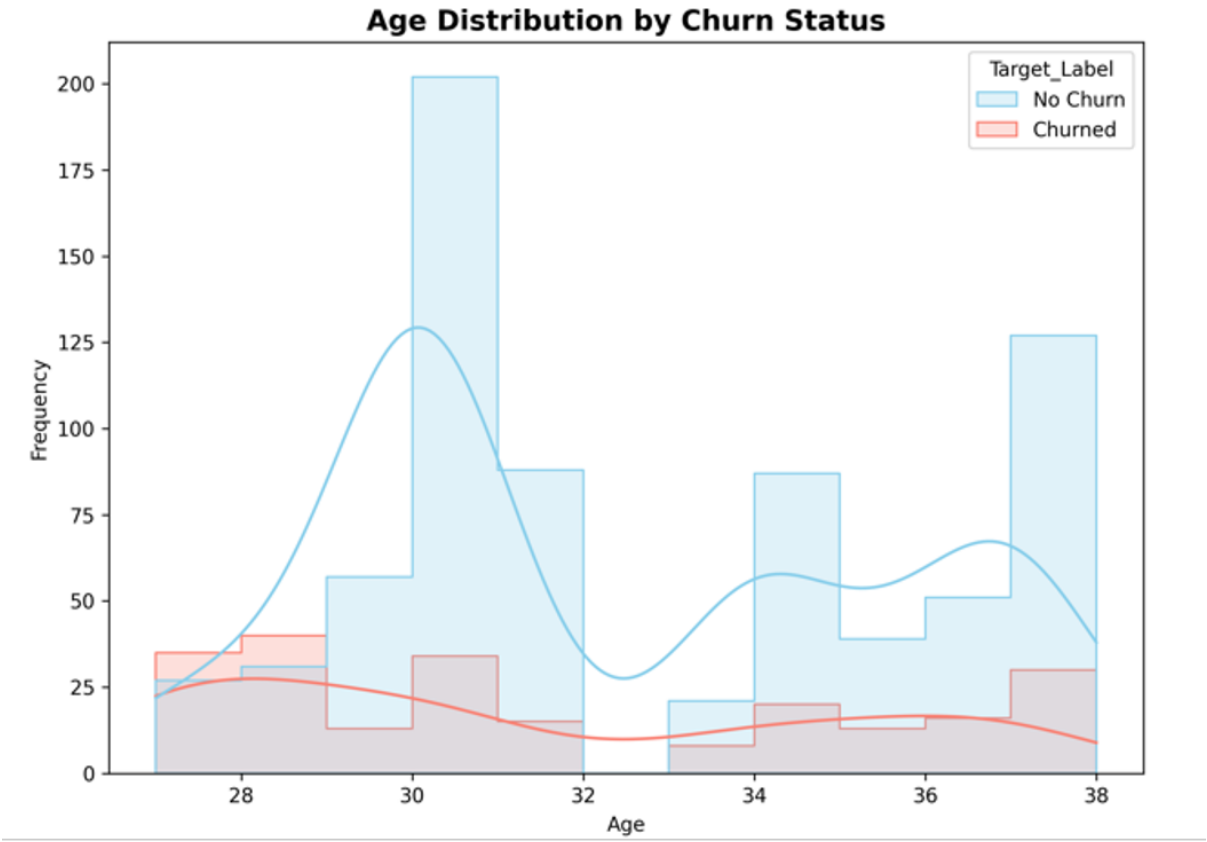



Figure 6: Chart showing the Age distribution by Churn Rates

Churn Rate by Age Group showed that the [20-29] age group (27.8%) churned more than the [30-39] age group (19.8%). There were no other groups in the Dataset so any insights on age was very limited. Overall, age did not appear to be a strong churn predictor.

**ServicesOpted by Churn Rate**

| ServicesOpted | Count | Churned | Churn_Rate | Churn_Rate_Pct |
|---|---|---|---|---|
| 1 | 404 | 93 | 0.23 | 23 |
| 2 | 176 | 52 | 0.295 | 29.5 |
| 3 | 124 | 10 | 0.081 | 8.1 |
| 4 | 117 | 22 | 0.188 | 18.8 |
| 5 | 69 | 26 | 0.377 | 37.7 |
| 6 | 64 | 21 | 0.328 | 32.8 |

Figure 7: Table showing ServicesOpted by Churn Rates

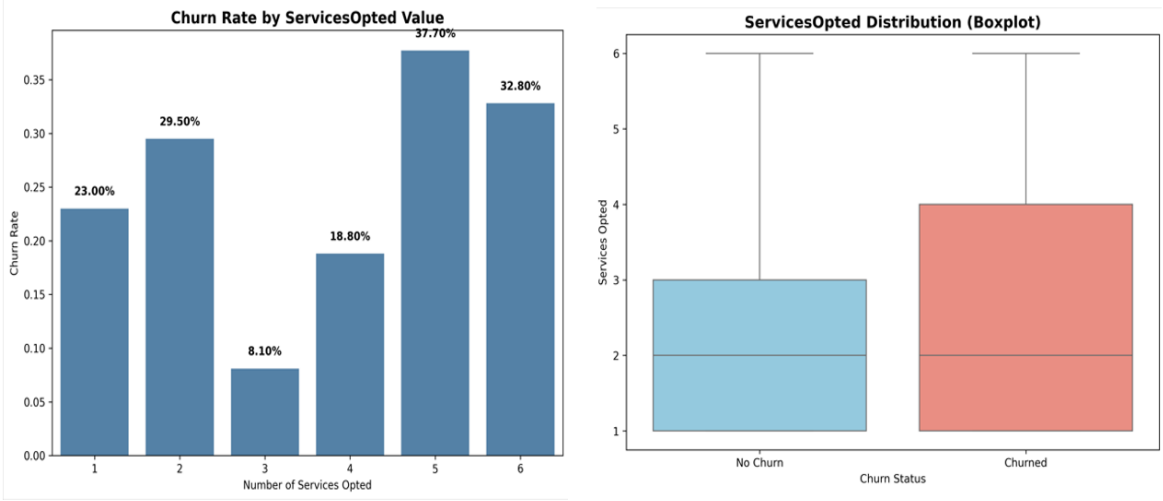



Figure 8: Histogram and Boxplot showing the ServicesOpted distribution By Churn Rates

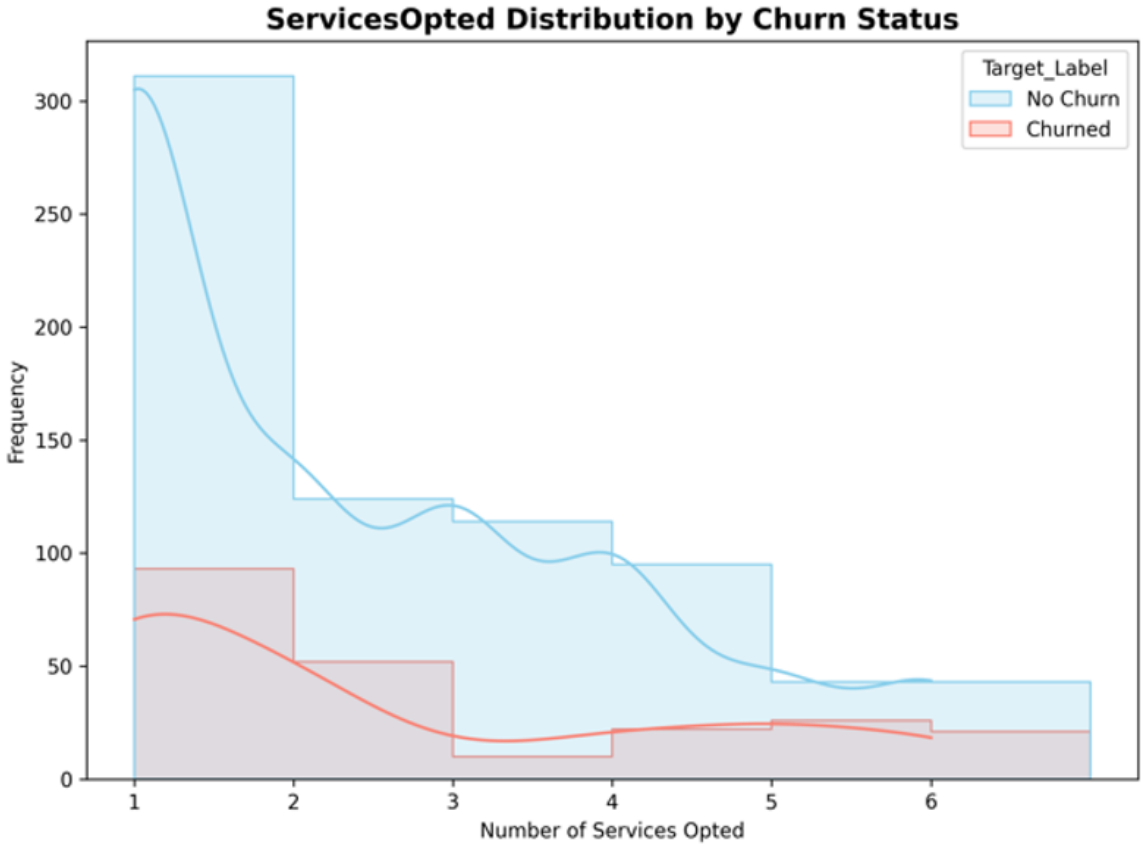



Figure 9: Chart showing the ServicesOpted distribution by Churn Rates

This Churn distribution was rather unusual. Churn rates were high at the 1, 2 Services Options and at 5, 6 Services Options. Its lowest was at 3 Services Options forming a U-shaped distribution curve.

A possible interpretation could be:

- Low service levels (1–2 services): engagement with the company was limited, making it easy to switch to other travel providers.
- High service levels (5–6 services): customers might find the cost too high and may feel overwhelmed, leading to dissatisfaction and eventually churn.
- Moderate service levels (3 services): these customers appear to be in the middle ground and were generally satisfied with the services.

Examining Categorical Features

As per the table displayed below, churned rates were examined across several key categorical features:

| Categorical Features | Category | Count | Churned | Not_Churned | Churn_Rate | Churn_Rate_Pct |
|---|---|---|---|---|---|---|
| FrequentFlyer | No | 608 | 69 | 539 | 0.113 | 11.3 |
| FrequentFlyer | No Record | 60 | 8 | 52 | 0.133 | 13.3 |
| FrequentFlyer | Yes | 286 | 147 | 139 | 0.514 | 51.4 |
| AnnualIncomeClass | High Income | 159 | 92 | 67 | 0.579 | 57.9 |
| AnnualIncomeClass | Low Income | 386 | 104 | 282 | 0.269 | 26.9 |
| AnnualIncomeClass | Middle Income | 409 | 28 | 381 | 0.068 | 6.8 |
| AccountSyncedToSocialMedia | No | 594 | 125 | 469 | 0.21 | 21 |
| AccountSyncedToSocialMedia | Yes | 360 | 99 | 261 | 0.275 | 27.5 |
| BookedHotelOrNot | No | 576 | 176 | 400 | 0.306 | 30.6 |
| BookedHotelOrNot | Yes | 378 | 48 | 330 | 0.127 | 12.7 |

Figure 10: Table showing the Categorical Features by Churn Rates

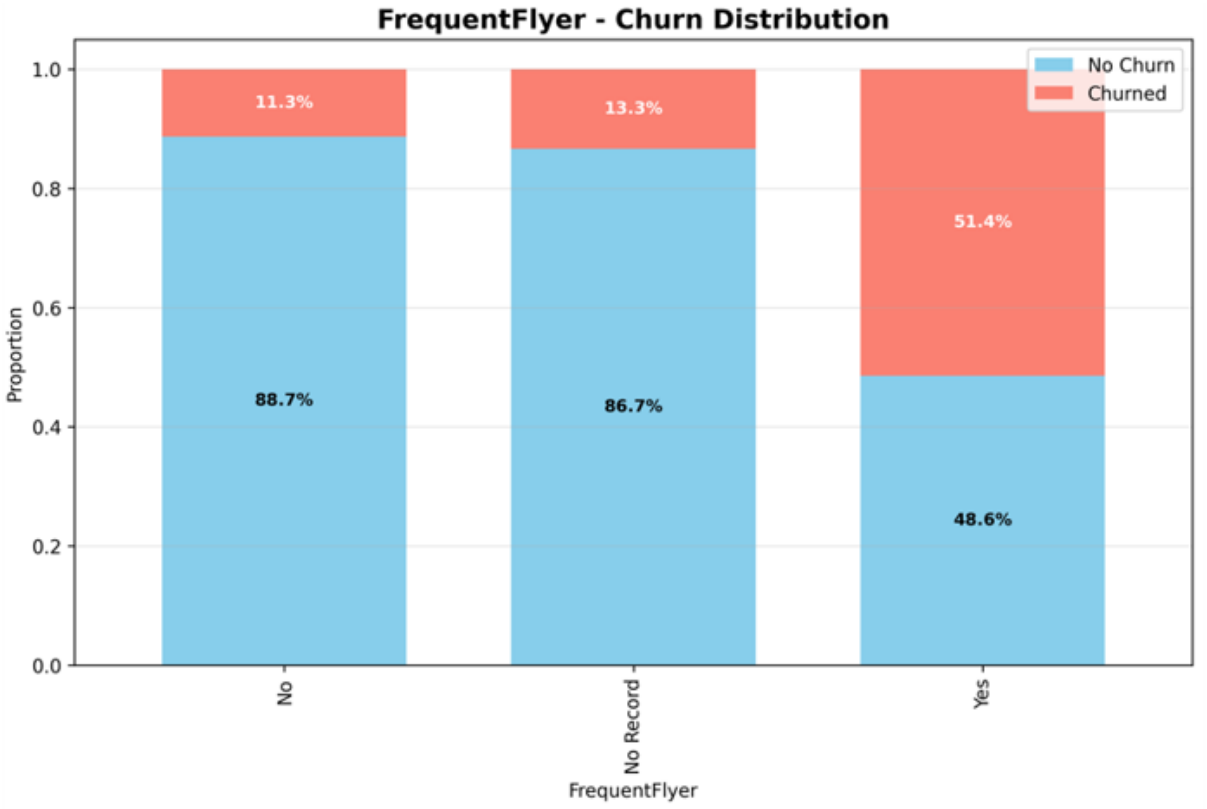



Figure 11: Histogram showing the FrequentFlyer distribution by Churn Rates

For FrequentFlyer:

- FrequentFlyer (Yes) – 51.4% churn
- FrequentFlyer (No) – 11.3% churn
- FrequentFlyer (No Record) – 13.3% churn

This showed that FrequentFlyer (Yes) customers churned much more than others, suggesting that frequent travellers had higher expectations and were more likely to leave if those expectations were not met. This made FrequentFlyer a very strong churn driver.

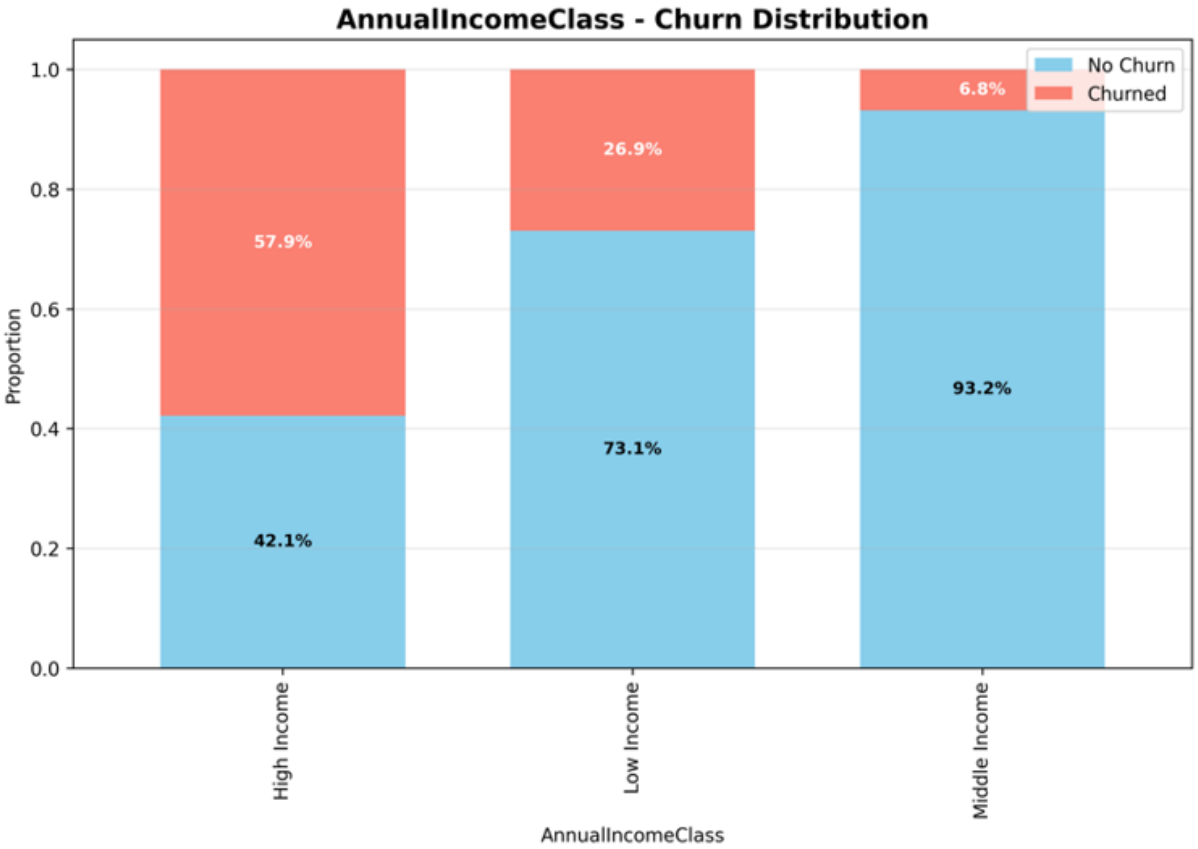



Figure 12: Histogram showing the AnnualIncomeClass distribution by Churn Rates

For AnnualIncomeClass:

- High Income – 57.9% churn
- Low Income – 26.9% churn
- Middle Income – 6.8% churn

High income customers appeared to churn the most, likely due to expectations of premium service. Low income customers might churn due to price sensitivity. Middle income customers showed the lowest churn and seemed to be a more stable, loyal segment. Overall, AnnualIncomeClass emerged as a major churn predictor.

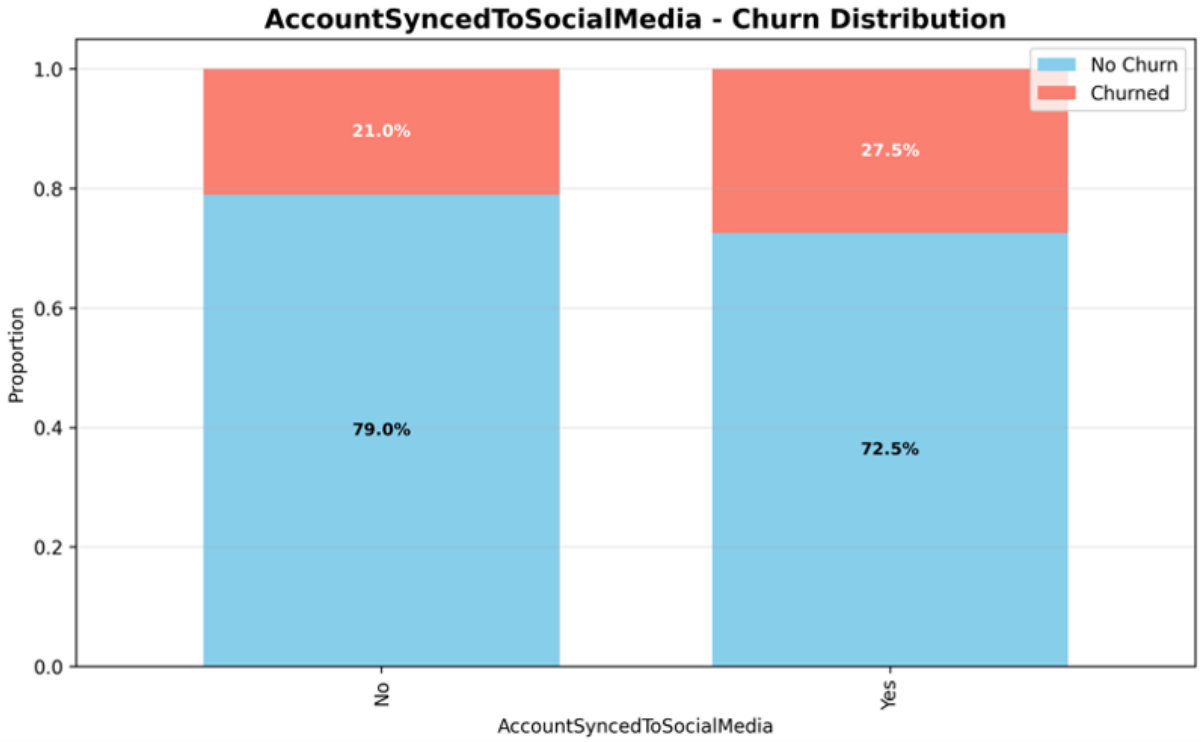



Figure 13: Histogram showing the AccountSyncedToSocialMedia distribution by Churn Rates

For AccountSyncedToSocialMedia:

- Yes – 27.5% churn
- No – 21% churn

Customers with social accounts synced churned slightly more, but the difference was small, suggesting a weak effect.

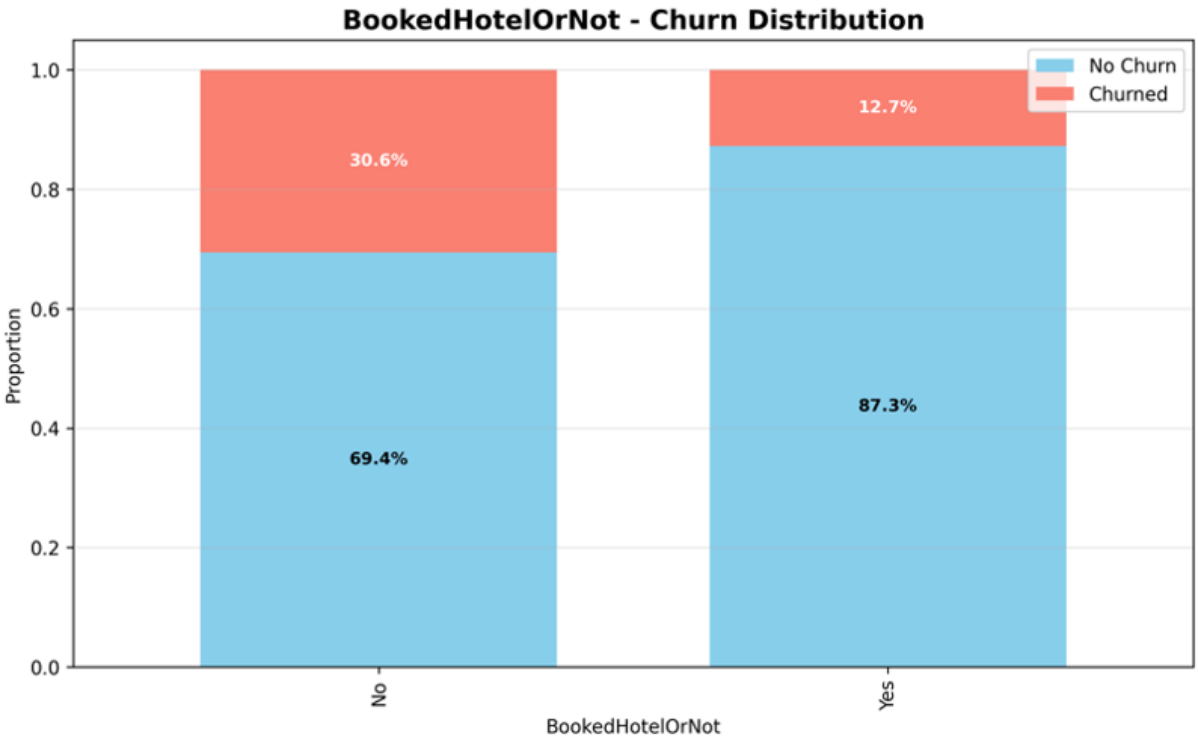



Figure 14: Histogram showing the BookedHotelOrNot distribution by Churn Rates

For BookedHotelOrNot:

- No – 30.6% churn
- Yes – 12.7% churn

Customers who booked hotels through the company were clearly more loyal. Hotel booking indicated the need for deeper engagement to lower churn, and this appeared to be a strong negative churn indicator.

Investigation into which attributes were related using the correlation matrix and heat map.

| | Age | ServicesOpted | Churn | FrequentFlyer | SocialMediaSync | BookedHotel | IncomeClass |
|---|---|---|---|---|---|---|---|
| **Age** | 1 | -0.012421838 | -0.131533532 | 0.049261332 | -0.01636654 | 0.024281138 | -0.016318538 |
| **ServicesOpted** | -0.012421838 | 1 | 0.038645645 | -0.173881125 | -0.148655288 | 0.1558858 | 0.090307478 |
| **Churn** | -0.131533532 | 0.038645645 | 1 | 0.430973012 | 0.073830865 | -0.206055246 | 0.142364555 |
| **FrequentFlyer** | 0.049261332 | -0.173881125 | 0.430973012 | 1 | -0.09404175 | -0.193280585 | 0.319077204 |
| **SocialMediaSync** | -0.01636654 | -0.148655288 | 0.073830865 | -0.09404175 | 1 | -0.100104163 | -0.227100681 |
| **BookedHotel** | 0.024281138 | 0.1558858 | -0.206055246 | -0.193280585 | -0.100104163 | 1 | -0.024067686 |
| **IncomeClass** | -0.016318538 | 0.090307478 | 0.142364555 | 0.319077204 | -0.227100681 | -0.024067686 | 1 |

Figure 15: Correlation Matrix

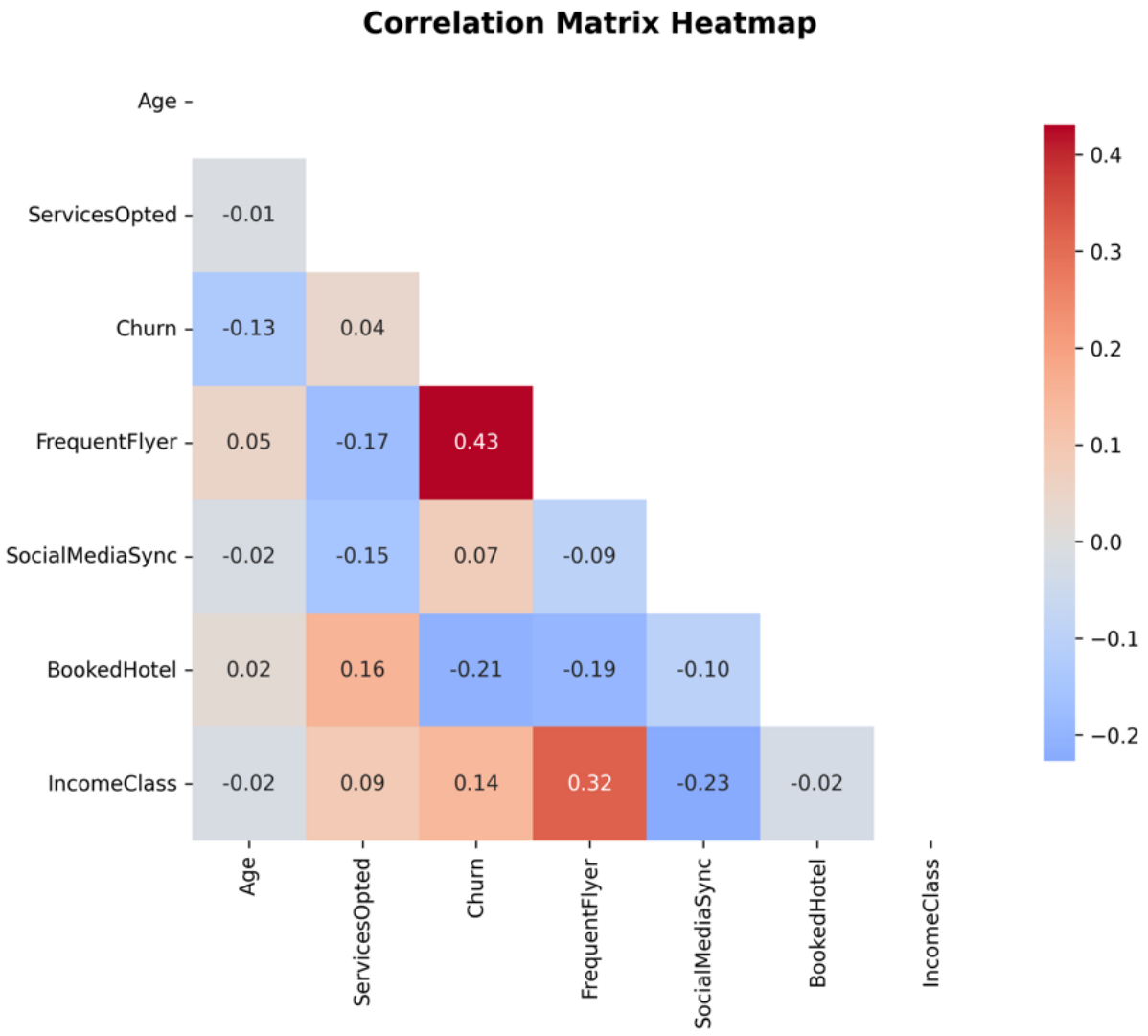



Figure 16: Correlation Heat Map

The correlation matrix and heatmap visual indicated the following noticeable relationships that were worth investigating:

- FrequentFlyer + AnnualIncomeClass (32%)
- ServicesOpted + BookedHotel (16%)

*Note: The other relationship, FrequentFlyer + Churn (43%), was already discussed earlier.*

Drilling into the pair-ed relationships

| IncomeClass | FrequentFlyer | Churn_Rate | Count |
|---|---|---|---|
| High Income | Yes | 0.578616352 | 159 |
| Low Income | No | 0.189873418 | 237 |
| Low Income | No Record | 0.181818182 | 22 |
| Low Income | Yes | 0.433070866 | 127 |
| Middle Income | No | 0.064690027 | 371 |
| Middle Income | No Record | 0.105263158 | 38 |

Figure 17: Table showing AnnualIncomeClass and FrequentFlyer Distribution Churn Rates

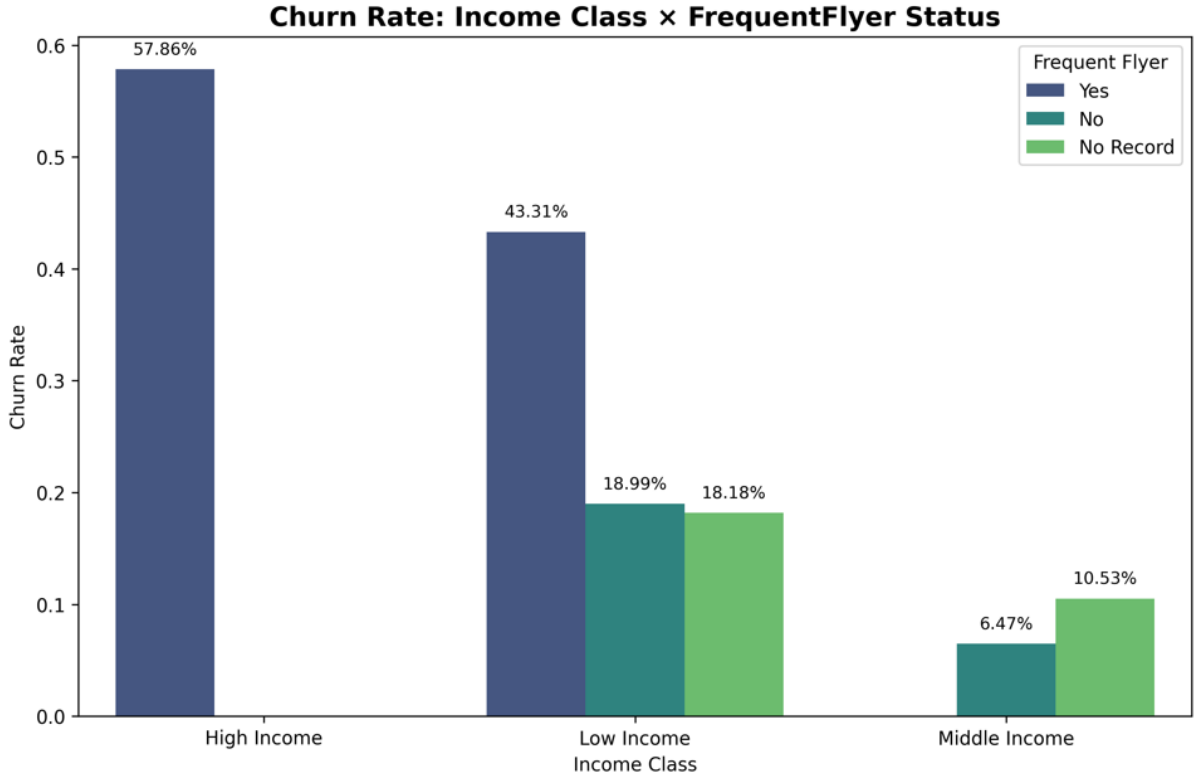



Figure 18: Histogram showing AnnualIncomeClass and FrequentFlyer Distribution Churn Rates

Both the table and the chart combining FrequentFlyer and AnnualIncomeClass revealed a sharper view of churn risk.

The highest churn rates were from:

- FrequentFlyer – Yes + AnnualIncomeClass – High (57.86%)
- FrequentFlyer – Yes + AnnualIncomeClass – Low (43.3%)

The lowest churn rates were from:

- AnnualIncomeClass – Middle + FrequentFlyer – No Record (10.5%)
- AnnualIncomeClass – Middle + FrequentFlyer – No (6.5%)

This interaction showed that High Income + FrequentFlyer (Yes) formed a high-risk churn profile, while Middle Income + FrequentFlyer (No/No Record) were relatively stable. This combination of income and travel frequency provided a powerful way to segment customers for targeted retention.

| ServicesGroup | BookedHotel | Churn_Rate | Count |
|---|---|---|---|
| Low (1-2) | No | 0.345646438 | 379 |
| Low (1-2) | Yes | 0.069651741 | 201 |
| Medium (3-4) | No | 0.157894737 | 152 |
| Medium (3-4) | Yes | 0.08988764 | 89 |
| High (5-6) | No | 0.466666667 | 45 |
| High (5-6) | Yes | 0.295454545 | 88 |

Figure 19: Table showing ServicesOpted and BookedHotelOrNot Distribution Churn Rates

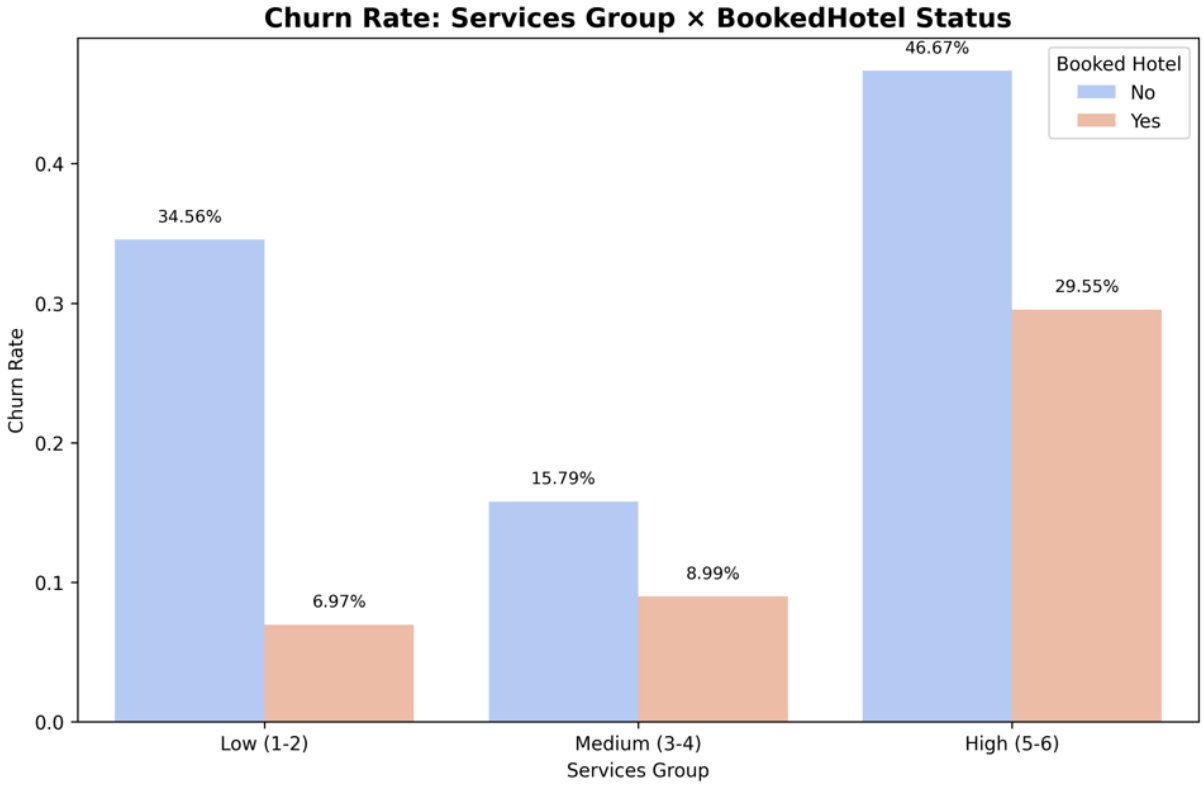



Figure 20: Histogram showing ServicesOpted and BookedHotelOrNot Distribution Churn Rates

The interaction between ServicesOpted and BookedHotelOrNot showed a consistent pattern. For every service level, customers who booked hotels churned less.

Examples:

- ServicesOpted – Low + BookedOrNot – No (34.56%)
- ServicesOpted – Low + BookedOrNot – Yes (6.96%)
- ServicesOpted – Medium + BookedOrNot – No (15.79%)
- ServicesOpted – Medium + BookedOrNot – Yes (8.99%)
- ServicesOpted – High + BookedOrNot – No (46.67%)
- ServicesOpted – High + BookedOrNot – Yes (29.5%)

Although churn rose at very high service levels, hotel booking consistently reduced churn rates within each ServiceOpted level. This confirmed BookedHotelOrNot as a strong protective factor and a key factor for retention.

# Summary

The EDA led to several important findings:

- F1-score and Recall were more appropriate evaluation metrics due to the imbalanced CustomerTravel dataset.

- The strongest churn predictors were FrequentFlyer–Yes and AnnualIncomeClass–High.
- BookedHotelOrNot appeared to be a strong retention factor.
- Age and AccountSyncedToSocialMedia contributed little to the churn probability.

This provided a good starting point for the data preparation and modelling stages. These findings could later be compared with the SHAP analysis to verify their correctness and uncover additional patterns not visible through EDA alone.

### 4.3 Data Quality Verification

A few data checks were done upfront to confirm the dataset was suitable for modelling. No missing values were found, so nothing is needed to be fixed there. As mentioned earlier in the EDA section, there was no need to check for duplicate records as the data had been anonymised leading to multiple similar records.

The Age distribution didn't show anything unusual either; most customers fell within a young age range. While this reflects the company's current audience, it also suggests that older travellers—who could be part of future growth—are not well represented in the data.

Rather than applying heavy cleaning, the aim was to keep the data largely intact while making sure the later steps—preprocessing, splitting, and modelling—were handled carefully to avoid leakage and keep everything consistent.

## 5 Data Preparation

The data preparation stage entailed data selection, data cleaning, data transformation, and data formatting to get it ready for the modelling stage.

### 5.1 Data Selection

For this project scope, all features in the CustomerTravel dataset were retained. The intention was to allow the model to learn from the full set of customer attributes before deciding whether any features should be removed. Since this was the first iteration of the project, feature reduction was deferred until there was a clearer understanding of how each variable contributed to model performance. Any feature elimination or dimensionality reduction would therefore be carried out only in future iterations, after analysing model behaviour, feature importance, and SHAP interpretations.

### 5.2 Data Cleaning

The dataset had already been checked for integrity during the Data Understanding stage. No missing values were found, and duplicate rows were preserved because the dataset had been anonymised. Although the dataset itself did not require imputation or duplicate handling, checks were necessary to ensure future operational robustness—for example,

validating that new incoming customer records would contain all required values and that the model would not fail due to missing fields.

## 5.3 Data Transformation / Feature Engineering

The dataset contained only two numerical variables, with the remainder being categorical. Since most machine learning algorithms require numerical inputs, both groups of variables needed transformation.

### Numerical Features

The numerical attributes were standardised so that they operated on comparable scales. Standardisation prevents variables with larger magnitudes from overpowering the learning process and generally improves model stability and convergence.

### Categorical Features

The categorical attributes had to be transformed into separate binary columns (0 or 1) through a process known as one-hot encoding because machine learning models cannot work with text labels. To avoid multicollinearity, the `drop_first` option was used.

A critical requirement was that all transformations be applied after the `train_test_split`. Applying scaling before splitting would leak information from the test set into the training process and compromise the model's validity. For this reason, the dataset was split first into training and testing sets before any transformations were applied.

All preprocessing operations were implemented using scikit-learn's `Pipeline` and `ColumnTransformer`. This automated the scaling and encoding steps, ensured that the exact transformation logic was reused throughout modelling, SHAP analysis, and deployment, and made the workflow reproducible.

Because one-hot encoding changes the number and ordering of columns, the full ordered list of transformed feature names was captured and stored. This was essential for interpretation tasks such as SHAP, which requires an accurate mapping back to the original attributes.

## 5.4 Data Formatting

After transformations, the outputs were assembled into a structure that the models could reliably consume.

### Consistent Data Types

Transformed columns were checked so that numerical outputs used floating-point types and encoded columns used efficient integer types. Keeping data types consistent helps prevent errors later on.

### Stable Column Ordering

One-hot encoding introduces multiple columns per feature. These columns must appear in a consistent order every time the data is transformed so that:

- - the trained model receives inputs in the same order it was fitted on.

- - SHAP can attach its values to the correct features.
- - the deployment environment can validate that incoming data is structured correctly.

**Unified Feature Matrix**

All pre-processed numerical and categorical outputs were merged into a single matrix. This final table served as the direct input to the models and removed any ambiguity about shape or alignment.

**Saving Preprocessing Artifacts with joblib / h5**

To ensure that the exact same transformations could be applied during prediction, SHAP analysis, or deployment, key Python objects were serialised using joblib / h5. These included:

- The fitted preprocessing pipeline (scaler + encoder)
- The ordered list of transformed feature names
- Additional metadata related to input schema

Saving these structures ensured complete consistency between training, modelling and deployment. No data integration was required, as the dataset came from a single prepared source.

# 6 Modelling

This stage was focused on choosing suitable algorithms and training them on the pre-processed data to determine which model best met the project's goals.

## 6.1 Model Techniques Selection

Four models were selected to cover a range of interpretability and predictive strength.

- **Logistic Regression** served as a clear baseline and showed how each feature affected churn. Its transparency was valuable for business explanation and for comparing more complex models against a known reference.
- **Random Forest** was chosen because it can deal with complex patterns. It uses the averaging of multiple trees to avoid overfitting. It also has a built-in feature for scoring feature importance which is useful in figuring out the main churn drivers.
- **Gradient Boosting** was selected for its strong performance on structured datasets. Because it trains in small updates rather than large jumps, it is able to pick up details that basic models might overlook.
- **ANN** was included to check if a deep learning model could capture additional details missed by the other classifiers. Its layered structure allows it to learn complex, non-linear interactions in the data.

Together, these four models provided a balanced evaluation of interpretability, capability and predictive strength.

## 6.2 Test Design

The test design ensured the pipeline was correct, stable, and reproducible.

A fixed random state was applied throughout—train/test split and model initialization, and all randomised operations. This guaranteed that repeated runs of the pipeline would produce the same transformed features, model parameters, and evaluation metrics. A mismatch between runs would normally mean that one of the random settings hadn't been fixed properly.

The dataset was divided using an 80/20 split that maintained the churn distribution. This was important given the imbalance in the target. A check was done to ensure that scaling was after the split to ensure no 'data leakage'.

Preprocessing artifacts (`preprocessor.joblib`, `metadata.json`) were verified to load correctly and to produce the same feature transformations in a new session. This ensured consistency between training, Shap analysis and deployment.

The model-training script was tested to confirm that it:

- Loaded preprocessing metadata.
- Trained all four models correctly.
- Saved `model.joblib` / `model.h5`, metrics, and feature importance files.
- Reproduced identical predictions and metrics under the same settings.

Evaluation tests ensured that each model produced all required metrics: Accuracy, Precision, Recall, F1-score, ROC-AUC, and a confusion matrix. Each model's metrics were saved to `metrics.json`. A separate check confirmed that the feature names produced by the preprocessing pipeline matched those displayed in the Shap results, so nothing was interpreted wrongly.

The web application was then tested by comparing its outputs - both the predictions and Shap values - to those generated during training. This helped confirm that the preprocessing and model loading were working as expected.

## 6.3 Model Building

The model training stage followed a uniform workflow across all four algorithms. The script began by loading the preprocessed training and test datasets (`X_train_processed`, `X_test_processed`, `y_train_raw`, and `y_test_raw`) together with the accompanying `metadata.json`. All feature engineering and preprocessing had already been completed earlier in the pipeline. No further transformations were applied at this stage, ensuring that every component in the workflow operated on data prepared in a consistent and uniform manner.

Each model was initialised with fixed hyperparameters and a consistent random state. This step was particularly important for Random Forest, Gradient Boosting, and ANN, but also beneficial for Logistic Regression to ensure reproducibility.

Cross-validation was not included in the workflow. The aim was to keep the training process straightforward, rely on a single fitted model for SHAP analysis, and avoid additional complexity when saving and deploying the final artifacts.

Once a model completed training, four outputs were generated:

- `model.joblib` / `model.h5` – the trained model
- `metrics.json` – summary of the main evaluation metrics
- `classification_report.json` – a detailed breakdown of model performance
- `feature_importance.csv` – feature-level contributions from the model

The same procedure was applied to all four models so that they were evaluated under identical preprocessing and assessment conditions.

### 6.4 Models' Assessment

The model evaluation was conducted using the 20% hold-out test set that has been preprocessed earlier in the workflow. No additional transformations were applied during the assessment stage.

Each model produced predictions on the test set, and the script calculated the standard classification metrics: Accuracy, Precision, Recall, F1-score, ROC-AUC, and a confusion matrix. The dataset contained far fewer churners than non-churners, so the F1-score was used as the key metric. It captures the balance between identifying true churners and limiting incorrect alerts, which made it suitable for judging overall performance.

A single test set was used rather than cross-validation. This kept the evaluation procedure straightforward and ensured that the SHAP analysis could be tied to one final fitted model.

The models were assessed in a consistent sequence. Logistic Regression served as the baseline for both interpretability and performance. Random Forest was then examined to determine how a non-linear ensemble handled the data and whether it offered improvements in Recall or F1-score. Gradient Boosting, which often performs well on structured datasets, was evaluated next. The Artificial Neural Network (ANN) was reviewed last to determine whether a deeper non-linear model added further predictive value.

All evaluations relied on the same transformed feature set. If any unusual metric changes or mismatches in feature dimensions appeared, the preprocessing outputs and random-state settings were checked.

Once all results were available, the metrics were compared with the project baseline. The model that showed the strongest and most consistent F1-score—together with reasonable Precision, Recall, and ROC-AUC—was selected for deployment. Checks using SHAP values, along with repeated runs under identical settings, confirmed that the chosen model produced stable outputs and remained consistent with the rest of the pipeline.

## 7 Evaluation

The evaluation covered Logistic Regression, Random Forest, Gradient Boosting, and an Artificial Neural Network (ANN). Standard metrics were used: Accuracy, Precision, Recall, F1-score, and ROC-AUC.

## 7.1 Model Evaluation Results

| Model | Accuracy | F1-Score | ROC-AUC |
|---|---|---|---|
| Random Forest | 0.8743 | 0.7551 | 0.9590 |
| Gradient Boosting | 0.8901 | 0.7470 | 0.9625 |
| ANN | 0.8796 | 0.7294 | 0.9463 |
| Logistic Regression | 0.7958 | 0.6355 | 0.8617 |

Random Forest produced the most reliable mix of detecting churners while limiting unnecessary alerts. Its scores were:

- Accuracy: 0.874
- Precision: 0.698
- Recall: 0.822
- F1-score: 0.755
- ROC-AUC: 0.959

Its confusion matrix was:

- True Positives: 37
- True Negatives: 130
- False Positives: 16
- False Negatives: 8

These results met the project thresholds of 75% F1-score and 80% Recall.

## 7.2 Review Processes

The review focused on two areas: handling class imbalance and understanding feature importance.

### Class Imbalance

Logistic Regression and Random Forest were trained with `class_weight="balanced"` to give additional weight to churn cases. Gradient Boosting and ANN did not use weighting, and no resampling (such as SMOTE) or probability-threshold adjustments were applied. This resulted in an uneven approach to imbalance handling. A consistent strategy will be needed in the next iteration to produce a fairer baseline for comparing models.

### Feature Importance and Explainability

Initial feature importance scores from Random Forest and Gradient Boosting suggested Age and ServicesOpted as dominant predictors. Because tree-based importance can favour continuous variables, these results were interpreted cautiously. SHAP was then used to obtain a more reliable, model-agnostic explanation.

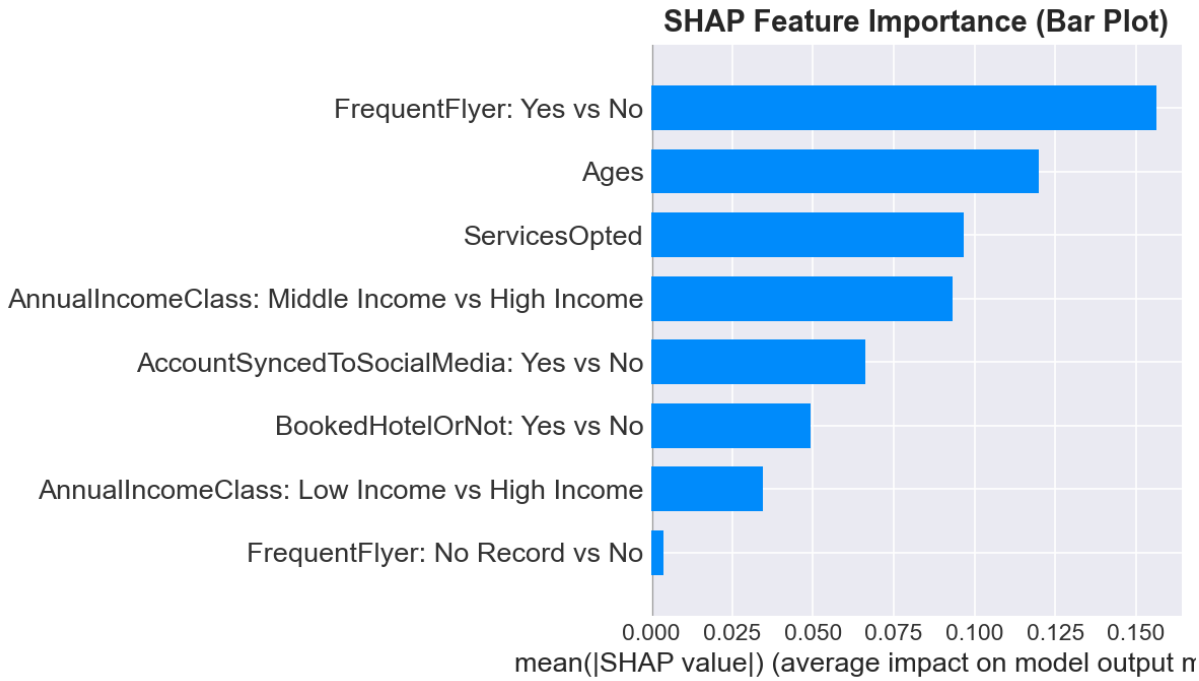



Figure 21: Bar Plot of Shap's Feature Importance

SHAP produced a clearer ranking of influence:

1. FrequentFlyer_Yes
2. Age
3. ServicesOpted
4. Income-related features
5. AccountSyncedToSocialMedia_Yes
6. BookedHotelOrNot_Yes

These results corrected the bias seen in model-derived scores and confirmed that FrequentFlyer_Yes was the strongest churn driver.

## 7.3 Next Phase for the Chosen Model

The Random Forest model is suitable for operational deployment, but several enhancements will strengthen its reliability:

- Apply class weighting across all models to ensure a fair and consistent benchmark, so performance differences reflect the models themselves rather than uneven imbalance handling.

- Explore resampling techniques such as SMOTE to further support the minority class.
- Tune the decision threshold to raise Recall for high-risk customer identification.
- Introduce more robust cross-validation to improve stability and reduce variance in model scores.
- Expand explainability through global SHAP summaries and deeper error analysis.

These steps will produce a more dependable model and offer clearer insight for business users.

## 7.4 Model Saving

Three of the models and the preprocessing pipeline were saved using joblib. Joblib is preferred over pickle in this context because it handles large arrays more efficiently. The ANN model is the only exception because Keras models cannot be saved with joblib. It is stored in h5 format. This still fits with the overall workflow, as the same preprocessing pipeline used during training is applied again during deployment.

# 8 Deployment

## 8.1 Deployment Plan

The deployment plan was to turn the final model into a working prototype that business users can interact with. Flask was chosen because it provides a lightweight and reliable way to serve predictions and explanations through a web interface.

After training, the model and preprocessing pipeline were saved using joblib / h5. These files are loaded by the Flask backend at startup so that all customer input is processed using the same steps as during model development. A dashboard was then built using HTML, CSS, and JavaScript, allowing users to enter customer information and immediately view churn predictions and insights.

## 8.2 Deployment Type and Key Considerations

### Prototype Deployment Approach

The prototype was set up as a server-based application with a web front-end. Its design was shaped by a few practical considerations, especially the need to keep the behaviour of the deployed system aligned with the training environment.

### Consistency with Training

To ensure that predictions were produced using the same processing steps as during model development, Flask was configured to load several components as soon as the server started. These included the preprocessing pipeline (`preprocessor.joblib`), the feature metadata (`metadata.json`), and a small configuration file (`best_model_info.json`) that identifies which trained model should be used. Flask then loaded the corresponding `model.joblib` / `model.h5` . Bringing these pieces together at startup kept the prototype's behaviour consistent with the training workflow.

### Input Validation

The prototype applied checks on user inputs at two points. The JavaScript in the front-end (`app.js`) first reviewed the values entered by the user to ensure they were in an acceptable format. This validated data was then sent to the backend server for another round of validation checks before the data was passed to the model. This two-step approach reduced the chance of invalid information being processed.

### SHAP Interpretability

SHAP was included to highlight the factors that influenced each prediction.For every request, the backend calculated SHAP values that showed which features increased or lowered the predicted churn level.These results were displayed in a simple format so users could understand the basis of the prediction.

### Recommendations and Business Insights

The prototype also included a small rule-based component that used the churn probability, the risk category, and the main SHAP factors to generate basic retention suggestions. This was to help guide the users respond with relevant retention incentives to customers identfied as high risk ones.

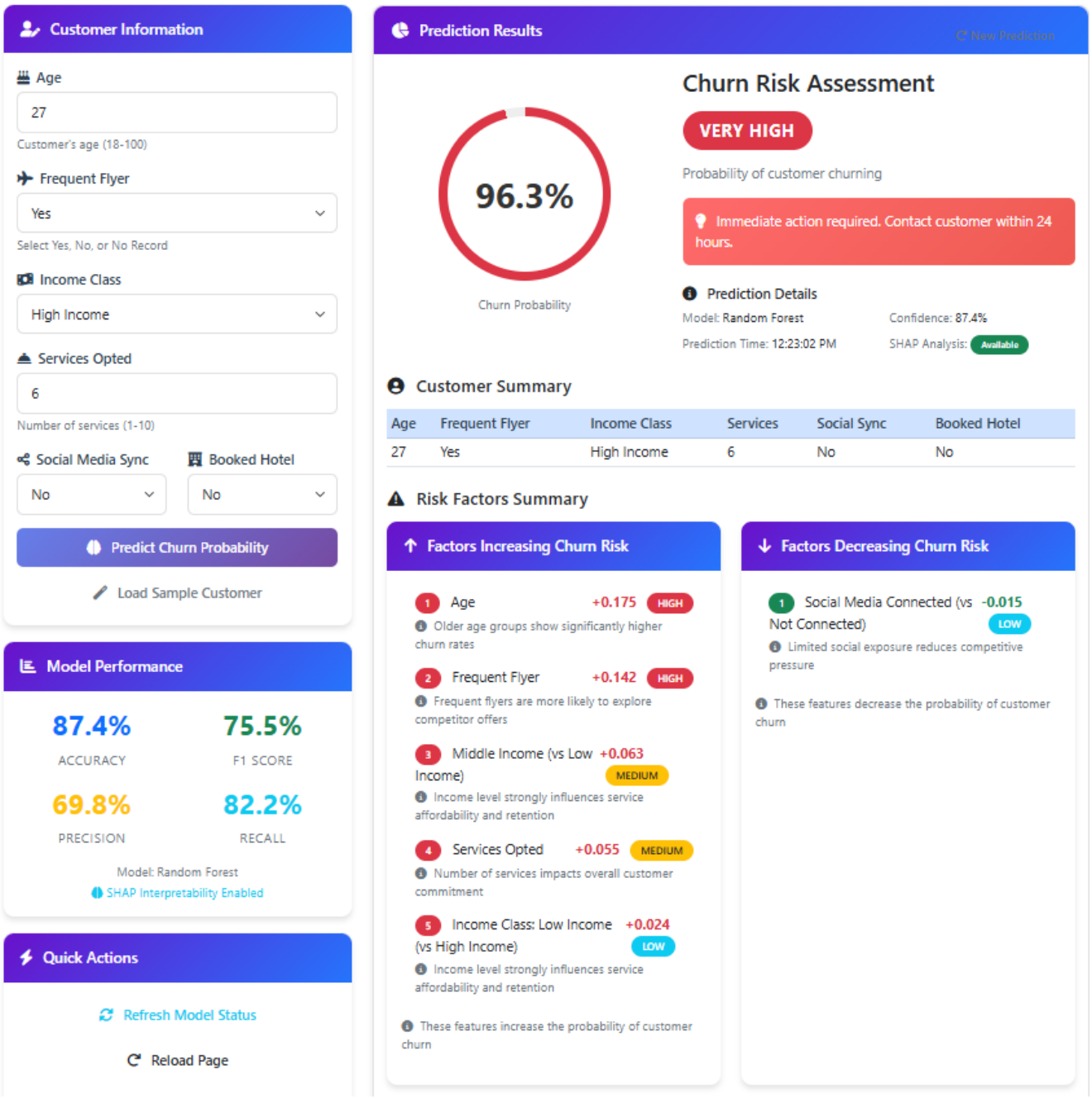



Figure 22: Customer Churn Prediction Web Interface

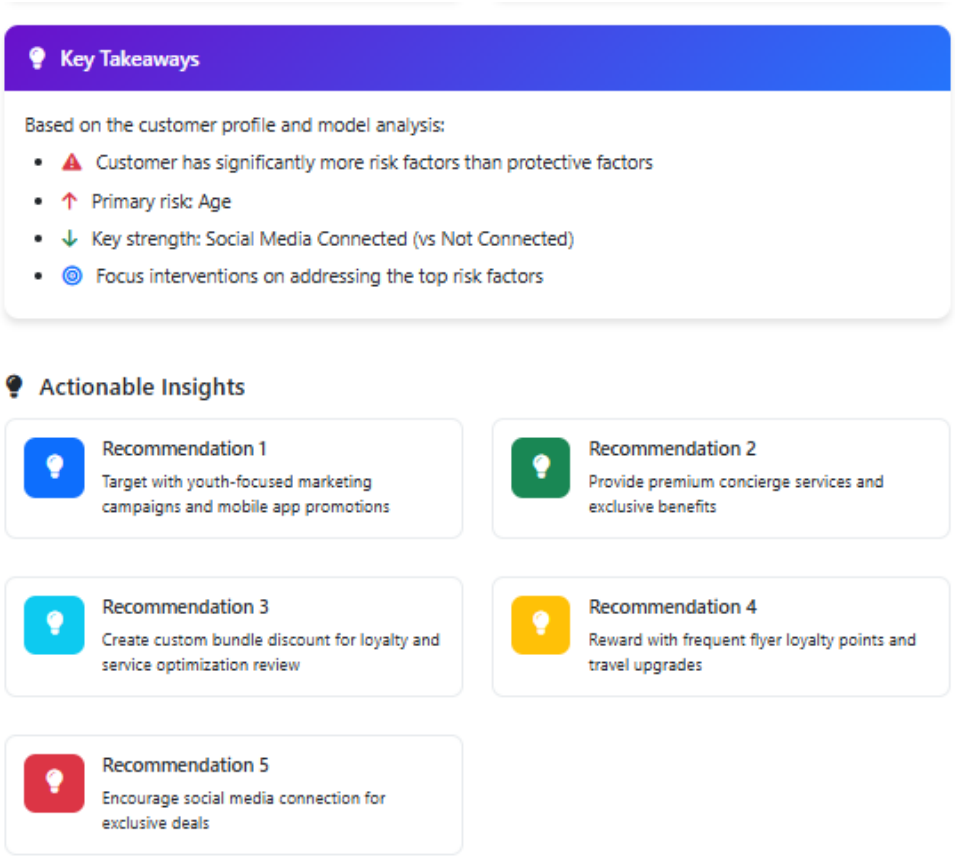



Figure 23: Customer Churn Prediction Web Interface - Actionable Insights

# 9 Conclusion

The work carried out in this project was guided by three main business goals, and the results showed that each of them has been met.

## Predictive Accuracy

One aim was to reach performance levels of at least 75% for the F1-score and 80% for Recall. The final Random Forest model met both targets, with an F1-score of 75.5% and a Recall of 82.2%. This indicates that the model can identify a large share of customers who are likely to churn.

## Driver Identification

The second goal was to identify the strongest indicators of churn so that the business could understand the underlying customer behaviour related to churn. Insights were drawn from EDA, model-based importance, and SHAP. EDA initially suggested that Age and several attributes had little influence. SHAP, however, revealed that Age was the second most influential churn factor after FrequentFlyer_Yes.

These deeper insights can guide the business in creating better loyalty and retention programs suited to its customer demographics.

### Strategic Support for Retention Planning

The third goal was to provide a practical tool to assist retention planning. A web-based prototype was developed that allows teams to enter customer details, view churn predictions in real-time, examine SHAP explanations, and access suggested follow-up actions. This helps the business to shift from broad campaigns to more focused, proactive responses.

Overall, the project has met all three business objectives and can be deployed to deliver clear business value. Operationally, it can help the business focus limited resources on the highest-risk customers instead of spending on broad, costly marketing campaigns that do little to address the churn problem. WIth the insights gained, it can strategically help the business to review and revamp existing loyalty and retention programs to better suit different customer segments.

Future improvements would be to refine the model using better sampling techniques (such as SMOTE) to handle imbalance, adjust thresholds to improve high-risk customer identification, and expand explainability through global SHAP views. The prototype can be integrated into the company's CRM platform and improved over time as new enhancements are rolled out.

# 10 Appendix

## 10.1 PowerPoint Slides

# Customer Travel Churn Prediction Project

Ethan Ong (s4407636)

# The Business Challenge

**Current Churn Rate :** 23%

**Impact:** Revenue loss, Higher Marketing costs, Reactive strategy

**Gap:** No predictive tools currently exist

# The Project Objectives

**Goal 1 :** Predictive Accuracy (Target: > 75% F1-Score)

Focus : Balance Precision & Recall

**Goal 2:** Identify Drivers (Why are they leaving?)

**Goal 3:** Strategic Support (Targeted retention planning)

# The Approach

**Method :** CRISP-DM Framework
**Pipeline :** Data Prep (Scaling/Encoding)
Stratified Split (80/20)
Modeling
Evaluation
**Models Tested :** Logistic Regression, Random Forest, Gradient Boosting, ANN

# Winning Model Performance

**Winner :** Random Forest Classifier

**Performance :** F1-Score (75.5% - Target Met)

Recall (82.2% - Target Met)

**Why Random Forest :** Best balance of finding churners vs. minimizing false alarms

# Decoding the ‘Why’ (Shap Analysis)

**Top Drivers :** 1. Frequent Flyers

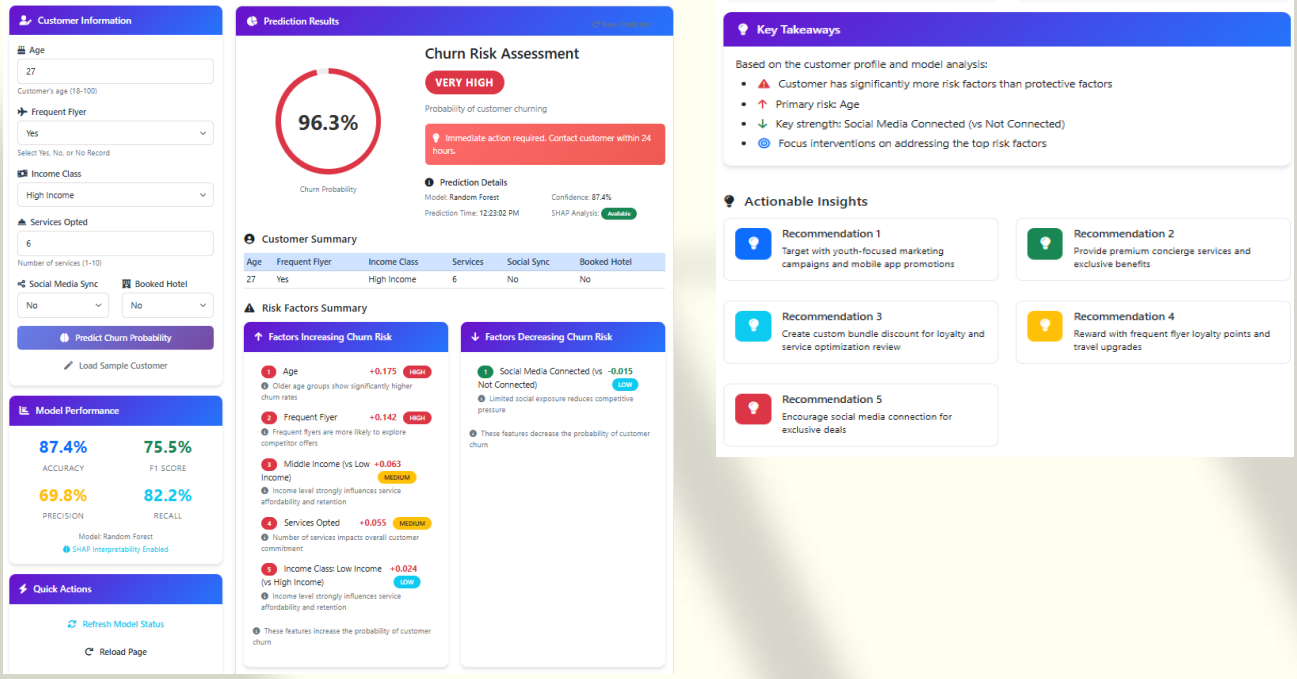
2. Age

3. Services Opted

**Key Finding :** Age - a hidden factor in EDA.

Shap – Age is 2nd strongest driver

# The Solution (Web Prototype)



# Business Value & Conclusion

**Impact :**

- Efficiency – Target only high-risk profiles
- Strategy **-** Revamp loyalty programs for frequent flyers

**Next step**:

- Refine model (advanced data techniques)
- Integrate into CRM

## 10.2 Data Preparation codes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import joblib
import json
import os


def prepare_data(file_path='Customertravel.csv', output_dir='prepared_data'):
    print("Loading data...")
    df = pd.read_csv(file_path)
    print(f"Data shape: {df.shape}")

    # Checking for missing values
    missing_values = df.isnull().sum()
    missing_cols = missing_values[missing_values > 0]
    # Case 1: No missing values at all
    if len(missing_cols) == 0:
        print("✓ No missing values found")

    # Case 2: There are missing values to handle
    else:
        print(f"⚠ Found missing values in columns: {list(missing_cols.index)}")
        print(f" Missing counts:\n{missing_cols}")

        # Process each column with missing values
        for col in missing_cols.index:

            # ------------------------------------------------------------
            # CASE A: Categorical columns (dtype = object)
            # ------------------------------------------------------------
            if df[col].dtype == 'object':
                mode_vals = df[col].mode()

                # Check if the mode exists
                if mode_vals.empty:
                    # No mode → fallback for empty or all-NaN categorical columns
                    fill_value = "Unknown"
```

```
            else:
                # Mode exists → take the first (most frequent) value
                fill_value = mode_vals.iloc[0]

            # Fill missing categorical values
            df[col] = df[col].fillna(fill_value)

        # ------------------------------------------------------
        # CASE B: Numeric columns
        # ------------------------------------------------------
        else:
            median_value = df[col].median()
            df[col] = df[col].fillna(median_value)

    print("✓ Missing values handled")

# Can't check for duplicates .. this function will drop 507 rows of customer data
# Can't check for outliers .. the only column applicable is Age.
# But we should not apply outlier on Age at all because the Age band from our collected data
# is very narrow (27-38) and this will reduce the model's ability to predict new
# market segments.

# Target variable
X = df.drop('Target', axis=1)
y = df['Target']

# Column types
numerical_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

print(f"\nNumerical: {numerical_cols}")
print(f"Categorical: {categorical_cols}")

#
# Using 'drop-first' for the oneHotencoding
#
categorical_pipeline = Pipeline([
    ('encoder', OneHotEncoder(
        drop='first',  # Standard practice
        sparse_output=False,
        handle_unknown='ignore'
```

```
    ))
])

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical_cols),
    ('cat', categorical_pipeline, categorical_cols)
])

#
# Splitting data
# Must remember to use random_state to 42 throughout pipeline
# so we get the same result everytime this code runs
# stratify set to y - to ensure we get the same ratio of churn in the split data
#
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
#
# Fit and transform
# Only apply fit_transform only on X_train
# Coz fit_transform learns parameters like mean, variance and then applies them
# Don't want information from X_test to get into the processing
# which will affect the model
#
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

#
# Merging numerical and categorical features
#
feature_names = numerical_cols.copy()
encoder = preprocessor.named_transformers_['cat'].named_steps['encoder']
cat_features = encoder.get_feature_names_out(categorical_cols)
feature_names.extend(cat_features)

#
# Important - must save the feature list
# After the onehotencoding - the number, the names and the order of the columns
# have changed.  We need the feature name list so Shap can use them later
# for the interpretation of the feature set.
#
```

```
category_info = {}
for col in categorical_cols:
    # Get original unique values
    unique_vals = sorted(X_train[col].dropna().unique())
    category_info[col] = {
        'original_values': list(unique_vals),
        'reference_category': unique_vals[0],  # The dropped one
        'encoded_columns': [f for f in cat_features if f.startswith(f"{col}_")]
    }

print(f"\n{'=' * 60}")
print("CATEGORY ENCODING DETAILS")
print(f"{'=' * 60}")
for col, info in category_info.items():
    print(f"\n{col}:")
    print(f"  All values: {info['original_values']}")
    print(f"  Reference (dropped): '{info['reference_category']}'")
    print(f"  Created features: {info['encoded_columns']}")

#
# Saving all the output files
#

os.makedirs(output_dir, exist_ok=True)

#
# Saving the processed files (scaling and onehotencoding)
#
pd.DataFrame(X_train_processed, columns=feature_names).to_csv(
    f'{output_dir}/X_train_processed.csv', index=False
)
pd.DataFrame(X_test_processed, columns=feature_names).to_csv(
    f'{output_dir}/X_test_processed.csv', index=False
)

#
# Saving targets as raw as no processing has been applied.
#
pd.DataFrame(y_train, columns=['Target']).to_csv(
    f'{output_dir}/y_train_raw.csv', index=False
)
```

```
    pd.DataFrame(y_test, columns=['Target']).to_csv(
        f'{output_dir}/y_test_raw.csv', index=False
    )

    # Save raw features for reference
    X_train.to_csv(f'{output_dir}/X_train_raw.csv', index=False)
    X_test.to_csv(f'{output_dir}/X_test_raw.csv', index=False)

    #
    # Using Joblib to save the preprocessor Python arrays
    # over pickle coz joblib handles large arrays better
    # needed for Shap explanations
    # want the exact object back and joblib ensures it loads fast and safely
    #
    joblib.dump(preprocessor, f'{output_dir}/preprocessor.joblib')

    #
    # Save metadata with everything needed (including the encoding choice
    #
    metadata = {
        "feature_names": feature_names,
        "numerical_features": numerical_cols,
        "categorical_features": categorical_cols,
        "category_info": category_info,
        "data_shapes": {
            "X_train_processed": X_train_processed.shape,
            "X_test_processed": X_test_processed.shape,
            "y_train_raw": y_train.shape,
            "y_test_raw": y_test.shape
        },
        "encoding_note": "Used drop='first' for one-hot encoding."
    }

    with open(f'{output_dir}/metadata.json', 'w', encoding='utf-8') as f:
        json.dump(metadata, f, indent=2)

    print(f"\n{'=' * 60}")
    print("DATA PREPARATION COMPLETE")
    print(f"{'=' * 60}")
    print(f"Output: {output_dir}/")
    print(f"Training set: {X_train_processed.shape}")
```

```
206     print(f"Test set: {X_test_processed.shape}")
207     print(f"Features: {len(feature_names)}")
208     print(f"\nFiles created:")
209     print(f"  X_train_processed.csv, X_test_processed.csv")
210     print(f"  y_train_raw.csv, y_test_raw.csv")
211     print(f"  X_train_raw.csv, X_test_raw.csv")
212     print(f"  preprocessor.joblib")
213     print(f"  metadata.json")
214
215     return X_train_processed, X_test_processed, y_train, y_test, preprocessor, metadata
216
217 #
218 # Main
219 #
220 if __name__ == "__main__":
221     # Prepare data
222     X_train, X_test, y_train, y_test, preprocessor, metadata = prepare_data()
223     print('Data prep is completed')
224
```

## 10.3 Model Training Codes

```
import pandas as pd
import numpy as np
import joblib
import json
import os
import warnings
from datetime import datetime

# Machine Learning
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, \
    classification_report, confusion_matrix
from sklearn.inspection import permutation_importance

# setting deterministic behaviour for ANN training
# to reduce run-to-run variability
os.environ['PYTHONHASHSEED'] = '42'
os.environ['TF_DETERMINISTIC_OPS'] = '1'   # best-effort deterministic ops
import random
random.seed(42)
np.random.seed(42)

# Deep Learning
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

#
# Set random seed to 42
# so we get the same result everytime this code runs
#
tf.random.set_seed(42)

#
```

```
# loading data
#
def load_prepared_data(data_dir='prepared_data'):

    # Loads prepared data
    # returns X_train, X_test, y_train, y_test (arrays) and metadata (dict)

    print("Loading prepared data...")

    # Check if files exist
    required_files = [
        f'{data_dir}/X_train_processed.csv',
        f'{data_dir}/X_test_processed.csv',
        f'{data_dir}/y_train_raw.csv',
        f'{data_dir}/y_test_raw.csv',
        f'{data_dir}/metadata.json'
    ]

    for file in required_files:
        if not os.path.exists(file):
            raise FileNotFoundError(f"Required file not found: {file}")

    # Load processed data
    X_train = pd.read_csv(f'{data_dir}/X_train_processed.csv')
    X_test = pd.read_csv(f'{data_dir}/X_test_processed.csv')
    y_train = pd.read_csv(f'{data_dir}/y_train_raw.csv').iloc[:, 0]
    y_test = pd.read_csv(f'{data_dir}/y_test_raw.csv').iloc[:, 0]

    # Load metadata
    with open(f'{data_dir}/metadata.json', 'r', encoding='utf-8') as f:
        metadata = json.load(f)

    print(f" Training set: {X_train.shape}")
    print(f" Test set: {X_test.shape}")
    print(f" Features: {X_train.shape[1]}")
    print(f" Target distribution (train): {pd.Series(y_train).value_counts().to_dict()}")

    return X_train.values, X_test.values, y_train.values, y_test.values, metadata

# ANN - chosen for the Deep Learning model
# ANN - doesnt have built-in mechanism for class-imbalance handling
```

```
# must be handled externally
def create_ann_model(input_dim, learning_rate=0.001):

    # Creating and compiling a simple ANN model for binary classification

    model = Sequential([
        # Hidden Layer 1 - learn high level feature interactions
        Dense(128, activation='relu', input_dim=input_dim,
            kernel_initializer='he_normal'),
        BatchNormalization(), # stabilizes and speeds up training
        Dropout(0.3), # prevents overfitting

        # Hidden Layer 2 - deeper pattern extraction
        Dense(64, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dropout(0.3),

        # Hidden Layer 3 - compress features before output
        Dense(32, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dropout(0.2),

        # Output Layer: sigmoid to return probability of churn
        Dense(1, activation='sigmoid')
    ])
    # configure optimizer
    optimizer = Adam(learning_rate=learning_rate)

    # compile model for binary classification - loss + key metrics
    model.compile(
        optimizer=optimizer,
        loss='binary_crossentropy',
        metrics=['accuracy', keras.metrics.AUC(name='auc')]
    )

    return model

# using logistic regression model as the baseline model
def train_logistic_regression(X_train, y_train, X_test, y_test, feature_names):

    print("\n" + "=" * 40)
```

```
124     print("TRAINING: Logistic Regression")
125     print("=" * 40)
126
127     # Initialize model with balanced class weights
128     # to handle class-imbalance
129     # liblinear - optimization algo (reliable for small datasets)
130     # c=1.0 - normal amount of regularization to keep model stable
131     model = LogisticRegression(
132         max_iter=1000,
133         random_state=42,
134         class_weight='balanced',
135         solver='liblinear',
136         C=1.0
137     )
138
139     # Train on 100% of training data
140     print(f"   Training on {len(X_train)} samples (100% training data)...")
141     model.fit(X_train, y_train)
142
143     # Predictions
144     y_pred = model.predict(X_test)
145     y_pred_proba = model.predict_proba(X_test)[:, 1]
146
147     # Logistic Regression has its built-in mechanism for feature importance
148     # During training - it learns coefficients(model.coef) for each feature
149     if len(model.coef_.shape) == 2 and model.coef_.shape[0] == 1:
150         # Binary classification
151         coefficients = np.abs(model.coef_[0])
152     elif len(model.coef_.shape) == 2:
153         # Multi-class, take average across classes
154         coefficients = np.abs(model.coef_).mean(axis=0)
155     else:
156         coefficients = np.abs(model.coef_)
157
158     feature_importance = pd.DataFrame({
159         'feature': feature_names[:len(coefficients)],
160         'importance': coefficients,
161         'importance_type': 'absolute_coefficient'
162     }).sort_values('importance', ascending=False)
163
164     print(f"   Model trained")
```

```
    print(f"   Feature importance calculated")

    return model, y_pred, y_pred_proba, feature_importance

# using Random Forest (ensemble) model for benchmark comparison
def train_random_forest(X_train, y_train, X_test, y_test, feature_names):
    print("\n" + "=" * 40)
    print("TRAINING: Random Forest")
    print("=" * 40)

    # Random Forest - in-built mechanism for handling class-imbalance
    # it gives more weight to minority class
    # so it doesnt ignore rare cases
    model = RandomForestClassifier(
        n_estimators=100,
        max_depth=10,
        min_samples_split=5,
        min_samples_leaf=2,
        class_weight='balanced',
        random_state=42,
        n_jobs=-1,
        bootstrap=True,
        max_features='sqrt'
    )

    # Train on 100% of training data
    print(f"   Training on {len(X_train)} samples (100% training data)...")
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Random Forest - also built-in mechanism for feature importance
    feature_importance = pd.DataFrame({
        'feature': feature_names,
        'importance': model.feature_importances_,
        'importance_type': 'gini_importance'
    }).sort_values('importance', ascending=False)

    print(f"   Model trained with {model.n_estimators} trees")
```

```
206     print(f"   Feature importance calculated")
207
208     return model, y_pred, y_pred_proba, feature_importance
209
210 # uses gradient boosting as another model for benchmarking comparison
211 # Gradient Boosting has a built-in mechanism for feature importance
212 # but none for handling class imbalance; must be handled externally
213 def train_gradient_boosting(X_train, y_train, X_test, y_test, feature_names):
214     print("\n" + "=" * 40)
215     print("TRAINING: Gradient Boosting")
216     print("=" * 40)
217
218     # Initialize model
219     model = GradientBoostingClassifier(
220         n_estimators=100,
221         learning_rate=0.1,
222         max_depth=5,
223         min_samples_split=5,
224         min_samples_leaf=2,
225         random_state=42,
226         subsample=0.8,
227         max_features='sqrt'
228     )
229
230     # Train on 100% of training data
231     print(f"   Training on {len(X_train)} samples (100% training data)...")
232     model.fit(X_train, y_train)
233
234     # Predictions
235     y_pred = model.predict(X_test)
236     y_pred_proba = model.predict_proba(X_test)[:, 1]
237
238     # Feature importance (split importance)
239     feature_importance = pd.DataFrame({
240         'feature': feature_names,
241         'importance': model.feature_importances_,
242         'importance_type': 'split_importance'
243     }).sort_values('importance', ascending=False)
244
245     print(f"   Model trained with {model.n_estimators} trees")
246     print(f"   Feature importance calculated")
```

```
247
248     return model, y_pred, y_pred_proba, feature_importance
249
250 # Using ANN for the deep learning model - benchmark comaprison
251 def train_ann(X_train, y_train, X_test, y_test, feature_names):
252     print("\n" + "=" * 40)
253     print("TRAINING: Artificial Neural Network")
254     print("=" * 40)
255
256     # Create model
257     model = create_ann_model(X_train.shape[1])
258
259     # Train on 100% of training data (NO validation split)
260     print(f"   Training on {len(X_train)} samples (100% training data)...")
261     print(f"   NO validation split - Using fixed epochs for fair comparison")
262
263     # using epochs = 50 - a conservative number to avoid overfitting
264     epochs = 50  # Conservative number to avoid overfitting
265
266     history = model.fit(
267         X_train, y_train,
268         epochs=epochs,  # Fixed epochs, no early stopping
269         batch_size=32,
270         verbose=0
271     )
272
273     # Training summary
274     n_epochs = len(history.history['loss'])
275     final_loss = history.history['loss'][-1]
276     final_accuracy = history.history['accuracy'][-1]
277
278     print(f"   Training completed in {n_epochs} epochs")
279     print(f"   Final training loss: {final_loss:.4f}")
280     print(f"   Final training accuracy: {final_accuracy:.4f}")
281
282     # Predictions
283     y_pred_proba = model.predict(X_test, verbose=0).flatten()
284     y_pred = (y_pred_proba > 0.5).astype(int)
285
286     # ANN has no built-in feature importance
287     # so permutation importance is used to estimate how much each feature affects predictions
```

```
    # this is an external method applied after training
    print("   Calculating permutation importance...")

    # Use lambda function for prediction
    predictor = lambda X: model.predict(X, verbose=0).flatten()

    # Calculate permutation importance on a reasonable subset
    sample_size = min(200, len(X_test))
    try:
        perm_result = permutation_importance(
            predictor,
            X_test[:sample_size],
            y_test[:sample_size],
            n_repeats=10,
            random_state=42,
            n_jobs=-1,
            scoring='neg_mean_squared_error'
        )

        feature_importance = pd.DataFrame({
            'feature': feature_names,
            'importance': perm_result.importances_mean,
            'importance_std': perm_result.importances_std,
            'importance_type': 'permutation_importance'
        }).sort_values('importance', ascending=False)

        print(f"    Permutation importance calculated on {sample_size} samples")

    except Exception as e:
        print(f"   Could not calculate permutation importance: {e}")
        print("   Using uniform importance as placeholder")

        # Create placeholder importance
        feature_importance = pd.DataFrame({
            'feature': feature_names,
            'importance': np.ones(len(feature_names)) / len(feature_names),
            'importance_std': np.zeros(len(feature_names)),
            'importance_type': 'uniform_placeholder'
        }).sort_values('importance', ascending=False)

    return model, y_pred, y_pred_proba, feature_importance, history
```

```
329
330 # calculating metrics - for Accuracy, Precision, Recall, F1-score, ROC-AUC
331 # and Confusion Metrics
332 def calculate_metrics(y_true, y_pred, y_pred_proba, model_name):
333
334     # Basic metrics
335     metrics = {
336         'model': model_name,
337         'accuracy': accuracy_score(y_true, y_pred),
338         'precision': precision_score(y_true, y_pred, zero_division=0),
339         'recall': recall_score(y_true, y_pred, zero_division=0),
340         'f1': f1_score(y_true, y_pred, zero_division=0),
341         'roc_auc': roc_auc_score(y_true, y_pred_proba) if len(np.unique(y_pred_proba)) > 1 else
342 0.5,
343         'n_samples': len(y_true)
344     }
345
346     # Confusion matrix components
347     cm = confusion_matrix(y_true, y_pred)
348     if cm.shape == (2, 2):
349         tn, fp, fn, tp = cm.ravel()
350         metrics.update({
351             'true_negatives': int(tn),
352             'false_positives': int(fp),
353             'false_negatives': int(fn),
354             'true_positives': int(tp)
355         })
356
357     # Classification report
358     report = classification_report(y_true, y_pred, output_dict=True)
359
360     return metrics, report
361
362 # evaluating all models - to get to the best performing one on F1-score
363 def evaluate_and_select_best_model(all_results):
364     print("\n" + "=" * 60)
365     print("MODEL EVALUATION & COMPARISON")
366     print("=" * 60)
367     print("All models trained on 100% of training data for fair comparison")
368
369     # Find best model based on F1-score (primary metric)
```

```
    best_model_name = None
    best_f1 = 0
    best_metrics = None

    # Also track other metrics for comparison
    all_metrics = {}

    print("\n Model Performance Summary (Test Set):")
    print("-" * 85)
    header = f"{'Model':<25} {'Accuracy':<10} {'Precision':<10} {'Recall':<10} {'F1-Score':<10}
{'ROC-AUC':<10}"
    print(header)
    print("-" * 85)

    for name, results in all_results.items():
        metrics = results['metrics']
        all_metrics[name] = metrics

        # Print performance row
        print(f"{name:<25} "
            f"{metrics['accuracy']:<10.4f} "
            f"{metrics['precision']:<10.4f} "
            f"{metrics['recall']:<10.4f} "
            f"{metrics['f1']:<10.4f} "
            f"{metrics['roc_auc']:<10.4f}")

        # Update best model
        if metrics['f1'] > best_f1:
            best_f1 = metrics['f1']
            best_model_name = name
            best_metrics = metrics

    print("-" * 85)

    # Show best model details
    print(f"\n BEST MODEL: {best_model_name}")
    print(f"   Primary Metric (F1-Score): {best_f1:.4f}")
    print(f"   Accuracy: {best_metrics['accuracy']:.4f}")
    print(f"   Precision: {best_metrics['precision']:.4f}")
    print(f"   Recall: {best_metrics['recall']:.4f}")
    print(f"   ROC-AUC: {best_metrics['roc_auc']:.4f}")
```

```
411
412     # Show runner-ups
413     print(f"\n Runner-up Models:")
414     sorted_models = sorted(all_metrics.items(), key=lambda x: x[1]['f1'], reverse=True)
415     for i, (name, metrics) in enumerate(sorted_models[1:4], 2):
416         print(f"   {i}. {name}: F1={metrics['f1']:.4f}, AUC={metrics['roc_auc']:.4f}")
417
418     return best_model_name, all_results[best_model_name]
419
420 # saving models and results
421 def save_models_and_results(all_results, best_model_name, best_model_results,
422                             X_test, y_test, output_dir='models'):
423     print(f"\n Saving models and results to '{output_dir}/'...")
424
425     # Create output directory
426     os.makedirs(output_dir, exist_ok=True)
427
428     # Track model info for summary
429     model_info = {}
430
431     print("\n Model Details:")
432     print("-" * 80)
433
434     for name, results in all_results.items():
435         # Create model-specific directory
436         model_dir = f'{output_dir}/{name.replace(" ", "_").lower()}'
437         os.makedirs(model_dir, exist_ok=True)
438
439         print(f"\n{name}:")
440
441         # Special handling for ANN models because it must be saved using Keras native format (h5)
442         # unlike the rest of the scikit-learn models
443         if name == 'Artificial Neural Network':
444             # Save Keras model
445             model_path = f'{model_dir}/model.h5'
446             results['model'].save(model_path)
447             print(f"   Model saved: {model_path}")
448
449             # Save training history - ANN learns over many epochs
450             # Useful for checking training behaviour and overfitting
451             history_path = f'{model_dir}/training_history.joblib'
```

```
452             joblib.dump(results['history'], history_path)
453             print(f"   Training history saved")
454
455         else:
456             # Save scikit-learn model
457             model_path = f'{model_dir}/model.joblib'
458             joblib.dump(results['model'], model_path)
459             print(f"   Model saved: {model_path}")
460
461         # Save metrics
462         metrics_path = f'{model_dir}/metrics.json'
463         with open(metrics_path, 'w', encoding='utf-8') as f:
464             json.dump(results['metrics'], f, indent=2)
465         print(f"   Metrics saved")
466
467         # Save feature importance
468         importance_path = f'{model_dir}/feature_importance.csv'
469         results['feature_importance'].to_csv(importance_path, index=False)
470         print(f"   Feature importance saved")
471
472         # Save classification report
473         report_path = f'{model_dir}/classification_report.json'
474         with open(report_path, 'w', encoding='utf-8') as f:
475             json.dump(results['report'], f, indent=2)
476         print(f"   Classification report saved")
477
478         # Display top features
479         top_features = results['feature_importance'].head(3)
480         print(f"  Top 3 features:")
481         for _, row in top_features.iterrows():
482             print(f"    - {row['feature']}: {row['importance']:.4f}")
483
484         # Store in model info
485         model_info[name] = {
486             'metrics': results['metrics'],
487             'feature_importance_top5': results['feature_importance'].head(5).to_dict('records'),
488             'model_path': model_path
489         }
490
491     # Save consolidated results
492     print(f"\n Saving consolidated results...")
```

```
493
494     # Save all model results
495     all_results_path = f'{output_dir}/all_model_results.json'
496     with open(all_results_path, 'w', encoding='utf-8') as f:
497         json.dump(model_info, f, indent=2)
498     print(f"   All model results saved: {all_results_path}")
499
500     # Save best model info
501     best_model_info = {
502         'best_model_name': best_model_name,
503         'best_model_metrics': best_model_results['metrics'],
504         'best_model_top_features':
505 best_model_results['feature_importance'].head(10).to_dict('records'),
506         'selection_criteria': 'highest_f1_score',
507         'training_note': 'All models trained on 100% of training data for fair comparison',
508         'ann_training_note': 'ANN trained with fixed epochs (50), no validation split',
509         'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
510         'recommendation': 'For consistent cross-model feature importance comparison, use SHAP
511 analysis'
512     }
513
514     best_info_path = f'{output_dir}/best_model_info.json'
515     with open(best_info_path, 'w', encoding='utf-8') as f:
516         json.dump(best_model_info, f, indent=2)
517     print(f"   Best model info saved: {best_info_path}")
518
519     # Save best model predictions
520     best_predictions = pd.DataFrame({
521         'true_label': y_test,
522         'predicted_label': best_model_results['y_pred'],
523         'prediction_probability': best_model_results['y_pred_proba']
524     })
525     predictions_path = f'{output_dir}/best_model_predictions.csv'
526     best_predictions.to_csv(predictions_path, index=False)
527     print(f"   Best model predictions saved: {predictions_path}")
528
529     # Save confusion matrix for best model
530     cm = confusion_matrix(y_test, best_model_results['y_pred'])
531     cm_df = pd.DataFrame(cm,
532                          index=['Actual Negative', 'Actual Positive'],
533                          columns=['Predicted Negative', 'Predicted Positive'])
```

```
534     cm_path = f'{output_dir}/best_model_confusion_matrix.csv'
535     cm_df.to_csv(cm_path)
536     print(f"   Confusion matrix saved: {cm_path}")
537
538     print(f"\n All files saved successfully!")
539
540     return model_info, best_model_info
541
542
543 def generate_training_summary(model_info, best_model_info, output_dir='models'):
544     """
545     Generate a human-readable training summary
546     """
547     summary_path = f'{output_dir}/training_summary.txt'
548
549     with open(summary_path, 'w', encoding='utf-8') as f:
550         f.write("=" * 70 + "\n")
551         f.write("MODEL TRAINING SUMMARY\n")
552         f.write("=" * 70 + "\n\n")
553
554         f.write(f"Timestamp: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
555         f.write(f"Best Model: {best_model_info['best_model_name']}\n")
556         f.write(f"Selection Criteria: {best_model_info['selection_criteria']}\n")
557         f.write(f"Training Note: {best_model_info['training_note']}\n")
558         f.write(f"ANN Training: {best_model_info['ann_training_note']}\n\n")
559
560         f.write("-" * 70 + "\n")
561         f.write("FAIR COMPARISON NOTES:\n")
562         f.write("-" * 70 + "\n")
563         f.write(" All models trained on 100% of training data\n")
564         f.write(" NO validation split for ANN (fixed 50 epochs)\n")
565         f.write(" Same train/test split for all models\n")
566         f.write(" Same evaluation metrics for all models\n\n")
567
568         f.write("-" * 70 + "\n")
569         f.write("PERFORMANCE COMPARISON\n")
570         f.write("-" * 70 + "\n\n")
571
572         # Create performance table
573         f.write(f"{'Model':<25} {'Accuracy':<10} {'F1-Score':<10} {'ROC-AUC':<10}\n")
574         f.write("-" * 55 + "\n")
```

```
575
576         for name, info in sorted(model_info.items(),
577                                  key=lambda x: x[1]['metrics']['f1'],
578                                  reverse=True):
579             metrics = info['metrics']
580             f.write(f"{name:<25} {metrics['accuracy']:<10.4f} "
581                     f"{metrics['f1']:<10.4f} {metrics['roc_auc']:<10.4f}\n")
582
583         f.write("\n" + "-" * 70 + "\n")
584         f.write(f"BEST MODEL DETAILS: {best_model_info['best_model_name']}\n")
585         f.write("-" * 70 + "\n\n")
586
587         metrics = best_model_info['best_model_metrics']
588         f.write(f"Accuracy:  {metrics['accuracy']:.4f}\n")
589         f.write(f"Precision: {metrics['precision']:.4f}\n")
590         f.write(f"Recall:    {metrics['recall']:.4f}\n")
591         f.write(f"F1-Score:  {metrics['f1']:.4f}\n")
592         f.write(f"ROC-AUC:   {metrics['roc_auc']:.4f}\n")
593
594         if 'true_positives' in metrics:
595             f.write(f"\nConfusion Matrix:\n")
596             f.write(f"  True Positives:  {metrics['true_positives']}\n")
597             f.write(f"  True Negatives:  {metrics['true_negatives']}\n")
598             f.write(f"  False Positives: {metrics['false_positives']}\n")
599             f.write(f"  False Negatives: {metrics['false_negatives']}\n")
600
601         f.write(f"\nTop 5 Features:\n")
602         for i, feat in enumerate(best_model_info['best_model_top_features'][:5], 1):
603             f.write(f"  {i}. {feat['feature']}: {feat['importance']:.4f}\n")
604
605     print(f"Training summary saved: {summary_path}")
606
607
608 def main():
609     try:
610         # Step 1: Load prepared data
611         print("\n STEP 1: Loading prepared data...")
612         X_train, X_test, y_train, y_test, metadata = load_prepared_data()
613         feature_names = metadata['feature_names']
614
615         # Step 2: Train all models (ALL using 100% training data)
```

```
616         print("\n STEP 2: Training models...")
617         print("   All models using 100% of training data for fair comparison")
618         all_results = {}
619
620         # 2.1 Logistic Regression (100% training data)
621         lr_model, lr_pred, lr_proba, lr_importance = train_logistic_regression(
622             X_train, y_train, X_test, y_test, feature_names
623         )
624         lr_metrics, lr_report = calculate_metrics(y_test, lr_pred, lr_proba, 'Logistic Regression')
625
626         all_results['Logistic Regression'] = {
627             'model': lr_model,
628             'y_pred': lr_pred,
629             'y_pred_proba': lr_proba,
630             'feature_importance': lr_importance,
631             'metrics': lr_metrics,
632             'report': lr_report
633         }
634
635         # 2.2 Random Forest (100% training data)
636         rf_model, rf_pred, rf_proba, rf_importance = train_random_forest(
637             X_train, y_train, X_test, y_test, feature_names
638         )
639         rf_metrics, rf_report = calculate_metrics(y_test, rf_pred, rf_proba, 'Random Forest')
640
641         all_results['Random Forest'] = {
642             'model': rf_model,
643             'y_pred': rf_pred,
644             'y_pred_proba': rf_proba,
645             'feature_importance': rf_importance,
646             'metrics': rf_metrics,
647             'report': rf_report
648         }
649
650         # 2.3 Gradient Boosting (100% training data)
651         gb_model, gb_pred, gb_proba, gb_importance = train_gradient_boosting(
652             X_train, y_train, X_test, y_test, feature_names
653         )
654         gb_metrics, gb_report = calculate_metrics(y_test, gb_pred, gb_proba, 'Gradient Boosting')
655
656         all_results['Gradient Boosting'] = {
```

```
            'model': gb_model,
            'y_pred': gb_pred,
            'y_pred_proba': gb_proba,
            'feature_importance': gb_importance,
            'metrics': gb_metrics,
            'report': gb_report
        }

        # 2.4 Artificial Neural Network (100% training data, NO validation split)
        ann_model, ann_pred, ann_proba, ann_importance, ann_history = train_ann(
            X_train, y_train, X_test, y_test, feature_names
        )
        ann_metrics, ann_report = calculate_metrics(y_test, ann_pred, ann_proba, 'Artificial
Neural Network')

        all_results['Artificial Neural Network'] = {
            'model': ann_model,
            'history': ann_history,
            'y_pred': ann_pred,
            'y_pred_proba': ann_proba,
            'feature_importance': ann_importance,
            'metrics': ann_metrics,
            'report': ann_report
        }

        # Step 3: Evaluate and select best model
        print("\n STEP 3: Evaluating models...")
        best_model_name, best_model_results = evaluate_and_select_best_model(all_results)

        # Step 4: Save models and results
        print("\n STEP 4: Saving results...")
        model_info, best_model_info = save_models_and_results(
            all_results, best_model_name, best_model_results, X_test, y_test
        )

        # Step 5: Generate summary
        generate_training_summary(model_info, best_model_info)

        return True

    except Exception as e:
```

```
698         print(f"\n MODEL TRAINING FAILED: {str(e)}")
699         print("\nDebugging info:")
700         import traceback
701         traceback.print_exc()
702
703         # Try to save error log
704         try:
705             error_log =
706     f"model_training_error_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
707             with open(error_log, 'w', encoding='utf-8') as f:
708                 f.write(f"Error: {str(e)}\n\n")
709                 traceback.print_exc(file=f)
710             print(f"\nError log saved to: {error_log}")
711         except:
712             pass
713
714         return False
715
716     if __name__ == "__main__":
717         # Start the training pipeline
718         success = main()
719
720         # Exit with appropriate code
721         import sys
722
723         sys.exit(0 if success else 1)
724
```

## 10.4 Shap Analysis Codes

```
import pandas as pd
import numpy as np
import joblib
import json
import os
import shap
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')

# Set style for better visualizations
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")

# Load models and test data for Shap analysis
# This includes the Feature_name list saved during the data preprocessing stage
def load_models_and_data(models_dir='models', data_dir='prepared_data'):
    print("Loading models and data for SHAP analysis...")

    # Loading the best model info
    with open(f'{models_dir}/best_model_info.json', 'r', encoding='utf-8') as f:
        best_model_info = json.load(f)

    best_model_name = best_model_info['best_model_name']
    print(f"Best model: {best_model_name}")

    # Load test data
    X_test = pd.read_csv(f'{data_dir}/X_test_processed.csv')
    y_test = pd.read_csv(f'{data_dir}/y_test_raw.csv').iloc[:, 0]

    # Load metadata for feature names and category info
    with open(f'{data_dir}/metadata.json', 'r', encoding='utf-8') as f:
        metadata = json.load(f)
    feature_names = metadata['feature_names']
    category_info = metadata.get('category_info', {})

    # Load the best model
    model_dir = f'{models_dir}/{best_model_name.replace(" ", "_").lower()}'

```

```
42      if best_model_name == 'Artificial Neural Network':
43          # If best model is ANN - then need to load model.h5
44          # coz keras stores using its own native saving format
45          import tensorflow as tf
46          model = tf.keras.models.load_model(f'{model_dir}/model.h5')
47          # Compile the model
48          model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
49      else:
50          # The rest of the models uses joblib coz they are all scikit-learn models
51          model = joblib.load(f'{model_dir}/model.joblib')
52
53      return model, X_test.values, y_test.values, feature_names, best_model_name,
54  category_info
55
56
57  def create_shap_explainer(model, model_name, X_sample):
58      # Shap uses different explainer engines based on model type
59      print(f"\nCreating SHAP explainer for {model_name}...")
60
61      try:
62          if model_name == 'Artificial Neural Network':
63              # For ANN - use KernelExplainer with a simple prediction function
64
65              def model_predict(data):
66                  # Shap needs a 1dimensional vector of model outputs
67                  # ANN often returns predictions in a 2 column format (n,2)
68                  # Shap must receive only 1 output per sample
69                  # Hence model_predict is a wrapper that transforms ANN's prediction output
70                  # into a proper 1D vector of probabilities for Class 1 (ie. Churn).
71                  if len(data.shape) == 1:
72                      data = data.reshape(1, -1)
73                  # Predict and return probability of class 1 (churn)
74                  predictions = model.predict(data, verbose=0)
75                  # If predictions are 2D with shape (n, 1), flatten
76                  if predictions.shape[1] == 1:
77                      return predictions.flatten()
78                  # If predictions are 2D with shape (n, 2), return probability of class 1
79                  elif predictions.shape[1] == 2:
80                      return predictions[:, 1]
81                  else:
82                      return predictions
```

```
83
84          # Use a background sample for KernelExplainer
85          # KernelExplainer is computationally expensive
86          # using ony 50 out of 200 records from X_sample
87          # is std practice to keep Shap computations manageable
88
89          background = shap.sample(X_sample, min(50, X_sample.shape[0]))
90          explainer = shap.KernelExplainer(model_predict, background)
91          shap_values = explainer.shap_values(X_sample, nsamples=100, silent=True)
92
93      elif model_name in ['Random Forest', 'Gradient Boosting']:
94          # Use TreeEXplainer for Tree-based models
95          explainer = shap.TreeExplainer(model)
96          shap_values = explainer.shap_values(X_sample)
97
98          # Handle binary classification output
99          if isinstance(shap_values, list) and len(shap_values) == 2:
100             shap_values = shap_values[1]  # Positive class
101
102     elif model_name == 'Logistic Regression':
103         # Use LinearExplainer for Linear model
104         explainer = shap.LinearExplainer(model, X_sample)
105         shap_values = explainer.shap_values(X_sample)
106
107     else:
108         # Use KernelExplainer again for anything not matching
109         # the above models
110         def model_predict(data):
111             if len(data.shape) == 1:
112                 data = data.reshape(1, -1)
113             return model.predict_proba(data)[:, 1]
114
115         background = shap.sample(X_sample, min(50, X_sample.shape[0]))
116         explainer = shap.KernelExplainer(model_predict, background)
117         shap_values = explainer.shap_values(X_sample, nsamples=100, silent=True)
118
119     print(f"SHAP explainer created successfully")
120
121     # Ensure shap_values is numpy array
122     if hasattr(shap_values, 'values'):
123         shap_array = shap_values.values
```

```
        else:
            shap_array = np.array(shap_values)

        # For ANN KernelExplainer, sometimes we get list of arrays
        if isinstance(shap_array, list):
            shap_array = np.array(shap_array)

        return explainer, shap_array

    except Exception as e:
        print(f"Could not create standard SHAP explainer: {e}")
        print("Creating approximate SHAP values using permutation...")
        print("model name : ", model_name)
        # When the above specialized Shap explainers fail,
        # the Fallback PermutationExplainer is used
        # It works - coz it does not rely on model internals
        # It just estimates feature importance by shuffling features
        # and measuring the change in predictions

        if model_name == 'Artificial Neural Network':
            def model_predict(data):
                if len(data.shape) == 1:
                    data = data.reshape(1, -1)
                predictions = model.predict(data, verbose=0)
                if predictions.shape[1] == 1:
                    return predictions.flatten()
                elif predictions.shape[1] == 2:
                    return predictions[:, 1]
                else:
                    return predictions

            explainer = shap.Explainer(model_predict, X_sample[:50])
        else:
            explainer = shap.Explainer(model.predict_proba, X_sample[:50])

        shap_values = explainer(X_sample[:50])

        if hasattr(shap_values, 'values'):
            shap_array = shap_values.values
        else:
            shap_array = np.array(shap_values)
```

```
165
166         return explainer, shap_array
167
168
169     def calculate_shap_importance(shap_values, feature_names):
170         # Calculate shap importance from shap values
171         print("\nCalculating SHAP-based feature importance...")
172
173         # Shap returns inconsistent types across explainers
174         # so have to ensure shap_values is numpy array
175         if hasattr(shap_values, 'values'):
176             shap_array = shap_values.values
177         else:
178             shap_array = np.array(shap_values)
179
180         # Shap outputs vary across explainers
181         # so have to handle different shapes
182         if len(shap_array.shape) == 3:
183             # Multi-class: take mean across classes
184             shap_array = np.mean(np.abs(shap_array), axis=2)
185         elif len(shap_array.shape) == 1:
186             # Single sample: reshape
187             shap_array = shap_array.reshape(1, -1)
188
189         # Compute global feature importance
190         # Shap provides values per sample; to get 'global' importance
191         # we take mean(abs(Shap) across all samples.
192         # this is the sfd metric used in Shap summary plots
193         if len(shap_array.shape) == 2:
194             shap_importance = np.abs(shap_array).mean(axis=0)
195         else:
196             shap_importance = np.abs(shap_array)
197
198         # ensure feature count matches
199         # Shap occasionally outputs fewer features due to
200         # dropped columns, encoding issues etc.
201         n_features = min(len(shap_importance), len(feature_names))
202
203         # Numpy arrays do not store column labels
204         # Hence the need to convert to Dataframe to make it easier
205         # for business interpretation, plotting and exporting
```

```
206     feature_importance = pd.DataFrame({
207         'feature': feature_names[:n_features],
208         'shap_importance': shap_importance[:n_features],
209         'shap_importance_normalized': shap_importance[:n_features] /
210 shap_importance[:n_features].sum() * 100
211     }).sort_values('shap_importance', ascending=False)
212
213     print(f"Calculated SHAP importance for {len(feature_importance)} features")
214     return feature_importance
215
216 # Build a clean business friendly table
217 # compute Shap direction and suggest simple actions
218 def create_business_friendly_interpretation(
219     feature_importance: pd.DataFrame,
220     category_info: dict,
221     shap_values: np.ndarray = None,
222     feature_names: list = None
223 ) -> pd.DataFrame:
224
225     # Copy input and ensure required column exists
226     bi = feature_importance.copy().reset_index(drop=True)
227     if 'feature' not in bi.columns:
228         raise ValueError("feature_importance must contain a 'feature' column")
229
230     # Set up new columns for buisness friendly outputs
231     bi['feature_type'] = 'numerical'
232     bi['business_interpretation'] = ''
233     bi['impact_direction'] = 'neutral'
234     bi['mean_shap'] = np.nan
235     bi['business_meaning'] = ''
236     bi['recommended_action'] = ''
237     # Use normalized Shap importance if available ; fallback to percentage share
238     bi['confidence_pct'] = bi.get('shap_importance_normalized',
239             (bi['shap_importance'] / bi['shap_importance'].sum() * 100)).fillna(0)
240
241     # Convert Shap values to 2D and align to features
242     # Coz SHap output can come in several shapes and we need to standardize to
243     # a single consistent format before computing per feature averages
244     mean_shap_by_feature = None
245     if shap_values is not None:
246         # normalize shap_values into a 2D array (N, F)
```

```
247         try:
248             if hasattr(shap_values, 'values'):
249                 arr = shap_values.values
250             else:
251                 arr = np.array(shap_values)
252 
253             # If shap returns 3-d (classes), average across classes (signed)
254             if arr.ndim == 3:
255                 # average across classes then across samples
256                 arr = np.mean(arr, axis=2)
257 
258             if arr.ndim == 1:
259                 arr = arr.reshape(1, -1)
260 
261             # If arr shape mismatches feature count but feature_names provided, try to align
262 (best-effort)
263             if feature_names is not None and arr.shape[1] != len(feature_names):
264                 # if arr has fewer columns, assume first columns correspond to features in
265 feature_importance order
266                 # otherwise, if arr has more, truncate
267                 n = min(arr.shape[1], len(feature_names))
268                 arr = arr[:, :n]
269 
270             mean_shap_by_feature = np.mean(arr, axis=0)  # signed mean SHAP per feature
271 
272         except Exception:
273             mean_shap_by_feature = None
274 
275     # business glossary - readable menaing for known features
276     business_glossary = {
277         'ServicesOpted': 'Number of services / products the customer uses',
278         'Ages': 'Customer age',
279         'FrequentFlyer': 'Whether customer is in the frequent-flyer (loyalty) program',
280         'AnnualIncomeClass': 'Customer income segment (Low/Middle/High)',
281         'AccountSyncedToSocialMedia': 'Customer connected account to social media',
282         'BookedHotelOrNot': 'Whether customer booked hotel (engagement indicator)',
283         # add more domain-specific mappings here
284     }
285 
286     # Simple rule based action generator
287     def recommend_action(feature, direction, mean_shap, conf):
```

```
288         # return short suggested action based on feature impact
289         f = feature
290         d = direction
291
292         if 'ServicesOpted' in f:
293           if d == 'increases churn':
294             return 'Offer targeted upsell bundle or 30-day free trial to increase engagement'
295           elif d == 'reduces churn':
296             return 'Highlight existing service benefits in retention messaging'
297         if 'FrequentFlyer' in f:
298           if d == 'increases churn':
299             return 'Target with loyalty incentives and travel benefits'
300           else:
301             return 'Maintain loyalty perks; consider cross-sell'
302         if 'AnnualIncomeClass' in f:
303           if 'Low' in f or ('Low Income' in f):
304             return 'Offer budget bundles and flexible payment options'
305           else:
306             return 'Promote premium add-ons and concierge offers'
307         if 'AccountSyncedToSocialMedia' in f:
308           if d == 'increases churn':
309             return 'Prompt account completion and offer referral incentives'
310           else:
311             return 'Use social channels for retention campaigns'
312         if 'BookedHotel' in f or 'BookedHotelOrNot' in f:
313           if d == 'increases churn':
314             return 'Provide hotel loyalty rewards and special packages'
315           else:
316             return 'Upsell complementary travel services'
317         if 'Age' in f or 'Ages' in f:
318           if d == 'increases churn':
319             return 'Use segment-specific offers (youth or senior campaigns)'
320           else:
321             return 'Maintain general retention messaging'
322         # fallback
323         if d == 'increases churn':
324           return 'Investigate root cause; consider targeted retention tests'
325         if d == 'reduces churn':
326           return 'Leverage this feature in acquisition/retention messaging'
327         return ''
328
```

```
    # Build business friendly rows
    for idx, row in bi.iterrows():
        feat = str(row['feature'])
        # determine feature type and readable interpretation
        is_cat = False
        for cat_feature, info in (category_info or {}).items():
            enc_cols = info.get('encoded_columns', []) or []
            if feat in enc_cols:
                # this is an encoded categorical column
                is_cat = True
                actual_value = feat.replace(f"{cat_feature}_", "")
                ref = info.get('reference_category', '')
                bi.at[idx, 'feature_type'] = 'categorical'
                bi.at[idx, 'business_interpretation'] = \
                    f"When {cat_feature} is '{actual_value}' (vs '{ref}')"
                break

        if not is_cat:
            # numerical fallback
            bi.at[idx, 'business_interpretation'] = f"{feat} (numerical value)"

        # business meaning from glossary or fallback
        base_name = feat.split('_')[0]
        bi.at[idx, 'business_meaning'] = business_glossary.get(base_name,
                                            business_glossary.get(feat, 'Feature related to customer
behaviour'))

        # Assign mean SHap and churn direction
        if mean_shap_by_feature is not None and idx < len(mean_shap_by_feature):
            m = float(mean_shap_by_feature[idx])
            bi.at[idx, 'mean_shap'] = m
            # threshold to call neutral
            eps = max(1e-6, np.abs(mean_shap_by_feature).max() * 0.01)
            if m > eps:
                dir_text = 'increases churn'
            elif m < -eps:
                dir_text = 'reduces churn'
            else:
                dir_text = 'neutral'
            bi.at[idx, 'impact_direction'] = dir_text
        else:
```

```
            bi.at[idx, 'impact_direction'] = 'neutral'
            bi.at[idx, 'mean_shap'] = np.nan

        # recommended action (rule-based)
        bi.at[idx, 'recommended_action'] = recommend_action(feat, bi.at[idx,
'impact_direction'],
                                                bi.at[idx, 'mean_shap'], bi.at[idx, 'confidence_pct'])

    # sort by shap_importance descending (preserve original sort if present)
    if 'shap_importance' in bi.columns:
        bi = bi.sort_values('shap_importance', ascending=False).reset_index(drop=True)

    # add rank
    bi.insert(0, 'rank', range(1, len(bi) + 1))

    # final column ordering (extend as needed)
    cols = [
        'rank', 'feature', 'feature_type', 'business_interpretation',
        'business_meaning', 'impact_direction', 'mean_shap',
        'shap_importance', 'shap_importance_normalized',
        'confidence_pct', 'recommended_action'
    ]
    # keep only existing columns in that order
    cols = [c for c in cols if c in bi.columns]
    bi = bi[cols]

    return bi



def generate_shap_plots(explainer, shap_values, X_sample, feature_names,
                        category_info, output_dir='shap_results'):
    # Generate and save Shap Feature Importance (Bar Plot)
    print(f"\nGenerating SHAP Feature Importance (Bar Plot)")

    # Create output directory
    os.makedirs(output_dir, exist_ok=True)

    # Ensure shapes match
    if shap_values.shape[0] != X_sample.shape[0]:
        print(f"Adjusting SHAP values shape: {shap_values.shape} to match X_sample:
```

```
411     {X_sample.shape}")
412         # If SHAP values are for a single class in binary classification
413         if len(shap_values.shape) == 2 and shap_values.shape[0] != X_sample.shape[0]:
414             # Try transposing
415             if shap_values.shape[1] == X_sample.shape[0]:
416                 shap_values = shap_values.T
417             # Or take first row if it's representative
418             elif shap_values.shape[0] == 1:
419                 shap_values = np.tile(shap_values, (X_sample.shape[0], 1))
420
421     # Create better feature names with interpretation
422     interpretable_feature_names = []
423     for feature in feature_names[:X_sample.shape[1]]:
424         interpretation_found = False
425
426         # Check if this is an encoded categorical feature
427         for cat_feature, info in category_info.items():
428             if feature in info['encoded_columns']:
429                 actual_value = feature.replace(f"{cat_feature}_", "")
430                 ref_category = info['reference_category']
431                 interpretable_feature_names.append(
432                     f"{cat_feature}: {actual_value} vs {ref_category}"
433                 )
434                 interpretation_found = True
435                 break
436
437         # If not an encoded feature, use original name
438         if not interpretation_found:
439             interpretable_feature_names.append(feature)
440
441     # Ensure we have the right number of feature names
442     interpretable_feature_names = interpretable_feature_names[:X_sample.shape[1]]
443
444     #
445     # Summary Bar Plot
446     #
447     try:
448         plt.figure(figsize=(10, 8))
449         shap.summary_plot(shap_values, X_sample,
450                           feature_names=interpretable_feature_names,
451                           plot_type="bar", show=False)
```

```
452         plt.title("SHAP Feature Importance (Bar Plot)", fontsize=14, fontweight='bold')
453         plt.tight_layout()
454         plt.savefig(f'{output_dir}/shap_summary_bar.png', dpi=150, bbox_inches='tight')
455         plt.close()
456         print("SHAP summary bar plot saved")
457     except Exception as e:
458         print(f"Could not generate bar plot: {e}")
459
460     return shap_values
461
462 def save_comprehensive_results(feature_importance, business_interpretation,
463                                category_info, model_name, shap_values,
464                                output_dir='shap_results'):
465     # Save comprehensive Shap results - enriched + concise and summary files
466     os.makedirs(output_dir, exist_ok=True)
467
468     print(f"\nSaving comprehensive SHAP results...")
469
470     # Defensive copy and alignment
471     comprehensive = business_interpretation.copy().reset_index(drop=True)
472
473     # Ensure rank exists BEFORE saving enriched file
474     # saving rank early - so saved files have a clear consistent feature order
475     if 'rank' not in comprehensive.columns:
476         comprehensive.insert(0, 'rank', range(1, len(comprehensive) + 1))
477
478     # Save enriched full file (preserve all enrichment columns)
479     enriched_path = f'{output_dir}/shap_feature_importance_enriched.csv'
480     comprehensive.to_csv(enriched_path, index=False)
481
482     # Create a concise "main" file for simpler consumption (choose columns present)
483     # Concise file gives a clean summary for quick use; the full file is too detailed.
484     main_cols = [
485         'rank', 'feature', 'business_interpretation', 'feature_type',
486         'impact_direction', 'mean_shap', 'shap_importance', 'shap_importance_normalized',
487         'recommended_action'
488     ]
489     # keep only existing columns in that order
490     main_cols = [c for c in main_cols if c in comprehensive.columns]
491     concise_df = comprehensive[main_cols].copy()
492
```

```
493     main_path = f'{output_dir}/shap_feature_importance.csv'
494     concise_df.to_csv(main_path, index=False)
495
496     # Build reference categories safely
497     reference_data = []
498     for cat_feature, info in (category_info or {}).items():
499       reference_data.append({
500         'categorical_feature': cat_feature,
501         'reference_category': info.get('reference_category', ''),
502         'all_categories': ', '.join(info.get('original_values', [])),
503         'encoded_features_in_model': ', '.join(info.get('encoded_columns', []))
504       })
505
506     reference_df = pd.DataFrame(reference_data)
507     reference_path = f'{output_dir}/reference_categories_info.csv'
508     reference_df.to_csv(reference_path, index=False)
509
510     # Build summary (safe shap values handling)
511     summary = {
512       'model_name': model_name,
513       'total_features': len(feature_importance),
514       'top_3_features': concise_df.head(3).to_dict('records'),
515       'reference_categories': reference_data,
516       'visualization_files': [
517         'shap_summary_bar.png',
518       ],
519       'dependence_plots': []
520     }
521
522     # Compute shap_importance_vals
523     try:
524       if hasattr(shap_values, 'values'):
525         shap_arr = np.array(shap_values.values)
526       else:
527         shap_arr = np.array(shap_values)
528
529       # handle 3-d (classes) or 1-d single sample
530       if shap_arr.ndim == 3:
531         shap_arr = np.mean(shap_arr, axis=2)
532       if shap_arr.ndim == 1:
533         shap_arr = shap_arr.reshape(1, -1)
```

```
534
535         shap_importance_vals = np.abs(shap_arr).mean(axis=0)
536     except Exception:
537         shap_importance_vals = None
538
539     if shap_importance_vals is not None:
540         n_top = min(4, len(shap_importance_vals))
541         top_indices = np.argsort(shap_importance_vals)[-n_top:][::-1]
542         for idx in top_indices:
543             if idx < len(feature_importance):
544                 feature = feature_importance.iloc[idx]['feature']
545                 clean_name = str(feature).replace(' ', '_').replace('/', '_').replace(':', '_')
546                 summary['dependence_plots'].append(f'shap_dependence_{clean_name}.png')
547
548     summary_path = f'{output_dir}/shap_analysis_summary.json'
549     with open(summary_path, 'w', encoding='utf-8') as f:
550         json.dump(summary, f, indent=2)
551
552     print(f"Enriched results saved to: {enriched_path}")
553     print(f"Concise results saved to: {main_path}")
554     print(f"Reference info saved to: {reference_path}")
555     print(f"Summary saved to: {summary_path}")
556
557     return comprehensive
558
559 def main():
560     """Main SHAP analysis pipeline - COMPLETE WITH ALL VISUALIZATIONS"""
561     print("=" * 60)
562     print("SHAP ANALYSIS PIPELINE")
563     print("=" * 60)
564     print("Generating comprehensive results with ALL visualizations")
565     print("=" * 60)
566
567     try:
568         # Step 1: Load models and data
569         model, X_test, y_test, feature_names, model_name, category_info =
570 load_models_and_data()
571
572         # Use sample for faster computation
573         sample_size = min(200, X_test.shape[0])
574         X_sample = X_test[:sample_size]
```

```
575
576         print(f"\nData shapes:")
577         print(f"  X_sample shape: {X_sample.shape}")
578         print(f"  Number of features: {len(feature_names)}")
579
580         # Step 2: Create SHAP explainer
581         explainer, shap_values = create_shap_explainer(model, model_name, X_sample)
582
583         print(f"\nSHAP values shape: {shap_values.shape}")
584
585         # Step 3: Calculate importance
586         feature_importance = calculate_shap_importance(shap_values, feature_names)
587
588         # Step 4: Generate ALL visualizations
589         shap_vals = generate_shap_plots(explainer, shap_values, X_sample,
590                                         feature_names, category_info)
591
592         # Step 5: Create business interpretation
593         business_interpretation = create_business_friendly_interpretation(
594             feature_importance, category_info, shap_values=shap_vals,
595     feature_names=feature_names
596         )
597
598         # Step 6: Save comprehensive results
599         comprehensive_results = save_comprehensive_results(
600             feature_importance, business_interpretation, category_info,
601             model_name, shap_vals
602         )
603
604         # ============================================================
605         # SHAP OUTPUT SUMMARY
606         # ============================================================
607
608         print("\n SHAP OUTPUT FILES")
609         print("-" * 60)
610
611         print("\n Data Files:")
612         print("- shap_feature_importance_enriched.csv")
613         print("- shap_feature_importance.csv")
614         print("- reference_categories_info.csv")
615         print("- shap_analysis_summary.json")
```

```
616
617         print("\n Visualization Files:")
618         print("- shap_summary_bar.png")
619
620         # Reference categories
621         if category_info:
622             print(f"\nReference categories (dropped during one-hot encoding):")
623             print("-" * 60)
624             for cat_feature, info in category_info.items():
625                 ref = info.get('reference_category', '')
626                 enc = ', '.join(info.get('encoded_columns', []))
627                 print(f"{cat_feature}: '{ref}'")
628                 print(f"Model uses: {enc}")
629
630         print(f"\n All files saved to: 'shap_results/' directory")
631         print(" Ready for deployment and presentation")
632
633
634     except Exception as e:
635         print(f"\n SHAP ANALYSIS FAILED DURING SUMMARY OUTPUT: {e}")
636         import traceback
637         traceback.print_exc()
638         return False
639
640
641 if __name__ == "__main__":
642     main()
643
```

## 10.5 App.py Codes

```
from flask import Flask, render_template, request, jsonify, send_file
import pandas as pd
import numpy as np
import joblib
import json
import os
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import base64
from io import BytesIO

# Set up matplotlib to avoid GUI issues
import matplotlib

matplotlib.use('Agg')

app = Flask(__name__)

# Global variables for model and preprocessor
model = None
preprocessor = None
feature_names = None
metadata = None
shap_available = False


def load_model_and_preprocessor():
    # Loading the trained model and preprocessor
    global model, preprocessor, feature_names, metadata, shap_available

    try:
        # Load metadata
        with open('prepared_data/metadata.json', 'r', encoding='utf-8') as f:
            metadata = json.load(f)
        feature_names = metadata['feature_names']

        # Load preprocessor
        preprocessor = joblib.load('prepared_data/preprocessor.joblib')

```

```
        # Load best model info
        with open('models/best_model_info.json', 'r', encoding='utf-8') as f:
            best_model_info = json.load(f)

        best_model_name = best_model_info['best_model_name']
        model_dir = f"models/{best_model_name.replace(' ', '_').lower()}"

        # Load the model
        if best_model_name == 'Artificial Neural Network':
            import tensorflow as tf
            model = tf.keras.models.load_model(f'{model_dir}/model.h5')
        else:
            model = joblib.load(f'{model_dir}/model.joblib')

        # Check if SHAP analysis is available
        shap_available = os.path.exists('shap_results/shap_summary_bar.png')

        print(f"Model loaded: {best_model_name}")
        print(f"Features: {len(feature_names)}")
        print(f"SHAP available: {shap_available}")

        return True

    except Exception as e:
        print(f"Error loading model: {e}")
        return False


# Load model on startup
if not load_model_and_preprocessor():
    print("Could not load model. Please run the training pipeline first.")


def predict_churn(customer_data):
    # Predicting the churn probability for a customer
    try:
        # Create DataFrame from customer data
        df = pd.DataFrame([customer_data])

        # Preprocess the data
        processed_data = preprocessor.transform(df)
```

```
83
84          # Make prediction
85          if hasattr(model, 'predict_proba'):
86              probability = model.predict_proba(processed_data)[0][1]
87              prediction = int(probability > 0.5)
88          else:
89              # For neural networks
90              probability = model.predict(processed_data, verbose=0)[0][0]
91              prediction = int(probability > 0.5)
92
93          return probability, prediction, processed_data
94
95      except Exception as e:
96          print(f"Prediction error: {e}")
97          import traceback
98          traceback.print_exc()
99          return None, None, None
100
101 def generate_shap_explanation(customer_data, processed_data):
102     # Generating Shap explanations for the prediction
103     try:
104         print("Generating SHAP explanation ..")
105
106         # Guard: need processed_data and model
107         if processed_data is None or model is None:
108             print("Processed data or model missing — skipping SHAP")
109             return {'contributions': [], 'total_positive': 0.0, 'total_negative': 0.0, 'available': False}
110
111         # Choose explainer based on model type
112         if hasattr(model, 'predict_proba'):
113             print("Using TreeExplainer for tree-based model")
114             explainer = shap.TreeExplainer(model)
115             shap_values = explainer.shap_values(processed_data)
116             # If list returned (per-class), pick positive class for binary
117             if isinstance(shap_values, list):
118                 if len(shap_values) == 2:
119                     shap_values = shap_values[1]
120                 else:
121                     shap_values = shap_values[0]
122         else:
123             print("Using KernelExplainer for non-tree model (may be slower)")
```

```
124         explainer = shap.KernelExplainer(model.predict, processed_data)
125         shap_values = explainer.shap_values(processed_data, nsamples=50)
126
127     # Optional expected value
128     expected_value = None
129     if hasattr(explainer, 'expected_value'):
130         try:
131             ev = explainer.expected_value
132             if isinstance(ev, np.ndarray):
133                 if len(ev) == 2:
134                     expected_value = float(ev[1])
135                 elif len(ev) > 0:
136                     expected_value = float(ev[0])
137             else:
138                 expected_value = float(ev)
139         except Exception:
140             expected_value = None
141
142     # Convert to numpy and reduce to 1D for first sample
143     shap_array = np.array(shap_values)
144
145     if shap_array.ndim == 3:
146         # (1, classes, features) -> pick positive class if present
147         if shap_array.shape[1] == 2:
148             shap_array = shap_array[0, 1, :]
149         else:
150             shap_array = shap_array[0]
151     elif shap_array.ndim == 2:
152         shap_array = shap_array[0] if shap_array.shape[0] == 1 else shap_array[0]
153     elif shap_array.ndim == 1:
154         pass
155     else:
156         shap_array = shap_array.reshape(-1)
157
158     shap_array = np.asarray(shap_array)
159     print(f"Processed SHAP array shape: {shap_array.shape}")
160
161     # Determine feature names
162     if 'feature_names' in globals() and feature_names:
163         fnames = feature_names
164     else:
```

```
165             if hasattr(processed_data, 'columns'):
166                 fnames = list(processed_data.columns)
167             else:
168                 fnames = [f'feature_{i}' for i in range(len(shap_array))]
169 
170         contributions = []
171         for i, fname in enumerate(fnames):
172             if i < len(shap_array):
173                 contribution = float(shap_array[i])
174                 contributions.append({
175                     'feature': fname,
176                     'simplified_name': (simplify_feature_name(fname, metadata) if
177     'simplify_feature_name' in globals() else fname),
178                     'contribution': contribution,
179                     'abs_contribution': abs(contribution),
180                     'direction': 'increases' if contribution > 0 else 'decreases'
181                 })
182 
183         # Sort by absolute contribution descending
184         contributions.sort(key=lambda x: x['abs_contribution'], reverse=True)
185 
186         total_positive = sum([c['contribution'] for c in contributions if c['contribution'] > 0])
187         total_negative = sum([c['contribution'] for c in contributions if c['contribution'] < 0])
188 
189         print(f"Total positive contributions: {total_positive}")
190         print(f"Total negative contributions: {total_negative}")
191         print(f"SHAP contributions computed: {len(contributions)} features")
192 
193         return {
194             'contributions': contributions[:10],  # top 10
195             'total_positive': float(total_positive),
196             'total_negative': float(total_negative),
197             'available': True
198         }
199 
200     except Exception as e:
201         print(f"SHAP explanation error (numeric): {e}")
202         import traceback
203         traceback.print_exc()
204         return {
205             'contributions': [],
```

```
            'available': False,
            'error': str(e)
        }

def simplify_feature_name(feature_name, metadata):
    # Simplify feature names for display
    # Map original names to display names
    display_names = {
        'Ages': 'Age',
        'ServicesOpted': 'Services Opted',
        'FrequentFlyer_No': 'Not a Frequent Flyer (vs Yes)',
        'FrequentFlyer_No Record': 'No Frequent Flyer Record (vs Yes)',
        'FrequentFlyer_Yes': 'Frequent Flyer',
        'AnnualIncomeClass_High Income': 'High Income (vs Low Income)',
        'AnnualIncomeClass_Middle Income': 'Middle Income (vs Low Income)',
        'AccountSyncedToSocialMedia_Yes': 'Social Media Connected (vs Not Connected)',
        'BookedHotelOrNot_Yes': 'Booked Hotel (vs Not Booked)'
    }

    # Return display name if available
    if feature_name in display_names:
        return display_names[feature_name]

    # Handle categorical features
    if '_' in feature_name:
        parts = feature_name.split('_')
        if len(parts) > 1:
            base_feature = parts[0]
            value = '_'.join(parts[1:])

            # Check if we have category info
            if 'category_info' in metadata and base_feature in metadata['category_info']:
                ref_category = metadata['category_info'][base_feature]['reference_category']

                # Convert base feature to display name
                base_display = {
                    'FrequentFlyer': 'Frequent Flyer',
                    'AnnualIncomeClass': 'Income Class',
                    'AccountSyncedToSocialMedia': 'Social Media Sync',
                    'BookedHotelOrNot': 'Hotel Booking'
                }.get(base_feature, base_feature)
```

```
247
248                 return f"{base_display}: {value} (vs {ref_category})"
249
250         return feature_name
251
252
253     def generate_recommendations(probability, customer_data, top_risk_factors):
254         # Generating personalized recommendations based on prediction
255         risk_level = "LOW"
256         risk_color = "#28a745"
257
258         if probability >= 0.8:
259             risk_level = "VERY HIGH"
260             risk_color = "#dc3545"
261         elif probability >= 0.6:
262             risk_level = "HIGH"
263             risk_color = "#fd7e14"
264         elif probability >= 0.4:
265             risk_level = "MEDIUM"
266             risk_color = "#ffc107"
267
268         # Base recommendation
269         if probability >= 0.6:
270             recommendation = "Immediate action required. Contact customer within 24 hours."
271         elif probability >= 0.4:
272             recommendation = "Proactive outreach recommended. Schedule call within 48 hours."
273         else:
274             recommendation = "Monitor customer. Include in next retention campaign."
275
276         # Generate specific recommendations based on customer data
277         specific_recommendations = []
278
279         # Extract customer data
280         age = customer_data.get('Ages')
281         income = customer_data.get('AnnualIncomeClass')
282         services = customer_data.get('ServicesOpted')
283         frequent_flyer = customer_data.get('FrequentFlyer')
284         social_sync = customer_data.get('AccountSyncedToSocialMedia')
285         booked_hotel = customer_data.get('BookedHotelOrNot')
286
287         # Age-based recommendations
```

```
    if age and age < 30:
        specific_recommendations.append("Target with youth-focused marketing campaigns
and mobile app promotions")
    elif age and age > 50:
        specific_recommendations.append("Offer senior discounts, personalized service, and
easy-to-use interfaces")

    # Income-based recommendations
    if income == "Low Income":
        specific_recommendations.append("Provide budget-friendly service bundles and
flexible payment options")
    elif income == "Middle Income":
        specific_recommendations.append("Offer family packages and value-added services")
    elif income == "High Income":
        specific_recommendations.append("Provide premium concierge services and exclusive
benefits")

    # Services-based recommendations
    if services and services <= 2:
        specific_recommendations.append("Upsell complementary services with 30-day free
trial")
    elif services and services >= 5:
        specific_recommendations.append("Create custom bundle discount for loyalty and
service optimization review")

    # Frequent Flyer recommendations
    if frequent_flyer == "Yes":
        specific_recommendations.append("Reward with frequent flyer loyalty points and travel
upgrades")
    elif frequent_flyer == "No Record":
        specific_recommendations.append("Complete travel profile for personalized offers and
travel preferences survey")
    elif frequent_flyer == "No":
        specific_recommendations.append("Introduce travel benefits program with sign-up
bonus")

    # Social media recommendations
    if social_sync == "Yes":
        specific_recommendations.append("Engage through social media exclusive offers and
referral program")
    else:
```

```
329         specific_recommendations.append("Encourage social media connection for exclusive
330 deals")
331
332     # Hotel booking recommendations
333     if booked_hotel == "Yes":
334         specific_recommendations.append("Offer hotel loyalty rewards and room upgrade
335 opportunities")
336     else:
337         specific_recommendations.append("Promote hotel booking packages with special
338 discounts")
339
340     # Add SHAP-based recommendations from top risk factors
341     for factor in top_risk_factors[:3]:
342         feature = factor['simplified_name'].lower()
343         if 'age' in feature and 'young' not in specific_recommendations[0].lower():
344             specific_recommendations.append("Consider age-appropriate engagement
345 strategies")
346         elif 'income' in feature and 'low' in feature:
347             specific_recommendations.append("Review pricing strategy for affordability")
348         elif 'service' in feature:
349             specific_recommendations.append("Conduct service satisfaction survey")
350
351     return {
352         'risk_level': risk_level,
353         'risk_color': risk_color,
354         'recommendation': recommendation,
355         'specific_recommendations': specific_recommendations[:5]  # Top 5
356     }
357
358
359 def load_model_info():
360     # Load model information for display
361     try:
362         with open('models/best_model_info.json', 'r', encoding='utf-8') as f:
363             best_model_info = json.load(f)
364
365         # Load model metrics
366         best_model_name = best_model_info['best_model_name']
367         model_dir = f"models/{best_model_name.replace(' ', '_').lower()}"
368
369         metrics_path = f'{model_dir}/metrics.json'
```

```
370         with open(metrics_path, 'r', encoding='utf-8') as f:
371             metrics = json.load(f)
372 
373         return {
374             'best_model': best_model_name,
375             'best_metrics': metrics,
376             'top_features': best_model_info.get('best_model_top_features', []),
377             'loaded': True
378         }
379     except Exception as e:
380         print(f" Error loading model info: {e}")
381         return {
382             'loaded': False,
383             'error': str(e)
384         }
385 
386 
387 @app.route('/')
388 def index():
389     # Rendering the default page
390     model_info = load_model_info() if model is not None else {'loaded': False}
391 
392     return render_template('index.html',
393                            model_loaded=model is not None,
394                            model_info=model_info,
395                            shap_available=shap_available)
396 
397 
398 @app.route('/predict', methods=['POST'])
399 def predict():
400     # Handling prediction requests
401     try:
402         # Get data from frontend (with correct column names)
403         customer_data = request.json
404 
405         # Validate input - using CSV column names
406         required_fields = ['Ages', 'FrequentFlyer', 'AnnualIncomeClass',
407                            'ServicesOpted', 'AccountSyncedToSocialMedia', 'BookedHotelOrNot']
408 
409         for field in required_fields:
410             if field not in customer_data:
```

```
411                 return jsonify({
412                     'success': False,
413                     'error': f'Missing field: {field}'
414                 }), 400
415 
416         # Convert to proper types
417         customer_data['Ages'] = int(customer_data['Ages'])
418         customer_data['ServicesOpted'] = int(customer_data['ServicesOpted'])
419 
420         # Make prediction
421         probability, prediction, processed_data = predict_churn(customer_data)
422 
423         if probability is None:
424             return jsonify({
425                 'success': False,
426                 'error': 'Prediction failed'
427             }), 500
428 
429         # Generate SHAP explanation
430         shap_explanation = generate_shap_explanation(customer_data, processed_data)
431 
432         # Get top risk factors from SHAP
433         top_risk_factors = shap_explanation.get('contributions', [])
434 
435         # Generate recommendations
436         recommendations = generate_recommendations(probability, customer_data,
437     top_risk_factors)
438 
439         # Load model info
440         model_info_data = load_model_info()
441 
442         # Return response with user-friendly field names for display
443         response = {
444             'success': True,
445             'timestamp': datetime.now().isoformat(),
446             'customer': {
447                 'age': customer_data['Ages'],
448                 'frequent_flyer': customer_data['FrequentFlyer'],
449                 'income_class': customer_data['AnnualIncomeClass'],
450                 'services_opted': customer_data['ServicesOpted'],
451                 'social_sync': customer_data['AccountSyncedToSocialMedia'],
```

```
452             'booked_hotel': customer_data['BookedHotelOrNot']
453         },
454         'prediction': {
455             'probability': float(probability),
456             'predicted_class': int(prediction),
457             'risk_level': recommendations['risk_level'],
458             'risk_color': recommendations['risk_color'],
459             'recommendation': recommendations['recommendation']
460         },
461         'recommendations': recommendations['specific_recommendations'],
462         'shap': shap_explanation,
463         'model_info': {
464             'name': model_info_data.get('best_model', 'Unknown'),
465             'accuracy': model_info_data.get('best_metrics', {}).get('accuracy', 0)
466         }
467     }
468
469     return jsonify(response)
470
471 except Exception as e:
472     print(f" Error in predict endpoint: {e}")
473     import traceback
474     traceback.print_exc()
475     return jsonify({
476         'success': False,
477         'error': str(e)
478     }), 500
479
480
481 @app.route('/model/info')
482 def model_info():
483     # Return model information
484     model_info_data = load_model_info()
485
486     return jsonify({
487         'success': model_info_data['loaded'],
488         'loaded': model is not None,
489         'model_info': model_info_data,
490         'shap_available': shap_available
491     })
492
```

```
493
494 @app.route('/health')
495 def health():
496     # Health check endpoint
497     return jsonify({
498         'status': 'healthy',
499         'model_loaded': model is not None,
500         'timestamp': datetime.now().isoformat()
501     })
502
503
504 if __name__ == '__main__':
505     # Create necessary directories
506     os.makedirs('static/css', exist_ok=True)
507     os.makedirs('static/js', exist_ok=True)
508     os.makedirs('templates', exist_ok=True)
509
510     # Run the app
511     print("Starting Flask application...")
512     print(f"Model loaded: {model is not None}")
513     print(f"SHAP available: {shap_available}")
514     print("Server running at http://localhost:5000")
515
516     app.run(debug=True, host='127.0.0.1', port=5000,use_reloader=False)
517
```

## 10.6 Index HTML (Form) Codes

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Customer Churn Prediction Dashboard</title>

  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">

  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">

  <!-- Custom CSS -->
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
  <link rel="stylesheet" href="{{ url_for('static', filename='css/dashboard.css') }}">

  <style>
    :root {
      --circle-color: #28a745;
      --percentage: 0;
    }

    .probability-circle {
      width: 200px;
      height: 200px;
      border-radius: 50%;
      display: flex;
      align-items: center;
      justify-content: center;
      margin: 0 auto;
      position: relative;
      background: conic-gradient(
        var(--circle-color) calc(var(--percentage) * 1%),
        #eee 0%
      );
    }

    .probability-circle::before {
```

```
            content: '';
            position: absolute;
            width: 180px;
            height: 180px;
            background: white;
            border-radius: 50%;
        }

        .probability-circle span {
            position: relative;
            z-index: 1;
            font-size: 2.5rem;
            font-weight: bold;
            color: #333;
        }

        .insight-icon.bg-secondary {
            background-color: #6c757d !important;
        }
        .insight-icon.bg-light {
            background-color: #f8f9fa !important;
            color: #212529 !important;
        }
        .insight-icon.bg-purple {
            background-color: #6f42c1 !important;
        }
        .insight-icon.bg-pink {
            background-color: #e83e8c !important;
        }
        .insight-icon.bg-orange {
            background-color: #fd7e14 !important;
        }
    </style>
</head>
<body>
    <!-- Loading Overlay -->
    <div class="loading-overlay" id="loadingOverlay">
        <div class="spinner"></div>
        <div class="mt-3 fs-5" id="loadingText">Processing your request...</div>
    </div>

```

```
<div class="container">
  <!-- Header -->
  <div class="dashboard-header">
    <div class="row align-items-center">
      <div class="col-md-8">
        <h1 class="header-title">
          <i class="fas fa-chart-line text-white me-2"></i>
          Customer Churn Prediction
        </h1>
        <p class="header-subtitle">
          Predict customer churn probability using advanced machine learning
          <span class="model-status ms-2" id="modelStatus">
            <i class="fas fa-circle"></i>
            <span id="modelStatusText">Checking model...</span>
          </span>
        </p>
      </div>
      <div class="col-md-4 text-end">
        <button class="btn btn-outline-light me-2" id="modelInfoBtn">
          <i class="fas fa-info-circle"></i> Model Info
        </button>
      </div>
    </div>
  </div>

  <div class="row mt-4">
    <!-- Left Column: Input Form -->
    <div class="col-lg-4">
      <div class="card">
        <div class="card-header">
          <i class="fas fa-user-edit me-2"></i> Customer Information
        </div>
        <div class="card-body">
          <form id="predictionForm">
            <div class="row">
              <!-- Age -->
              <div class="col-md-12 mb-3">
                <label for="age" class="form-label">
                  <i class="fas fa-birthday-cake me-1"></i> Age
                </label>
                <input type="number" class="form-control" id="age"
```

```
                min="18" max="100" value="34" required>
            <div class="form-text">Customer's age (18-100)</div>
        </div>

        <!-- Frequent Flyer -->
        <div class="col-md-12 mb-3">
            <label for="frequent_flyer" class="form-label">
                <i class="fas fa-plane me-1"></i> Frequent Flyer
            </label>
            <select class="form-select" id="frequent_flyer" required>
                <option value="Yes">Yes</option>
                <option value="No" selected>No</option>
                <option value="No Record">No Record</option>
            </select>
            <div class="form-text">Select Yes, No, or No Record</div>
        </div>

        <!-- Income Class -->
        <div class="col-md-12 mb-3">
            <label for="income_class" class="form-label">
                <i class="fas fa-money-bill-wave me-1"></i> Income Class
            </label>
            <select class="form-select" id="income_class" required>
                <option value="Low Income">Low Income</option>
                <option value="Middle Income" selected>Middle Income</option>
                <option value="High Income">High Income</option>
            </select>
        </div>

        <!-- Services Opted -->
        <div class="col-md-12 mb-3">
            <label for="services_opted" class="form-label">
                <i class="fas fa-concierge-bell me-1"></i> Services Opted
            </label>
            <input type="number" class="form-control" id="services_opted"
                min="1" max="6" value="4" required>
            <div class="form-text">Number of services (1-6)</div>
        </div>

        <!-- Social Media Sync -->
        <div class="col-md-6 mb-3">
```

```
                        <label for="social_sync" class="form-label">
                            <i class="fas fa-share-alt me-1"></i> Social Media Sync
                        </label>
                        <select class="form-select" id="social_sync" required>
                            <option value="Yes">Yes</option>
                            <option value="No" selected>No</option>
                        </select>
                    </div>

                    <!-- Booked Hotel -->
                    <div class="col-md-6 mb-3">
                        <label for="booked_hotel" class="form-label">
                            <i class="fas fa-hotel me-1"></i> Booked Hotel
                        </label>
                        <select class="form-select" id="booked_hotel" required>
                            <option value="Yes">Yes</option>
                            <option value="No" selected>No</option>
                        </select>
                    </div>
                </div>

                <div class="d-grid gap-2">
                    <button type="button" class="btn btn-primary" id="predictButton">
                        <i class="fas fa-brain me-2"></i> Predict Churn Probability
                    </button>

                    <button type="button" class="btn btn-outline-secondary"
id="loadSampleBtn">
                        <i class="fas fa-magic me-2"></i> Load Sample Customer
                    </button>
                </div>
            </form>
        </div>
    </div>

    <!-- Model Stats -->
    <div class="card">
        <div class="card-header">
            <i class="fas fa-chart-bar me-2"></i> Model Performance
        </div>
        <div class="card-body">
```

```
206                 {% if model_loaded and model_info.loaded %}
207                 <div class="row text-center">
208                     <div class="col-6 mb-3">
209                         <div class="stats-value text-primary">
210                             {{ "%.1f"|format(model_info.best_metrics.accuracy * 100) }}%
211                         </div>
212                         <div class="stats-label">Accuracy</div>
213                     </div>
214                     <div class="col-6 mb-3">
215                         <div class="stats-value text-success">
216                             {{ "%.1f"|format(model_info.best_metrics.f1 * 100) }}%
217                         </div>
218                         <div class="stats-label">F1 Score</div>
219                     </div>
220                     <div class="col-6">
221                         <div class="stats-value text-warning">
222                             {{ "%.1f"|format(model_info.best_metrics.precision * 100) }}%
223                         </div>
224                         <div class="stats-label">Precision</div>
225                     </div>
226                     <div class="col-6">
227                         <div class="stats-value text-info">
228                             {{ "%.1f"|format(model_info.best_metrics.recall * 100) }}%
229                         </div>
230                         <div class="stats-label">Recall</div>
231                     </div>
232                 </div>
233                 <div class="text-center mt-3">
234                     <small class="text-muted">Model: {{ model_info.best_model }}</small>
235                 </div>
236                 {% else %}
237                 <div class="text-center py-4">
238                     <i class="fas fa-exclamation-triangle fa-3x text-warning mb-3"></i>
239                     <p class="text-muted mb-0">Model not trained yet</p>
240                     <small>Run the training pipeline first</small>
241                 </div>
242                 {% endif %}
243             </div>
244         </div>
245
246     </div>
```

```
247
248         <!-- Right Column: Results -->
249         <div class="col-lg-8">
250           <!-- Welcome Card (shown by default) -->
251           <div class="card" id="welcomeCard">
252             <div class="card-header">
253               <i class="fas fa-rocket me-2"></i> Welcome to Churn Predictor
254             </div>
255             <div class="card-body text-center py-5">
256               <div class="display-1 mb-4">📊</div>
257               <h2 class="mb-3">Ready to Predict Customer Churn</h2>
258               <p class="text-muted mb-4 lead">
259                 Enter customer details in the form and click "Predict Churn Probability"
260                 to get instant risk assessment with actionable insights.
261               </p>
262
263               <div class="row mt-5 text-start">
264                 <div class="col-md-6 mb-3">
265                   <div class="d-flex">
266                     <div class="me-3">
267                       <i class="fas fa-bolt text-warning fa-2x"></i>
268                     </div>
269                     <div>
270                       <h6>Real-time Predictions</h6>
271                       <p class="text-muted small">Get instant churn probability scores</p>
272                     </div>
273                   </div>
274                 </div>
275                 <div class="col-md-6 mb-3">
276                   <div class="d-flex">
277                     <div class="me-3">
278                       <i class="fas fa-chart-bar text-success fa-2x"></i>
279                     </div>
280                     <div>
281                       <h6>Feature Analysis</h6>
282                       <p class="text-muted small">Understand key factors influencing
283 predictions</p>
284                     </div>
285                   </div>
286                 </div>
287                 <div class="col-md-6 mb-3">
```

```
                        <div class="d-flex">
                            <div class="me-3">
                                <i class="fas fa-lightbulb text-info fa-2x"></i>
                            </div>
                            <div>
                                <h6>Actionable Insights</h6>
                                <p class="text-muted small">Get recommendations for customer
retention</p>
                            </div>
                        </div>
                    </div>
                </div>

                <div class="mt-4">
                    <div class="alert alert-secondary">
                        <i class="fas fa-info-circle me-2"></i>
                        <strong>How to use:</strong>
                        <ol class="mb-0 mt-2">
                            <li>Fill in customer details or click "Load Sample Customer"</li>
                            <li>Click "Predict Churn Probability" button</li>
                            <li>View results and insights</li>
                            <li><strong>Note:</strong> Frequent Flyer has three options: Yes, No,
and No Record</li>
                        </ol>
                    </div>
                </div>
            </div>
        </div>

        <!-- Results Card (hidden by default) -->
        <div class="card result-card" id="resultsCard" style="display: none;">
            <div class="card-header">
                <i class="fas fa-chart-pie me-2"></i> Prediction Results
                <span class="float-end">
                    <button class="btn btn-sm btn-outline-secondary" id="newPredictionBtn">
                        <i class="fas fa-redo"></i> New Prediction
                    </button>
                </span>
            </div>
            <div class="card-body">
                <!-- Probability Display -->
```

```
                <div class="row align-items-center mb-4">
                  <div class="col-md-5 text-center">
                    <div class="probability-circle" id="probabilityCircle">
                      <span id="probabilityValue">0%</span>
                    </div>
                    <div class="mt-3">
                      <small class="text-muted">Churn Probability</small>
                    </div>
                  </div>
                  <div class="col-md-7">
                    <h3 class="mb-3">Churn Risk Assessment</h3>
                    <div class="mb-3">
                      <div class="risk-badge mb-2" id="riskBadge">LOW RISK</div>
                      <p class="text-muted mt-2" id="riskDescription">
                        Probability of customer churning
                      </p>
                    </div>
                    <div class="alert" id="recommendationAlert">
                      <i class="fas fa-lightbulb me-2"></i>
                      <span id="recommendationText">Recommendation will appear
here</span>
                    </div>

                    <!-- Prediction Details -->
                    <div class="mt-4">
                      <h6><i class="fas fa-info-circle me-2"></i> Prediction Details</h6>
                      <div class="row small">
                        <div class="col-6">
                          <span class="text-muted">Model:</span>
                          <span id="modelName">-</span>
                        </div>
                        <div class="col-6">
                          <span class="text-muted">Confidence:</span>
                          <span id="confidenceLevel">-</span>
                        </div>
                        <div class="col-6 mt-2">
                          <span class="text-muted">Prediction Time:</span>
                          <span id="predictionTime">-</span>
                        </div>
                      </div>
                    </div>
```

```
370                 </div>
371               </div>
372
373               <!-- Customer Summary -->
374               <div class="row mb-4">
375                 <div class="col-12">
376                   <h5 class="mb-3">
377                     <i class="fas fa-user-circle me-2"></i> Customer Summary
378                   </h5>
379                   <div class="table-responsive">
380                     <table class="table table-hover table-sm">
381                       <thead>
382                         <tr class="table-primary">
383                           <th>Age</th>
384                           <th>Frequent Flyer</th>
385                           <th>Income Class</th>
386                           <th>Services</th>
387                           <th>Social Sync</th>
388                           <th>Booked Hotel</th>
389                         </tr>
390                       </thead>
391                       <tbody>
392                         <tr>
393                           <td id="summaryAge">-</td>
394                           <td id="summaryFlyer">-</td>
395                           <td id="summaryIncome">-</td>
396                           <td id="summaryServices">-</td>
397                           <td id="summarySocial">-</td>
398                           <td id="summaryHotel">-</td>
399                         </tr>
400                       </tbody>
401                     </table>
402                   </div>
403                 </div>
404               </div>
405
406               <!-- RISK FACTORS SUMMARY SECTION -->
407               <div class="row mb-4" id="riskFactorsSummary" style="display: none;">
408                 <div class="col-12">
409                   <h5 class="mb-3">
410                     <i class="fas fa-exclamation-triangle me-2"></i> Risk Factors Summary
```

```
411                     </h5>
412                     <div class="row">
413                       <div class="col-md-6">
414                         <div class="card h-100 border-danger">
415                           <div class="card-header bg-danger text-white">
416                             <i class="fas fa-arrow-up me-2"></i> Factors Increasing Churn
417 Risk
418                           </div>
419                           <div class="card-body">
420                             <div id="increasingFactorsList">
421                               <div class="text-center py-3">
422                                 <i class="fas fa-spinner fa-spin text-muted"></i>
423                                 <p class="text-muted mt-2">Analyzing risk factors...</p>
424                               </div>
425                             </div>
426                             <div class="mt-3 small text-muted">
427                               <i class="fas fa-info-circle me-1"></i>
428                               These features increase the probability of customer churn
429                             </div>
430                           </div>
431                         </div>
432                       </div>
433                       <div class="col-md-6">
434                         <div class="card h-100 border-success">
435                           <div class="card-header bg-success text-white">
436                             <i class="fas fa-arrow-down me-2"></i> Factors Decreasing Churn
437 Risk
438                           </div>
439                           <div class="card-body">
440                             <div id="decreasingFactorsList">
441                               <div class="text-center py-3">
442                                 <i class="fas fa-spinner fa-spin text-muted"></i>
443                                 <p class="text-muted mt-2">Analyzing protective
444 factors...</p>
445                               </div>
446                             </div>
447                             <div class="mt-3 small text-muted">
448                               <i class="fas fa-info-circle me-1"></i>
449                               These features decrease the probability of customer churn
450                             </div>
451                           </div>
```

```
452                         </div>
453                       </div>
454                     </div>
455
456                     <!-- Key Takeaways -->
457                     <div class="row mt-4">
458                       <div class="col-12">
459                         <div class="card border-info">
460                           <div class="card-header bg-info text-white">
461                             <i class="fas fa-lightbulb me-2"></i> Key Takeaways
462                           </div>
463                           <div class="card-body">
464                             <div id="keyTakeaways">
465                               <p class="mb-2">Based on the customer profile and model
466 analysis:</p>
467                               <ul id="takeawaysList" class="mb-0">
468                                 <li>Analyzing customer risk profile...</li>
469                               </ul>
470                             </div>
471                           </div>
472                         </div>
473                       </div>
474                     </div>
475                   </div>
476                 </div>
477
478                 <!-- Insights -->
479                 <div class="row" id="insightsSection">
480                   <div class="col-12">
481                     <h5 class="mb-3">
482                       <i class="fas fa-lightbulb me-2"></i> Actionable Insights
483                     </h5>
484                     <div id="insightsContainer">
485                       <div class="text-center py-4">
486                         <i class="fas fa-spinner fa-spin fa-2x text-muted"></i>
487                         <p class="text-muted mt-2">Generating insights...</p>
488                       </div>
489                     </div>
490                   </div>
491                 </div>
492
```

```
493                 <!-- Additional Actions (New Customer & Print Report removed per request) -->
494                 <div class="row mt-4">
495                   <div class="col-12">
496                     <div class="d-flex justify-content-end">
497                       <button class="btn btn-outline-info" id="shareReportBtn">
498                         <i class="fas fa-share me-2"></i> Share Report
499                       </button>
500                     </div>
501                   </div>
502                 </div>
503               </div>
504             </div>
505           </div>
506         </div>
507
508         <!-- Footer -->
509         <div class="footer">
510           <p class="mb-0">
511             Customer Churn Prediction System
512           </p>
513           <small class="opacity-75"></small>
514           <div class="mt-2">
515             <small class="text-white-50">
516               <i class="fas fa-server me-1"></i> Server: localhost:5000 |
517               <i class="fas fa-code me-1 ms-2"></i> Flask
518             </small>
519           </div>
520         </div>
521       </div>
522
523       <!-- Bootstrap JS -->
524       <script
525     src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></scrip
526     t>
527
528       <!-- Custom JavaScript -->
529       <script src="{{ url_for('static', filename='js/app.js') }}"></script>
530
531       <script>
532         // Initialize after everything is loaded
533         window.addEventListener('load', function() {
```

```
534         console.log('Window fully loaded, initializing app...');
535
536         // Add keyboard shortcuts
537         document.addEventListener('keydown', function(event) {
538           // Ctrl+Enter to submit form
539           if (event.ctrlKey && event.key === 'Enter') {
540             event.preventDefault();
541             if (typeof handleFormSubmit === 'function') {
542               handleFormSubmit(event);
543             }
544           }
545           // F5 to load sample data
546           if (event.key === 'F5') {
547             event.preventDefault();
548             if (typeof loadSampleData === 'function') {
549               loadSampleData();
550             }
551           }
552         });
553
554         // Initial console message
555         console.log('Customer Churn Prediction System loaded');
556         console.log('Note: Frequent Flyer has three options: Yes, No, and No Record');
557       });
558     </script>
559 </body>
560 </html>
561
```

## 10.7 App.js Codes

```
console.log('Customer Churn Predictor JavaScript loaded');

// Global state
let currentPrediction = null;

// DOM Elements
const predictButton = document.getElementById('predictButton');
const loadSampleBtn = document.getElementById('loadSampleBtn');
const resultsCard = document.getElementById('resultsCard');
const welcomeCard = document.getElementById('welcomeCard');
const loadingOverlay = document.getElementById('loadingOverlay');
const loadingText = document.getElementById('loadingText');

// Initialize the application
document.addEventListener('DOMContentLoaded', function() {
    console.log('DOM fully loaded and parsed');
    setupEventListeners();
    checkModelStatus();
});

function setupEventListeners() {
    console.log('Setting up event listeners...');

    // Predict button
    if (predictButton) {
        console.log('Found predict button');
        predictButton.addEventListener('click', handleFormSubmit);
    } else {
        console.error('Predict button not found! Looking for #predictButton');
    }

    // Load sample data button
    if (loadSampleBtn) {
        loadSampleBtn.addEventListener('click', loadSampleData);
    }

    // Show model info button
    const modelInfoBtn = document.getElementById('modelInfoBtn');
    if (modelInfoBtn) {
        modelInfoBtn.addEventListener('click', showModelInfo);
    }
```

```
42
43      // Results card buttons
44      const newPredictionBtn = document.getElementById('newPredictionBtn');
45      if (newPredictionBtn) {
46          newPredictionBtn.addEventListener('click', resetForm);
47      }
48
49      // Prevent form submission on Enter key
50      const predictionForm = document.getElementById('predictionForm');
51      if (predictionForm) {
52          predictionForm.addEventListener('submit', function(event) {
53              event.preventDefault();
54              console.log('Form submit prevented');
55              return false;
56          });
57
58          // Also prevent any form submissions
59          predictionForm.onsubmit = function() {
60              console.log('Form onsubmit prevented');
61              return false;
62          };
63      }
64  }
65
66  async function checkModelStatus() {
67      console.log('Checking model status...');
68      try {
69          const response = await fetch('/model/info');
70          console.log('Model info response status:', response.status);
71
72          if (!response.ok) {
73              throw new Error(`HTTP error! status: ${response.status}`);
74          }
75
76          const data = await response.json();
77          console.log('Model info data:', data);
78
79          if (data.success) {
80              updateModelStatus(true, data.model_info.best_model, data.shap_available);
81          } else {
82              updateModelStatus(false, 'Not loaded - ' + (data.error || 'Unknown error'), false);
```

```
        }
    } catch (error) {
        console.error('Failed to check model status:', error);
        updateModelStatus(false, 'Connection error', false);
    }
}

function updateModelStatus(isLoaded, modelName, shapAvailable) {
    const statusElement = document.getElementById('modelStatus');
    const statusText = document.getElementById('modelStatusText');

    console.log(`Updating model status: loaded=${isLoaded}, name=${modelName}`);

    if (statusElement && statusText) {
        if (isLoaded) {
            statusElement.className = 'model-status model-active';
            statusElement.innerHTML = `<i class="fas fa-check-circle"></i> ${modelName}`;
            statusText.textContent = 'Model Active';

            // Update SHAP badge
            if (shapAvailable) {
                const shapBadge = document.getElementById('shapBadge');
                if (shapBadge) {
                    shapBadge.innerHTML = '<i class="fas fa-brain"></i> SHAP Enabled';
                    shapBadge.className = 'badge bg-info ms-2';
                }
            }
        } else {
            statusElement.className = 'model-status model-inactive';
            statusElement.innerHTML = `<i class="fas fa-exclamation-triangle"></i>
${modelName}`;
            statusText.textContent = modelName || 'Model Not Ready';
        }
    }
}

async function handleFormSubmit(event) {
    console.log('handleFormSubmit called');

    if (event) {
        event.preventDefault();
```

```
124         event.stopPropagation();
125     }
126
127     // Collect form data with CORRECT CSV column names
128     const formData = {
129         Ages: document.getElementById('age').value,
130         FrequentFlyer: document.getElementById('frequent_flyer').value,
131         AnnualIncomeClass: document.getElementById('income_class').value,
132         ServicesOpted: document.getElementById('services_opted').value,
133         AccountSyncedToSocialMedia: document.getElementById('social_sync').value,
134         BookedHotelOrNot: document.getElementById('booked_hotel').value
135     };
136
137     console.log('Form data:', formData);
138
139     // Validate form data
140     if (!validateFormData(formData)) {
141         showNotification('Please fill in all fields correctly', 'warning');
142         return false;
143     }
144
145     // Convert numeric fields
146     formData.Ages = parseInt(formData.Ages);
147     formData.ServicesOpted = parseInt(formData.ServicesOpted);
148
149     // Show loading
150     showLoading('Predicting churn probability...');
151
152     try {
153         console.log('Sending request to /predict...');
154
155         // First check if server is reachable
156         const healthCheck = await fetch('/health');
157         if (!healthCheck.ok) {
158             throw new Error('Server is not responding');
159         }
160
161         // Send prediction request
162         const response = await fetch('/predict', {
163             method: 'POST',
164             headers: {
```

```
165             'Content-Type': 'application/json',
166           },
167           body: JSON.stringify(formData)
168         });
169
170         console.log('Response received, status:', response.status);
171
172         if (!response.ok) {
173           const errorText = await response.text();
174           throw new Error(`Server error: ${response.status} - ${errorText}`);
175         }
176
177         const result = await response.json();
178         console.log('Prediction result:', result);
179
180         if (result.success) {
181           // Store prediction
182           currentPrediction = result;
183
184           // Update UI with results
185           updateResultsUI(result);
186
187           // Show results card
188           showResultsCard();
189
190           showNotification('Prediction completed successfully!', 'success');
191         } else {
192           showNotification(`Prediction failed: ${result.error}`, 'danger');
193           console.error('Prediction error:', result.error);
194         }
195
196       } catch (error) {
197         console.error('Prediction error:', error);
198         showNotification(`Failed: ${error.message}`, 'danger');
199       } finally {
200         hideLoading();
201       }
202
203       return false;
204     }
205
```

```
206  function validateFormData(data) {
207      console.log('Validating form data:', data);
208
209      // Check all fields are filled
210      for (const [key, value] of Object.entries(data)) {
211          if (!value || value.toString().trim() === '') {
212              console.log(`Validation failed: ${key} is empty`);
213              return false;
214          }
215      }
216
217      // Validate age (Ages)
218      const age = parseInt(data.Ages);
219      if (isNaN(age) || age < 18 || age > 100) {
220          console.log('Validation failed: Ages must be between 18 and 100');
221          return false;
222      }
223
224      // Validate services (ServicesOpted)
225      const services = parseInt(data.ServicesOpted);
226      if (isNaN(services) || services < 1 || services > 6) {
227          console.log('Validation failed: ServicesOpted must be between 1 and 6');
228          return false;
229      }
230
231      // Validate Frequent Flyer (must be one of three options)
232      const validFrequentFlyer = ['Yes', 'No', 'No Record'];
233      if (!validFrequentFlyer.includes(data.FrequentFlyer)) {
234          console.log('Validation failed: FrequentFlyer must be Yes, No, or No Record');
235          return false;
236      }
237
238      return true;
239  }
240
241  function showLoading(message) {
242      if (loadingOverlay && loadingText) {
243          loadingText.textContent = message || 'Processing...';
244          loadingOverlay.style.display = 'flex';
245      }
246  }
```

```
247
248 function hideLoading() {
249     if (loadingOverlay) {
250         loadingOverlay.style.display = 'none';
251     }
252 }
253
254 function showNotification(message, type) {
255     // Remove any existing notifications
256     const existingNotifications = document.querySelectorAll('.notification-toast');
257     existingNotifications.forEach(notification => notification.remove());
258
259     // Create notification element
260     const notification = document.createElement('div');
261     notification.className = `notification-toast alert alert-${type} alert-dismissible fade show`;
262     notification.innerHTML = `
263         ${message}
264         <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
265 label="Close"></button>
266     `;
267
268     // Style the notification
269     notification.style.position = 'fixed';
270     notification.style.top = '20px';
271     notification.style.right = '20px';
272     notification.style.zIndex = '9999';
273     notification.style.minWidth = '300px';
274     notification.style.maxWidth = '500px';
275
276     // Add to page
277     document.body.appendChild(notification);
278
279     // Auto-remove after 5 seconds
280     setTimeout(() => {
281         if (notification.parentNode) {
282             notification.remove();
283         }
284     }, 5000);
285 }
286
287 function updateResultsUI(result) {
```

```
288     // Update probability circle
289     const probability = result.prediction.probability * 100;
290     const probabilityCircle = document.getElementById('probabilityCircle');
291     const probabilityValue = document.getElementById('probabilityValue');
292
293     if (probabilityCircle && probabilityValue) {
294       probabilityCircle.style.setProperty('--percentage', probability);
295       probabilityCircle.style.setProperty('--circle-color', result.prediction.risk_color);
296       probabilityValue.textContent = `${probability.toFixed(1)}%`;
297     }
298
299     // Update risk badge
300     const riskBadge = document.getElementById('riskBadge');
301     if (riskBadge) {
302       riskBadge.textContent = result.prediction.risk_level;
303       riskBadge.className = 'risk-badge mb-2';
304       riskBadge.style.backgroundColor = result.prediction.risk_color;
305       riskBadge.style.color = 'white';
306     }
307
308     // Update recommendation
309     const recommendationAlert = document.getElementById('recommendationAlert');
310     const recommendationText = document.getElementById('recommendationText');
311     if (recommendationAlert && recommendationText) {
312       recommendationAlert.className = `alert alert-
313 ${getAlertType(result.prediction.risk_level)}`;
314       recommendationText.textContent = result.prediction.recommendation;
315     }
316
317     // Update model info
318     document.getElementById('modelName').textContent = result.model_info.name;
319     document.getElementById('confidenceLevel').textContent =
320 `${(result.model_info.accuracy * 100).toFixed(1)}%`;
321     document.getElementById('predictionTime').textContent = new
322 Date(result.timestamp).toLocaleTimeString();
323
324     // Update customer summary
325     document.getElementById('summaryAge').textContent = result.customer.age;
326     document.getElementById('summaryFlyer').textContent = result.customer.frequent_flyer;
327     document.getElementById('summaryIncome').textContent =
328 result.customer.income_class;
```

```
    document.getElementById('summaryServices').textContent =
result.customer.services_opted;
    document.getElementById('summarySocial').textContent = result.customer.social_sync;
    document.getElementById('summaryHotel').textContent = result.customer.booked_hotel;

    // Update actionable insights
    updateRecommendationsList(result.actionable_insights);
    // Update recommendations list
    updateRecommendationsList(result.recommendations);

    // Update risk factors summary:
    // Normalize contributions from various possible response shapes then call
updateRiskFactorsSummary
    try {
        // Normalize and choose the contributions array from server response
        let contributions = [];

        if (result && result.shap && Array.isArray(result.shap.contributions)) {
            contributions = result.shap.contributions;
        }
        else if (result && Array.isArray(result.shap_contributions)) {
            contributions = result.shap_contributions;
        }
        // Structured risk_factors (combine increasing + decreasing into one contributions-like
array)
        else if (result && result.risk_factors && (Array.isArray(result.risk_factors.increasing) ||
Array.isArray(result.risk_factors.decreasing))) {
            const inc = result.risk_factors.increasing || [];
            const dec = result.risk_factors.decreasing || [];
            // Keep sign and abs_contribution consistent
            contributions = inc.concat(dec);
        }
        // Legacy: result.contributions
        else if (result && Array.isArray(result.contributions)) {
            contributions = result.contributions;
        }

        // Ensure each contribution has required fields (simplified_name, contribution,
abs_contribution)
        contributions = contributions.map(c => {
            const contributionVal = (typeof c.contribution === 'number') ? c.contribution : (typeof
```

```
370     c.value === 'number' ? c.value : 0);
371             const absVal = (typeof c.abs_contribution === 'number') ? c.abs_contribution :
372     Math.abs(contributionVal);
373             return {
374                 feature: c.feature || c.name || 'unknown',
375                 simplified_name: c.simplified_name || c.display_name || c.feature || 'unknown',
376                 contribution: contributionVal,
377                 abs_contribution: absVal,
378                 reason: c.reason || c.explanation || ''
379             };
380         });
381       // Update risk factors summary:
382       // Pass the entire contribution so the summary function can handle either 'contributions'
383     or
384       // the server-side 'risk_factors', 'key_takeaways', 'actionable_insights'.
385       updateRiskFactorsSummary(contributions);
386     } catch (err) {
387       console.error("Failed to update risk factors summary:", err);
388     }}
389
390     function getAlertType(riskLevel) {
391       switch(riskLevel) {
392         case 'VERY HIGH': return 'danger';
393         case 'HIGH': return 'warning';
394         case 'MEDIUM': return 'info';
395         case 'LOW': return 'success';
396         default: return 'secondary';
397       }
398     }
399
400     function showResultsCard() {
401       if (welcomeCard) welcomeCard.style.display = 'none';
402       if (resultsCard) resultsCard.style.display = 'block';
403     }
404
405     function loadSampleData() {
406       console.log('Loading sample data...');
407
408       // Sample customer data with CORRECT CSV column names
409       const sampleData = {
410         Ages: 45,
```

```
411         FrequentFlyer: 'Yes',
412         AnnualIncomeClass: 'High Income',
413         ServicesOpted: 5,
414         AccountSyncedToSocialMedia: 'Yes',
415         BookedHotelOrNot: 'No'
416       };
417
418       // Fill form with sample data
419       document.getElementById('age').value = sampleData.Ages;
420       document.getElementById('frequent_flyer').value = sampleData.FrequentFlyer;
421       document.getElementById('income_class').value = sampleData.AnnualIncomeClass;
422       document.getElementById('services_opted').value = sampleData.ServicesOpted;
423       document.getElementById('social_sync').value =
424   sampleData.AccountSyncedToSocialMedia;
425       document.getElementById('booked_hotel').value = sampleData.BookedHotelOrNot;
426
427       showNotification('Sample customer data loaded!', 'info');
428   }
429
430   function showModelInfo() {
431       fetch('/model/info')
432           .then(response => response.json())
433           .then(data => {
434               if (data.success) {
435                   const info = data.model_info;
436                   const metrics = info.best_metrics || {};
437
438                   const message = `
439                       Model: ${info.best_model}
440
441                       Performance Metrics:
442                       • Accuracy: ${(metrics.accuracy * 100).toFixed(1)}%
443                       • F1 Score: ${(metrics.f1 * 100).toFixed(1)}%
444                       • Precision: ${(metrics.precision * 100).toFixed(1)}%
445                       • Recall: ${(metrics.recall * 100).toFixed(1)}%
446
447                       Status: ${data.loaded ? 'Loaded and ready' : 'Not loaded'}
448                       SHAP Available: ${data.shap_available ? 'Yes' : 'No'}
449                   `;
450
451                   alert(message);
```

```
452             } else {
453                 showNotification('Could not load model info: ' + data.error, 'warning');
454             }
455         })
456         .catch(error => {
457             console.error('Error fetching model info:', error);
458             showNotification('Failed to load model info', 'danger');
459         });
460 }
461
462 function updateRecommendationsList(recommendations) {
463     const insightsContainer = document.getElementById('insightsContainer');
464     if (!insightsContainer) return;
465         // Safety check
466     if (!Array.isArray(recommendations)) {
467         insightsContainer.innerHTML = `
468             <div class="text-muted small">No recommendations available.</div>
469         `;
470         return;
471     }
472
473     let html = '<div class="row">';
474
475     recommendations.forEach((rec, index) => {
476         const colors = ['primary', 'success', 'info', 'warning', 'danger'];
477         const color = colors[index % colors.length];
478         //const text = (typeof rec === 'string') ? rec : (rec.text || JSON.stringify(rec));
479
480         html += `
481             <div class="col-md-6 mb-3">
482                 <div class="insight-card">
483                     <div class="insight-icon bg-${color}">
484                         <i class="fas fa-lightbulb"></i>
485                     </div>
486                     <div class="insight-content">
487                         <h6>Recommendation ${index + 1}</h6>
488                         <p class="small mb-0">${rec}</p>
489                     </div>
490                 </div>
491             </div>
492         `;
```

```
493         });
494
495         html += '</div>';
496         insightsContainer.innerHTML = html;
497     }
498
499     // Handling Risk Factors summary update
500     function updateRiskFactorsSummary(contributions) {
501         console.log('Updating risk factors summary...');
502
503         const increasingFactorsContainer = document.getElementById('increasingFactorsList');
504         const decreasingFactorsContainer = document.getElementById('decreasingFactorsList');
505         const takeawaysList = document.getElementById('takeawaysList');
506         const riskFactorsSection = document.getElementById('riskFactorsSummary');
507
508         if (!riskFactorsSection) {
509             console.error('Risk factors section not found');
510             return;
511         }
512
513         // Show the section
514         riskFactorsSection.style.display = 'block';
515
516         // Separate increasing and decreasing factors
517         const increasingFactors = contributions.filter(c => c.contribution > 0);
518         const decreasingFactors = contributions.filter(c => c.contribution < 0);
519
520         // Update increasing factors (risk factors)
521         if (increasingFactorsContainer) {
522             if (increasingFactors.length > 0) {
523                 let html = '<div class="list-group list-group-flush">';
524                 increasingFactors.slice(0, 5).forEach((factor, index) => {
525                     const impactStrength = getImpactStrength(factor.abs_contribution);
526                     html += `
527                         <div class="list-group-item border-0 py-2">
528                             <div class="d-flex justify-content-between align-items-center">
529                                 <div>
530                                     <span class="badge bg-danger me-2">${index + 1}</span>
531                                     <span class="fw-semibold">${factor.simplified_name}</span>
532                                 </div>
533                                 <div>
```

```
                        <span class="text-danger fw-
bold">+${factor.contribution.toFixed(3)}</span>
                        <span class="badge ${getImpactBadgeClass(impactStrength)} ms-2">
                          ${impactStrength}
                        </span>
                      </div>
                    </div>
                    <div class="small text-muted mt-1">
                      <i class="fas fa-info-circle me-1"></i>
                      ${getRiskFactorExplanation(factor.simplified_name, factor.contribution)}
                    </div>
                  </div>
                `;
            });
            html += '</div>';
            increasingFactorsContainer.innerHTML = html;
        } else {
            increasingFactorsContainer.innerHTML = `
                <div class="text-center py-3">
                    <i class="fas fa-check-circle fa-2x text-success mb-2"></i>
                    <p class="text-muted">No significant risk-increasing factors identified</p>
                </div>
            `;
        }
    }

    // Update decreasing factors (protective factors)
    if (decreasingFactorsContainer) {
        if (decreasingFactors.length > 0) {
            let html = '<div class="list-group list-group-flush">';
            decreasingFactors.slice(0, 5).forEach((factor, index) => {
                const impactStrength = getImpactStrength(factor.abs_contribution);
                html += `
                    <div class="list-group-item border-0 py-2">
                        <div class="d-flex justify-content-between align-items-center">
                            <div>
                                <span class="badge bg-success me-2">${index + 1}</span>
                                <span class="fw-semibold">${factor.simplified_name}</span>
                            </div>
                            <div>
                                <span class="text-success fw-
```

```
575 bold">${factor.contribution.toFixed(3)}</span>
576                         <span class="badge ${getImpactBadgeClass(impactStrength)} ms-2">
577                           ${impactStrength}
578                         </span>
579                       </div>
580                     </div>
581                     <div class="small text-muted mt-1">
582                       <i class="fas fa-info-circle me-1"></i>
583                       ${getProtectiveFactorExplanation(factor.simplified_name,
584 factor.contribution)}
585                     </div>
586                   </div>
587                 `;
588               });
589               html += '</div>';
590               decreasingFactorsContainer.innerHTML = html;
591             } else {
592               decreasingFactorsContainer.innerHTML = `
593                 <div class="text-center py-3">
594                   <i class="fas fa-info-circle fa-2x text-warning mb-2"></i>
595                   <p class="text-muted">No significant protective factors identified</p>
596                 </div>
597               `;
598             }
599           }
600
601           // Update key takeaways
602           if (takeawaysList) {
603             let takeaways = generateKeyTakeaways(contributions, increasingFactors,
604 decreasingFactors);
605             takeawaysList.innerHTML = '';
606
607             takeaways.forEach(takeaway => {
608               const li = document.createElement('li');
609               li.className = 'mb-2';
610               li.innerHTML = `
611                 <i class="fas ${takeaway.icon} me-2 text-${takeaway.color}"></i>
612                 ${takeaway.text}
613               `;
614               takeawaysList.appendChild(li);
615             });
```

```
616         }
617     }
618
619     function getImpactStrength(absContribution) {
620         if (absContribution > 0.1) return 'HIGH';
621         if (absContribution > 0.05) return 'MEDIUM';
622         if (absContribution > 0.01) return 'LOW';
623         return 'MINIMAL';
624     }
625
626     function getImpactBadgeClass(impactStrength) {
627         switch(impactStrength) {
628             case 'HIGH': return 'bg-danger';
629             case 'MEDIUM': return 'bg-warning text-dark';
630             case 'LOW': return 'bg-info';
631             default: return 'bg-secondary';
632         }
633     }
634
635     function getRiskFactorExplanation(featureName, contribution) {
636         const explanations = {
637             'Age': `Older age groups show ${contribution > 0.05 ? 'significantly ' : ''}higher churn
638     rates`,
639             'Frequent Flyer': 'Frequent flyers are more likely to explore competitor offers',
640             'Income': 'Income level strongly influences service affordability and retention',
641             'Services': 'Number of services impacts overall customer commitment',
642             'Social': 'Social media connected customers are more exposed to competitor marketing',
643             'Hotel': 'Hotel booking history indicates engagement with travel services'
644         };
645
646         // Check for partial matches
647         featureName = featureName.toLowerCase();
648         for (const [key, explanation] of Object.entries(explanations)) {
649             if (featureName.includes(key.toLowerCase())) {
650                 return explanation;
651             }
652         }
653
654         return 'This feature significantly impacts churn prediction';
655     }
656
```

```
function getProtectiveFactorExplanation(featureName, contribution) {
    const explanations = {
        'Age': 'Age group shows lower churn propensity',
        'Frequent Flyer': 'Loyalty program participation increases retention',
        'Income': 'Income stability supports service continuity',
        'Services': 'Service bundle reduces likelihood of switching',
        'Social': 'Limited social exposure reduces competitive pressure',
        'Hotel': 'Travel service engagement indicates satisfaction'
    };

    // Check for partial matches
    featureName = featureName.toLowerCase();
    for (const [key, explanation] of Object.entries(explanations)) {
        if (featureName.includes(key.toLowerCase())) {
            return explanation;
        }
    }

    return 'This feature reduces churn risk probability';
}

function generateKeyTakeaways(allContributions, increasingFactors, decreasingFactors) {
    const takeaways = [];

    // Overall risk assessment
    const totalRisk = increasingFactors.reduce((sum, f) => sum + f.abs_contribution, 0);
    const totalProtection = decreasingFactors.reduce((sum, f) => sum +
Math.abs(f.contribution), 0);

    if (totalRisk > totalProtection * 1.5) {
        takeaways.push({
            icon: 'fa-exclamation-triangle',
            color: 'danger',
            text: 'Customer has significantly more risk factors than protective factors'
        });
    } else if (totalProtection > totalRisk * 1.5) {
        takeaways.push({
            icon: 'fa-shield-alt',
            color: 'success',
            text: 'Customer has strong protective factors against churn'
        });
```

```
698         } else {
699             takeaways.push({
700                 icon: 'fa-balance-scale',
701                 color: 'warning',
702                 text: 'Balanced mix of risk and protective factors'
703             });
704         }
705 
706         // Top risk factor
707         if (increasingFactors.length > 0) {
708             const topRisk = increasingFactors[0];
709             takeaways.push({
710                 icon: 'fa-arrow-up',
711                 color: 'danger',
712                 text: `Primary risk: ${topRisk.simplified_name}`
713             });
714         }
715 
716         // Top protective factor
717         if (decreasingFactors.length > 0) {
718             const topProtective = decreasingFactors[0];
719             takeaways.push({
720                 icon: 'fa-arrow-down',
721                 color: 'success',
722                 text: `Key strength: ${topProtective.simplified_name}`
723             });
724         }
725 
726         // Number of significant factors
727         const significantFactors = allContributions.filter(c => c.abs_contribution > 0.02).length;
728         if (significantFactors > 5) {
729             takeaways.push({
730                 icon: 'fa-chart-line',
731                 color: 'info',
732                 text: `Multiple factors (${significantFactors}) significantly influence churn risk`
733             });
734         }
735 
736         // Recommendation focus
737         if (increasingFactors.length > decreasingFactors.length) {
738             takeaways.push({
```

```
            icon: 'fa-bullseye',
            color: 'primary',
            text: 'Focus interventions on addressing the top risk factors'
        });
    } else {
        takeaways.push({
            icon: 'fa-bullseye',
            color: 'primary',
            text: 'Leverage existing strengths while addressing key risk areas'
        });
    }

    return takeaways.slice(0, 5); // Return top 5 takeaways
}

// Make functions globally available
window.handleFormSubmit = handleFormSubmit;
window.loadSampleData = loadSampleData;
window.showModelInfo = showModelInfo;
window.checkModelStatus = checkModelStatus;

// Helper function to reset form
window.resetForm = function() {
    document.getElementById('resultsCard').style.display = 'none';
    document.getElementById('welcomeCard').style.display = 'block';
    document.getElementById('riskFactorsSummary').style.display = 'none';

    // Clear form
    document.getElementById('age').value = '';
    document.getElementById('frequent_flyer').value = 'No';
    document.getElementById('income_class').value = 'Middle Income';
    document.getElementById('services_opted').value = '';
    document.getElementById('social_sync').value = 'No';
    document.getElementById('booked_hotel').value = 'No';

    // Scroll to top
    window.scrollTo({ top: 0, behavior: 'smooth' });
};

// Add keyboard shortcuts
document.addEventListener('keydown', function(event) {
```

```
780     // Ctrl+Enter to submit form
781     if (event.ctrlKey && event.key === 'Enter') {
782         event.preventDefault();
783         handleFormSubmit(event);
784     }
785     // F5 to load sample data
786     if (event.key === 'F5') {
787         event.preventDefault();
788         loadSampleData();
789     }
790 });
791
```

## 10.8 EDA Analysis Codes

```
1  import pandas as pd
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import os
6  from datetime import datetime
7  from scipy import stats
8
9  #
10 # Setting up the Output Directory
11 #
12
13 # Create output directories with timestamp for unique runs
14 timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
15 base_output_dir = f"data/eda_output_{timestamp}"
16 tables_dir = os.path.join(base_output_dir, "tables")
17 visuals_dir = os.path.join(base_output_dir, "visuals")
18
19 # Create directories if they don't exist
20 os.makedirs(tables_dir, exist_ok=True)
21 os.makedirs(visuals_dir, exist_ok=True)
22
23 print(f"Output will be saved to: {base_output_dir}")
24 print(f"Tables directory: {tables_dir}")
25 print(f"Visuals directory: {visuals_dir}")
26
27 #
28 # Load data .. no need to check column names
```

```
# No missing values .. column names Ok ..
#
================================================================================
=


df = pd.read_csv('Customertravel.csv')

if 'Ages' in df.columns and 'Age' not in df.columns:
    df.rename(columns={'Ages': 'Age'}, inplace=True)
# quick sanity print to confirm
print("Columns after rename:", list(df.columns))


print("\nFirst 5 rows:")
print(df.head())
print("\nColumn types:")
print(df.dtypes)

#
# Helper functions - to save csv files to /tables dir
# and charts to /visuals
#

def save_table_to_csv(dataframe, filename, description=""):
    """Save DataFrame to CSV with description"""
    filepath = os.path.join(tables_dir, filename)
    dataframe.to_csv(filepath, index=True)
    print(f"  ✓ Saved table: {filename}")
```

```
57      if description:
58          print(f"   Description: {description}")
59      return filepath
60
61
62  def save_plot_to_png(fig, filename, description=""):
63      """Save matplotlib figure to PNG"""
64      filepath = os.path.join(visuals_dir, filename)
65      fig.savefig(filepath, dpi=300, bbox_inches='tight', facecolor='white')
66      print(f"  ✓ Saved visual: {filename}")
67      if description:
68          print(f"   Description: {description}")
69      return filepath
70
71
72  #
73  # 1. Target variable analysis
74  #
75
76  print("\n" + "=" * 80)
77  print("1. TARGET VARIABLE ANALYSIS")
78  print("=" * 80)
79
80  # Create a copy of Target with labels for better visualization
81  df['Target_Label'] = df['Target'].map({0: 'No Churn', 1: 'Churned'})
82
83  # Target distribution summary
84  target_summary = df['Target'].value_counts().reset_index()
```

```
target_summary.columns = ['Churn', 'Count']
target_summary['Percentage'] = (target_summary['Count'] / len(df) * 100).round(2)
target_summary['Churn_Label'] = target_summary['Churn'].map({0: 'No Churn', 1:
'Churned'})

# Save target summary table
save_table_to_csv(target_summary, "1_target_summary.csv",
            "Distribution of churn vs no churn in dataset")

# Visualize target distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Pie chart
axes[0].pie(target_summary['Count'], labels=target_summary['Churn_Label'],
        autopct='%1.1f%%', colors=['skyblue', 'salmon'], startangle=90)
axes[0].set_title('Churn Distribution (Pie Chart)', fontsize=14, fontweight='bold')

# Bar chart
sns.barplot(data=target_summary, x='Churn_Label', y='Count', hue='Churn_Label',
        palette=['skyblue', 'salmon'], legend=False, ax=axes[1])
axes[1].set_title('Churn Distribution (Bar Chart)', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Churn Status')
axes[1].set_ylabel('Count')

# Add count labels on bars
for i, v in enumerate(target_summary['Count']):
    axes[1].text(i, v + 10, str(v), ha='center', va='bottom', fontweight='bold')

```

```
113 plt.tight_layout()
114 save_plot_to_png(fig, "1_target_distribution.png",
115             "Bar Chart - Churn Distribution")
116 plt.show()
117
118 #
119 # 2. Age Analysis
120 #
121
122 print("\n" + "=" * 80)
123 print("2. AGE ANALYSIS")
124 print("=" * 80)
125
126 # Create age groups
127 df['AgeGroup'] = pd.cut(df['Age'],
128                         bins=[20, 30, 40, 50, 60],
129                         labels=['20-29', '30-39', '40-49', '50-59'])
130
131 # Age statistics by churn status with MODE
132 age_stats_by_churn = df.groupby('Target')['Age'].agg([
133     'count', 'mean', 'median', lambda x: stats.mode(x)[0][0], 'std', 'min', 'max',
134     lambda x: x.quantile(0.25), lambda x: x.quantile(0.75)
135 ]).round(2)
136 age_stats_by_churn.columns = ['Count', 'Mean', 'Median', 'Mode', 'Std', 'Min', 'Max', 'Q1',
137 'Q3']
138 age_stats_by_churn.index = age_stats_by_churn.index.map({0: 'No Churn', 1: 'Churned'})
139
140 # Overall age statistics with MODE
```

```
141 age_mode = stats.mode(df['Age'])[0][0]
142 overall_age_stats = pd.DataFrame({
143     'Statistic': ['Count', 'Mean', 'Median', 'Mode', 'Std', 'Min', 'Max', 'Q1', 'Q3'],
144     'Value': [
145         len(df['Age']),
146         round(df['Age'].mean(), 2),
147         round(df['Age'].median(), 2),
148         round(age_mode, 2),
149         round(df['Age'].std(), 2),
150         df['Age'].min(),
151         df['Age'].max(),
152         round(df['Age'].quantile(0.25), 2),
153         round(df['Age'].quantile(0.75), 2)
154     ]
155 })
156
157 # Churn rate by age group
158 age_group_churn = df.groupby('AgeGroup')['Target'].agg(['count', 'mean', 'sum']).round(3)
159 age_group_churn.columns = ['Count', 'Churn_Rate', 'Churned_Count']
160 age_group_churn['No_Churn_Count'] = age_group_churn['Count'] -
161 age_group_churn['Churned_Count']
162
163 # Save age analysis tables
164 save_table_to_csv(age_stats_by_churn, "2_age_stats_by_churn.csv",
165                   "Age statistics grouped by churn status")
166 save_table_to_csv(overall_age_stats, "2_overall_age_stats.csv",
167                   "Overall age statistics for entire dataset")
168 save_table_to_csv(age_group_churn, "2_age_group_churn_rates.csv",
```

```
169                 "Churn rates by age group (binned)")
170
171     # Visualize age analysis
172     fig, axes = plt.subplots(2, 2, figsize=(16, 12))
173
174     # Histogram with KDE - using Target_Label for proper coloring
175     sns.histplot(data=df, x='Age', hue='Target_Label', kde=True, element='step',
176                  palette=['skyblue', 'salmon'], ax=axes[0, 0])
177     axes[0, 0].set_title('Age Distribution by Churn Status', fontsize=14, fontweight='bold')
178     axes[0, 0].set_xlabel('Age')
179     axes[0, 0].set_ylabel('Frequency')
180
181     # Boxplot - using Target_Label for proper coloring
182     sns.boxplot(data=df, x='Target_Label', y='Age', hue='Target_Label',
183                 palette=['skyblue', 'salmon'], legend=False, ax=axes[0, 1])
184     axes[0, 1].set_title('Age Distribution by Churn Status (Boxplot)', fontsize=14,
185     fontweight='bold')
186     axes[0, 1].set_xlabel('Churn Status')
187     axes[0, 1].set_ylabel('Age')
188
189     # Churn rate by age group
190     sns.barplot(data=age_group_churn.reset_index(), x='AgeGroup', y='Churn_Rate',
191                 color='steelblue', ax=axes[1, 0])
192     axes[1, 0].set_title('Churn Rate by Age Group', fontsize=14, fontweight='bold')
193     axes[1, 0].set_xlabel('Age Group')
194     axes[1, 0].set_ylabel('Churn Rate')
195
196     # Add churn rate labels on bars
```

```
197  for i, v in enumerate(age_group_churn['Churn_Rate']):
198      axes[1, 0].text(i, v + 0.01, f'{v:.2%}', ha='center', va='bottom', fontweight='bold')
199
200  #
201  # 3 - Categorical analysis
202  #
203
204  print("\n" + "=" * 80)
205  print("3. CATEGORICAL VARIABLES ANALYSIS")
206  print("=" * 80)
207
208  categorical_cols = ['FrequentFlyer', 'AnnualIncomeClass',
209                      'AccountSyncedToSocialMedia', 'BookedHotelOrNot']
210
211  # Initialize dictionary to store all categorical analysis results
212  categorical_analysis = {}
213
214  for col in categorical_cols:
215      print(f"\nAnalyzing: {col}")
216
217      # Calculate churn rates
218      churn_stats = df.groupby(col)['Target'].agg([
219          ('Count', 'count'),
220          ('Churned', 'sum'),
221          ('Not_Churned', lambda x: (x == 0).sum()),
222          ('Churn_Rate', 'mean'),
223          ('Std_Error', lambda x: np.std(x) / np.sqrt(len(x)))
224      ]).round(3)
```

```
225
226     churn_stats['Churn_Rate_Pct'] = (churn_stats['Churn_Rate'] * 100).round(1)
227
228     # Save individual table
229     save_table_to_csv(churn_stats, f"3_{col.lower()}_churn_stats.csv",
230                       f"Churn statistics for {col}")
231
232     # Store for combined analysis
233     categorical_analysis[col] = churn_stats
234
235 # Create combined categorical summary
236 combined_categorical = []
237 for col in categorical_cols:
238     temp_df = categorical_analysis[col].copy()
239     temp_df['Variable'] = col
240     temp_df['Category'] = temp_df.index
241     combined_categorical.append(temp_df.reset_index(drop=True))
242
243 combined_categorical_df = pd.concat(combined_categorical, ignore_index=True)
244 combined_categorical_df = combined_categorical_df[['Variable', 'Category', 'Count',
245                                                     'Churned', 'Not_Churned',
246                                                     'Churn_Rate', 'Churn_Rate_Pct',
247                                                     'Std_Error']]
248
249 save_table_to_csv(combined_categorical_df, "3_combined_categorical_analysis.csv",
250                   "Combined churn analysis for all categorical variables")
251
252 # Visualize categorical variables
```

```
fig, axes = plt.subplots(2, 2, figsize=(18, 12))
axes = axes.flatten()

for idx, col in enumerate(categorical_cols):
    # Create stacked bar chart
    crosstab = pd.crosstab(df[col], df['Target'], normalize='index')
    crosstab = crosstab[[0, 1]]  # Ensure correct order

    crosstab.plot(kind='bar', stacked=True, ax=axes[idx],
                  color=['skyblue', 'salmon'], width=0.7)

    axes[idx].set_title(f'{col} - Churn Distribution', fontsize=14, fontweight='bold')
    axes[idx].set_xlabel(col)
    axes[idx].set_ylabel('Proportion')
    axes[idx].legend(['No Churn', 'Churned'], loc='upper right')
    axes[idx].grid(axis='y', alpha=0.3)

    # Add percentage labels
    for i, (index, row) in enumerate(crosstab.iterrows()):
        height_accum = 0
        for j, val in enumerate(row):
            if val > 0.05:  # Only label if segment is large enough
                axes[idx].text(i, height_accum + val / 2,
                               f'{val:.1%}',
                               ha='center', va='center',
                               fontsize=9, fontweight='bold',
                               color='white' if j == 1 else 'black')
            height_accum += val
```

```
281
282 plt.tight_layout()
283 save_plot_to_png(fig, "3_categorical_variables_stacked.png",
284                  "Stacked bar charts showing churn distribution for all categorical variables")
285 plt.show()
286
287 #
288 # 4. Services Opted analysis
289 #
290
291 print("\n" + "=" * 80)
292 print("4. SERVICES OPTED ANALYSIS")
293 print("=" * 80)
294
295 # ServicesOpted statistics by churn status with MODE
296 services_stats_by_churn = df.groupby('Target')['ServicesOpted'].agg([
297     'count', 'mean', 'median', lambda x: stats.mode(x)[0][0], 'std', 'min', 'max',
298     lambda x: x.quantile(0.25), lambda x: x.quantile(0.75)
299 ]).round(2)
300 services_stats_by_churn.columns = ['Count', 'Mean', 'Median', 'Mode', 'Std', 'Min', 'Max',
301 'Q1', 'Q3']
302 services_stats_by_churn.index = services_stats_by_churn.index.map({0: 'No Churn', 1:
303 'Churned'})
304
305 # ServicesOpted frequency distribution
306 services_freq = df['ServicesOpted'].value_counts().sort_index().reset_index()
307 services_freq.columns = ['Services_Opted', 'Count']
308 services_freq['Percentage'] = (services_freq['Count'] / len(df) * 100).round(2)
309
```

```
# Churn rate by ServicesOpted value
services_churn_rate = df.groupby('ServicesOpted')['Target'].agg([
    ('Count', 'count'),
    ('Churned', 'sum'),
    ('Churn_Rate', 'mean')
]).round(3)
services_churn_rate['Churn_Rate_Pct'] = (services_churn_rate['Churn_Rate'] *
100).round(1)

# Save ServicesOpted tables
save_table_to_csv(services_stats_by_churn, "4_services_stats_by_churn.csv",
                  "ServicesOpted statistics grouped by churn status")
save_table_to_csv(services_freq, "4_services_frequency_distribution.csv",
                  "Frequency distribution of ServicesOpted values")
save_table_to_csv(services_churn_rate, "4_services_churn_rates.csv",
                  "Churn rates by ServicesOpted value")

# Visualize ServicesOpted analysis
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Histogram with KDE
sns.histplot(data=df, x='ServicesOpted', hue='Target_Label', kde=True, element='step',
             palette=['skyblue', 'salmon'], bins=range(1, 8), ax=axes[0, 0])
axes[0, 0].set_title('ServicesOpted Distribution by Churn Status', fontsize=14,
fontweight='bold')
axes[0, 0].set_xlabel('Number of Services Opted')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_xticks(range(1, 7))

```

```
# Boxplot
sns.boxplot(data=df, x='Target_Label', y='ServicesOpted', hue='Target_Label',
        palette=['skyblue', 'salmon'], legend=False, ax=axes[0, 1])
axes[0, 1].set_title('ServicesOpted Distribution (Boxplot)', fontsize=14, fontweight='bold')
axes[0, 1].set_xlabel('Churn Status')
axes[0, 1].set_ylabel('Services Opted')

# Churn rate by ServicesOpted value
sns.barplot(data=services_churn_rate.reset_index(), x='ServicesOpted', y='Churn_Rate',
        color='steelblue', ax=axes[1, 0])
axes[1, 0].set_title('Churn Rate by ServicesOpted Value', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Number of Services Opted')
axes[1, 0].set_ylabel('Churn Rate')
axes[1, 0].set_xticks(range(0, 6))

# Add churn rate labels on bars
for i, v in enumerate(services_churn_rate['Churn_Rate']):
    axes[1, 0].text(i, v + 0.01, f'{v:.2%}', ha='center', va='bottom', fontweight='bold')

#
# 5. Outlier Detection ANalyss
#

print("\n" + "=" * 80)
print("5. OUTLIER DETECTION ANALYSIS (Tukey's Method)")
print("=" * 80)

numeric_cols = ['Age', 'ServicesOpted']
```

```
outlier_results = []

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_fence = Q1 - 1.5 * IQR
    upper_fence = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[col] < lower_fence) | (df[col] > upper_fence)]
    outlier_count = len(outliers)

    # Create outlier details
    outlier_details = outliers[[col, 'Target']].copy()
    outlier_details['Outlier_Type'] = np.where(outlier_details[col] < lower_fence, 'Low', 'High')

    # Save individual outlier details
    save_table_to_csv(outlier_details, f"5_{col.lower()}_outliers_detail.csv",
                      f"Detailed outlier information for {col}")

    # Calculate mode for summary
    mode_val = stats.mode(df[col])[0][0]

    # Summary statistics with MODE
    outlier_results.append({
        'Variable': col,
        'Q1': round(Q1, 2),
```

```
395         'Q3': round(Q3, 2),
396         'IQR': round(IQR, 2),
397         'Lower_Fence': round(lower_fence, 2),
398         'Upper_Fence': round(upper_fence, 2),
399         'Outlier_Count': outlier_count,
400         'Outlier_Percentage': round((outlier_count / len(df)) * 100, 2),
401         'Min_Value': round(df[col].min(), 2),
402         'Max_Value': round(df[col].max(), 2),
403         'Mean': round(df[col].mean(), 2),
404         'Median': round(df[col].median(), 2),
405         'Mode': round(mode_val, 2)  # Added mode
406     })
407
408 # Create outlier summary table
409 outlier_summary_df = pd.DataFrame(outlier_results)
410 save_table_to_csv(outlier_summary_df, "5_outlier_summary.csv",
411                   "Summary of outlier detection using Tukey's method")
412
413 # Visualize outliers
414 fig, axes = plt.subplots(1, 2, figsize=(16, 6))
415
416 for idx, col in enumerate(numeric_cols):
417     # Boxplot showing outliers
418     boxplot_data = df[[col, 'Target_Label']].copy()
419
420     sns.boxplot(data=boxplot_data, x='Target_Label', y=col, hue='Target_Label',
421                 palette=['skyblue', 'salmon'], legend=False, ax=axes[idx])
422
```

```
423     # Calculate outlier boundaries
424     Q1 = df[col].quantile(0.25)
425     Q3 = df[col].quantile(0.75)
426     IQR = Q3 - Q1
427     lower_fence = Q1 - 1.5 * IQR
428     upper_fence = Q3 + 1.5 * IQR
429
430     # Add fence lines
431     axes[idx].axhline(y=lower_fence, color='red', linestyle='--', alpha=0.5, label='Lower Fence')
432     axes[idx].axhline(y=upper_fence, color='red', linestyle='--', alpha=0.5, label='Upper Fence')
433
434     axes[idx].set_title(f'{col} - Boxplot with Outlier Boundaries', fontsize=14,
435 fontweight='bold')
436     axes[idx].set_xlabel('Churn Status')
437     axes[idx].set_ylabel(col)
438
439     if idx == 0:
440         axes[idx].legend()
441
442 plt.tight_layout()
443 save_plot_to_png(fig, "5_outlier_detection_boxplots.png",
444                  "Boxplots showing outliers with Tukey's fences")
445 plt.show()
446
447 #
448 # 6. Paired interaction analysis
449 #
450
```

```
451 print("\n" + "=" * 80)
452 print("6. INTERACTION ANALYSIS")
453 print("=" * 80)
454
455 # Interaction 1: AnnualIncomeClass × FrequentFlyer
456 interaction_1 = pd.crosstab([df['AnnualIncomeClass'], df['FrequentFlyer']],
457                             df['Target'],
458                             normalize='index')[1].reset_index()
459 interaction_1.columns = ['IncomeClass', 'FrequentFlyer', 'Churn_Rate']
460
461 # Add count information
462 interaction_counts = df.groupby(['AnnualIncomeClass', 'FrequentFlyer']).size().reset_index()
463 interaction_counts.columns = ['IncomeClass', 'FrequentFlyer', 'Count']
464 interaction_1 = pd.merge(interaction_1, interaction_counts, on=['IncomeClass',
465 'FrequentFlyer'])
466
467 save_table_to_csv(interaction_1, "6_interaction_income_frequentflyer.csv",
468                   "Interaction analysis: Income Class × FrequentFlyer Status")
469
470 # Interaction 2: ServicesOpted × BookedHotelOrNot
471 df['ServicesGroup'] = pd.cut(df['ServicesOpted'],
472                              bins=[0, 2, 4, 6],
473                              labels=['Low (1-2)', 'Medium (3-4)', 'High (5-6)'])
474
475 interaction_2 = pd.crosstab([df['ServicesGroup'], df['BookedHotelOrNot']],
476                             df['Target'],
477                             normalize='index')[1].reset_index()
478 interaction_2.columns = ['ServicesGroup', 'BookedHotel', 'Churn_Rate']
```

```
479
480 # Add count information
481 interaction_counts_2 = df.groupby(['ServicesGroup',
482 'BookedHotelOrNot']).size().reset_index()
483 interaction_counts_2.columns = ['ServicesGroup', 'BookedHotel', 'Count']
484 interaction_2 = pd.merge(interaction_2, interaction_counts_2, on=['ServicesGroup',
485 'BookedHotel'])
486
487 save_table_to_csv(interaction_2, "6_interaction_services_hotel.csv",
488                   "Interaction analysis: Services Group × BookedHotel Status")
489
490 # Visualize interactions
491 fig, axes = plt.subplots(1, 2, figsize=(18, 6))
492
493 # Interaction 1: Income × FrequentFlyer
494 sns.barplot(data=interaction_1, x='IncomeClass', y='Churn_Rate', hue='FrequentFlyer',
495             palette='viridis', ax=axes[0])
496 axes[0].set_title('Churn Rate: Income Class × FrequentFlyer Status',
497                   fontsize=14, fontweight='bold')
498 axes[0].set_xlabel('Income Class')
499 axes[0].set_ylabel('Churn Rate')
500 axes[0].legend(title='Frequent Flyer')
501
502 # Add count labels
503 for container in axes[0].containers:
504     for bar in container:
505         height = bar.get_height()
506         if not np.isnan(height) and height > 0:
507             axes[0].text(bar.get_x() + bar.get_width() / 2, height + 0.01,
```

```
                    f'{height:.2%}', ha='center', va='bottom', fontsize=9)

# Interaction 2: Services × Hotel
sns.barplot(data=interaction_2, x='ServicesGroup', y='Churn_Rate', hue='BookedHotel',
        palette='coolwarm', ax=axes[1])
axes[1].set_title('Churn Rate: Services Group × BookedHotel Status',
              fontsize=14, fontweight='bold')
axes[1].set_xlabel('Services Group')
axes[1].set_ylabel('Churn Rate')
axes[1].legend(title='Booked Hotel')

# Add count labels
for container in axes[1].containers:
    for bar in container:
        height = bar.get_height()
        if not np.isnan(height) and height > 0:
            axes[1].text(bar.get_x() + bar.get_width() / 2, height + 0.01,
                    f'{height:.2%}', ha='center', va='bottom', fontsize=9)

plt.tight_layout()
save_plot_to_png(fig, "6_interaction_analysis.png",
              "Interaction analysis between key variables")
plt.show()

#
# 7. Correlation analysis
#

```

```
536 print("\n" + "=" * 80)
537 print("7. CORRELATION ANALYSIS")
538 print("=" * 80)
539
540 # Prepare data for correlation analysis
541 df_corr = df.copy()
542
543 # Encode categorical variables for correlation
544 encoding_map = {
545     'FrequentFlyer': {'Yes': 1, 'No': 0, 'No Record': 0},
546     'AccountSyncedToSocialMedia': {'Yes': 1, 'No': 0},
547     'BookedHotelOrNot': {'Yes': 1, 'No': 0},
548     'AnnualIncomeClass': {'Low Income': 0, 'Middle Income': 1, 'High Income': 2}
549 }
550
551 for col, mapping in encoding_map.items():
552     if col in df_corr.columns:
553         df_corr[col + '_Encoded'] = df_corr[col].map(mapping)
554
555 # Select columns for correlation
556 corr_columns = ['Age', 'ServicesOpted', 'Target',
557                 'FrequentFlyer_Encoded', 'AccountSyncedToSocialMedia_Encoded',
558                 'BookedHotelOrNot_Encoded', 'AnnualIncomeClass_Encoded']
559
560 corr_matrix = df_corr[corr_columns].corr()
561
562 # Rename columns for better readability
563 column_names = {
```

```
564     'Age': 'Age',
565     'ServicesOpted': 'ServicesOpted',
566     'Target': 'Churn',
567     'FrequentFlyer_Encoded': 'FrequentFlyer',
568     'AccountSyncedToSocialMedia_Encoded': 'SocialMediaSync',
569     'BookedHotelOrNot_Encoded': 'BookedHotel',
570     'AnnualIncomeClass_Encoded': 'IncomeClass'
571 }
572
573 corr_matrix = corr_matrix.rename(columns=column_names, index=column_names)
574
575 # Save correlation matrix
576 save_table_to_csv(corr_matrix, "7_correlation_matrix.csv",
577                   "Pearson correlation matrix for all variables (encoded)")
578
579 # Extract correlations with target
580 target_correlations = corr_matrix['Churn'].sort_values(ascending=False)
581 target_corr_df = pd.DataFrame({
582     'Variable': target_correlations.index,
583     'Correlation_with_Churn': target_correlations.values
584 }).round(3)
585
586 save_table_to_csv(target_corr_df, "7_correlations_with_target.csv",
587                   "Correlation coefficients between each variable and churn")
588
589 # Visualize correlation matrix
590 fig, axes = plt.subplots(1, 2, figsize=(18, 7))
591
```

```
# Heatmap
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            fmt='.2f', square=True, mask=mask, ax=axes[0],
            cbar_kws={"shrink": 0.8})
axes[0].set_title('Correlation Matrix Heatmap', fontsize=14, fontweight='bold')

# Bar chart of correlations with target
sorted_corr = target_corr_df[target_corr_df['Variable'] !=
'Churn'].sort_values('Correlation_with_Churn')
colors = ['salmon' if x > 0 else 'skyblue' for x in sorted_corr['Correlation_with_Churn']]
bars = axes[1].barh(sorted_corr['Variable'], sorted_corr['Correlation_with_Churn'],
color=colors)
axes[1].set_title('Correlation with Churn', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Correlation Coefficient')
axes[1].axvline(x=0, color='black', linestyle='-', alpha=0.3)

# Add value labels
for bar in bars:
    width = bar.get_width()
    axes[1].text(width + (0.01 if width >= 0 else -0.01), bar.get_y() + bar.get_height() / 2,
                 f'{width:.3f}', va='center',
                 ha='left' if width >= 0 else 'right',
                 fontweight='bold')

plt.tight_layout()
save_plot_to_png(fig, "7_correlation_analysis.png",
                 "Correlation heatmap and bar chart showing relationships with churn")
plt.show()
```

```
621
622     print('Analysis Complete!')
```