Configuration Manual

MSc Research Project
Data Analytics

Pattamaporn Sanluang
Student ID: X21122466

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pattamaporn Sanluang
Student ID:	X21122466
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	16/09/2024
Project Title:	Configuration Manual
Word Count:	824
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Pattamaporn Sanluang
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pattamaporn Sanluang
X21122466

1 Introduction

This configuration manual details the steps in implementing this research to address the customer churn in the telecommunications sector by utilizing machine learning models along with Explainable AI (XAI) methods to enhance transparency and interpretability. The approach incorporates data from a telecommunications dataset and applies various XAI methods, including SHAP, LIME, and Feature Importance to identify and interpret the key factors influencing customer churn.

2 Environmental Setup

This section details the technical environment required to implement the research. It covers the hardware and software specifications, along with the setup of necessary tools and libraries used throughout the project.

2.1 Hardware Specifications

The research was conducted on a local machine equipped with an AMD Ryzen 7 5800H processor, 16GB of RAM, running Windows 11 Home Single Language (64-bit) as shown in figure 1 below.

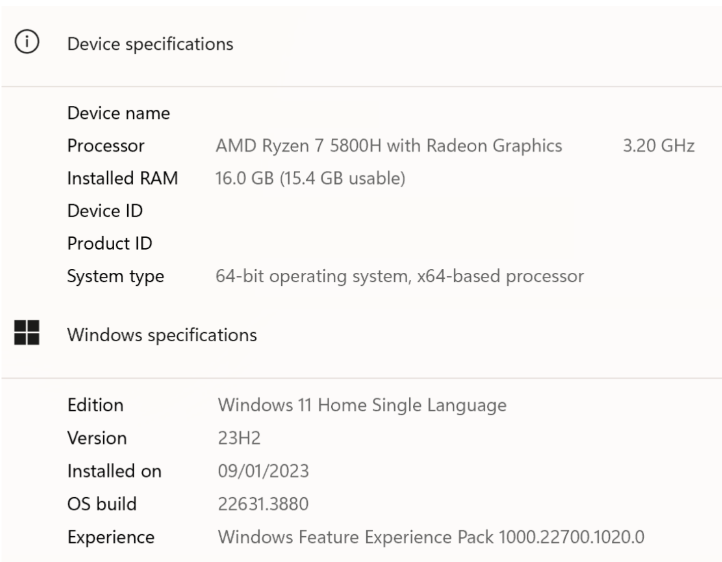

Figure 1: Hardware Specification

2.2 Software Specifications

This research used Anaconda as the virtual environment manager and Jupyter Notebook as the interactive platform for executing the Python programming language, manipulating data, and documenting the process.

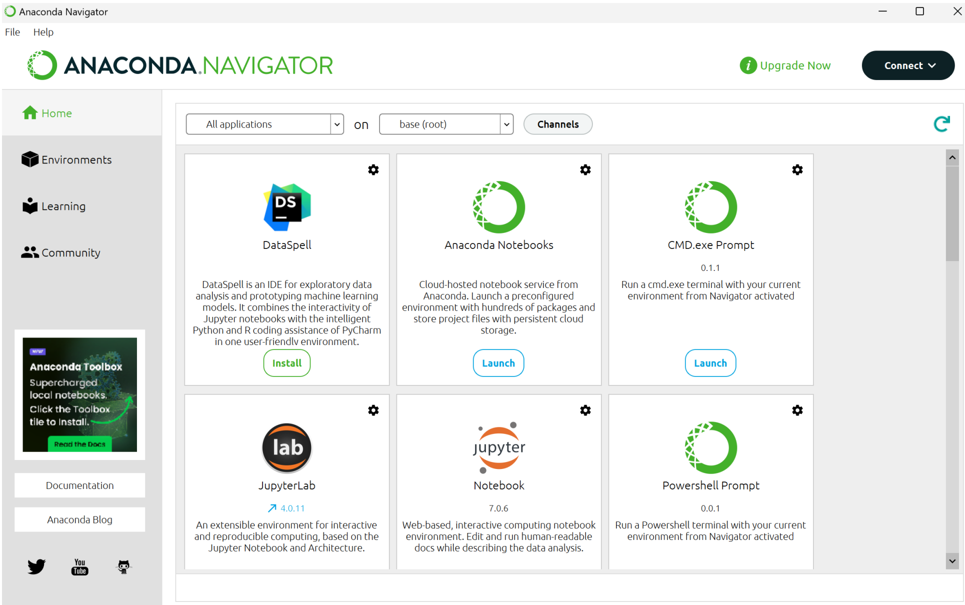

Figure 2: Anaconda Navigator

2.3 Python Library Packages

```python
import pandas as pd
import numpy as np
from numpy import random

# Visualisation & Stats
import matplotlib.pyplot as plt
import seaborn as sns
import re
import statsmodels.api as sm
from scipy import stats
import scipy.stats as stats

# Preprocessing & Modelling
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
import xgboost as xgb

# XAI Methods
import shap
import lime
import lime.lime_tabular

# Evaluation
from sklearn.metrics import roc_auc_score, f1_score, accuracy_score, roc_curve
from sklearn.metrics import confusion_matrix, classification_report, auc
```

Figure 3: Python Libraries

Python programming language and libraries were utilised for implementing the baseline machine learning models and XAI techniques. The libraries used include Pandas for data manipulation, Scikit-learn for machine learning, SHAP and LIME for interpretability, SMOTE for handling imbalanced datasets, Matplotlib for visualizations along with modeling and evaluation metrics.

3 Data Acquisition

Table 1 below details 21 features and its description in Telco dataset used in this research. It was acquired from open-sourced, Kaggle [1]

Feature	**Description**
customerID	A unique ID that identifies each customer
gender	The customer's gender
SeniorCitizen	The customer is 65 or older
Partner	The customer is married
Dependents	The customer lives with any dependents
tenure	The total amount of months that the customer has been with the company
PhoneService	Customer subscribes to home phone service
MultipleLines	Customer subscribes to multiple telephone lines
InternetService	Customer subscribes to Internet service
OnlineSecurity	Customer subscribes to additional online security service
OnlineBackup	Customer subscribes to additional online backup service
DeviceProtection	Customer subscribes to an additional device protection plan for their Internet equipment
TechSupport	Customer subscribes to additional technical support plan
StreamingTV	Customer uses their Internet service to stream television from a third party provider
StreamingMovies	Customer uses their Internet service to stream movies from a third party provider
Contract	Customer's current contract type: Month-to-Month, One Year, Two Year
PaperlessBilling	Customer has chosen paperless billing
PaymentMethod	How the customer pays their bill: Bank Withdrawal, Credit Card, Mailed Check
MonthlyCharges	Customer's current total monthly charges of their services
TotalCharges	Total charges
Churn	Customer's churn status, whether they left or remained with the company

Table 1: Dataset Features and Descriptions

4 Data Preprocessing

This section shows the steps taken to prepare the data for modeling. This includes data preparation, Exploratory Data Analysis and data transformation. The preprocessing

[1] `https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset.`

ensures that the data is in the best possible state for accurate and reliable model training.

4.1 Data Preparation

```
file_path = r"C:\Users\Asus OLED\Desktop\Research\Telco_Churn.csv"
df = pd.read_csv(file_path)
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	St
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	

Figure 4: Imported Telco dataset

Telco dataset called Telco_Churn.csv was imported then loaded into df dataframe using Pandas function `read_csv`. The dataframe was checked by printing out to see if it loaded succesfully.

```
df.drop(columns=['customerID'], inplace=True)
```

```
df.shape
```

```
(7043, 20)
```

```
#checking for duplicate rows
df.duplicated().sum()
```

```
22
```

```
#drop dupes
df.drop_duplicates(inplace=True)
```

```
na = df.isna().sum().sort_values(ascending=False)
pct = 100 * na / df.shape[0]
nan_stats = pd.concat([na, pct], axis=1)
nan_stats.columns = ['num_of_null', 'percentage_of_null']
nan_stats
```

	num_of_null	percentage_of_null
TotalCharges	11	0.156673

```
#drop null values in TotalCharges
df.dropna(subset=['TotalCharges'], inplace=True)
```

Figure 5: Data cleaning process

The data cleaning process undertaken in figure5 show the steps to prepare the dataset for analysis. Initially, the `customerID` column is dropped as it does not provide any relevant important for the modeling process. Following this, the dataset was checks and removed any duplicate rows to keep only unique data entries. Lastly, missing values were identified in the TotalCharges column and they were dropped accordingly.

4.2 Exploratory Data Analysis

Figure 6 shows features with low unique counts were converted into categorical columns and then grouped into similar list to facilitate easier visualization and better understand the distribution of these features.

```python
# Columns suitable for conversion to categorical
categorical_cols = [
    'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
    'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
    'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
    'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn'
]

# Convert each column to categorical
for col in categorical_cols:
    df[col] = df[col].astype('category')
```

Figure 6: Categorical features

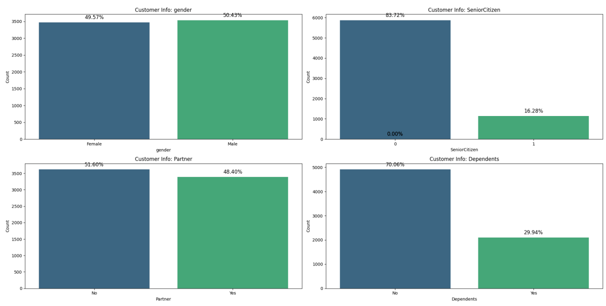

Figure 7: Distribution of Customer Info: gender, SeniorCitizen, Partner, Dependents

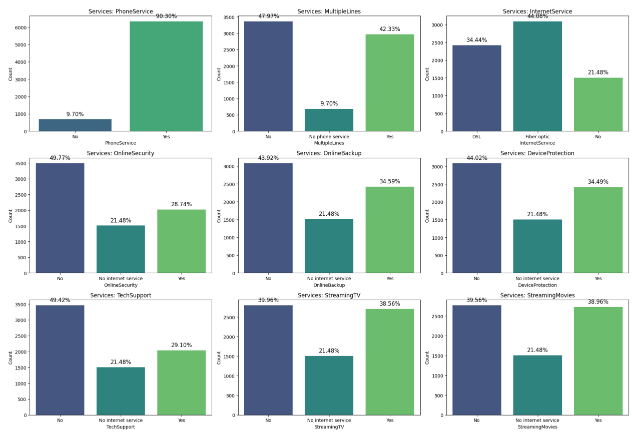

Figure 8: Distribution of services subscribed by customers

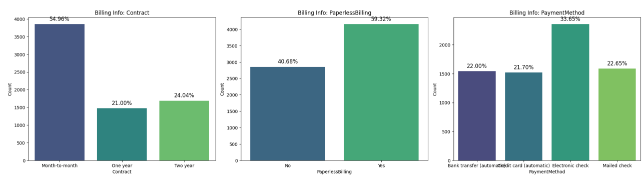

Figure 9: Distribution of Billing Info: Contract type, PaperlessBilling, PaymentMethod

```python
# Define numeric columns
numeric_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
```

Figure 10: Define numerical columns

```python
X_train_backup = X_train.copy()

# Apply SMOTE to the training set
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Standardize the numerical features
scaler = StandardScaler()
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
X_train_res[num_cols] = scaler.fit_transform(X_train_res[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

Figure 11: SMOTE and Standardize technique

Numerical columns were defined and visualized using box plots to explore their distribution. The skewness was noticeable. Hence, the `StandardScaler` from the scikit-learn library was used to standardize the numeric data. Additionally, SMOTE technique) was used to handle class imbalance and the result was demonstrated in the research paper.

4.3 Data Transformation

```python
# Selecting categorical columns
cat_cols = X.select_dtypes(include=['category']).columns.tolist()
df = df.drop('Churn', axis=1)

encoder = OneHotEncoder(sparse_output=False, drop=None)
X_encoded = encoder.fit_transform(df[cat_cols])
X_encoded_df = pd.DataFrame(X_encoded, columns=encoder.get_feature_names_out(cat_cols))


X_encoded_df.reset_index(drop=True, inplace=True)
df.reset_index(drop=True, inplace=True)
X = pd.concat([df.drop(columns=cat_cols), X_encoded_df], axis=1)
```

```python
print(X.shape, y.shape)
```

```
(7010, 46) (7010,)
```

```python
X.head()
```

	tenure	MonthlyCharges	TotalCharges	gender_Female	gender_Male	SeniorCitizen_0	SeniorCitizen_1	Partner_No	Partner_Yes	Dependents_No	...	StreamingMovies_
0	1	29.85	29.85	1.0	0.0	1.0	0.0	0.0	1.0	1.0	...	
1	34	56.95	1889.50	0.0	1.0	1.0	0.0	1.0	0.0	1.0	...	
2	2	53.85	108.15	0.0	1.0	1.0	0.0	1.0	0.0	1.0	...	
3	45	42.30	1840.75	0.0	1.0	1.0	0.0	1.0	0.0	1.0	...	
4	2	70.70	151.65	1.0	0.0	1.0	0.0	1.0	0.0	1.0	...	

Figure 12: One-Hot Encoding

```python
# Stratified train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```python
print(X_train.shape, y_train.shape)
```

```
(4907, 46) (4907,)
```

```python
print(X_test.shape, y_test.shape)
```

```
(2103, 46) (2103,)
```

Figure 13: Train/Test Split

Figure 12 shows how `OneHotEncoder` from Scikit-learn library was used to assign a numerical value to categorical features. Then the dataset was randomly split into training and testing sets with `train_test_split` function at 70/30 ratio.

5 Implementation

5.1 Machine Learning Models & Explainable AI (XAI)

5.1.1 Random Forest

Random Forest was applied as baseline model and in XAI experiment. Code snippets below is process of model training and applying proposed XAI methods to access feature important. Model evaluation metrics was access and shown in classification report.

```
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_res, y_train_res)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred = clf.predict(X_test)
```

```
print("Classification Report: Random Forest")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Classification Report: Random Forest
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1546
           1       0.65      0.51      0.57       557

    accuracy                           0.80      2103
   macro avg       0.74      0.71      0.72      2103
weighted avg       0.79      0.80      0.79      2103
```

Figure 14: Random Forest Model

Random Forest: Feature Important

```
feature_importances = clf.feature_importances_
# Get feature names
feature_names = X.columns

# Pair feature names with their importance scores
feature_importances_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})

# Sort features by importance (descending order)
feature_importances_df = feature_importances_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)
```

Figure 15: Feature Important on Random Forest Model

Random Forest: SHAP

```
explainer_rf = shap.TreeExplainer(clf)
shap_values_rf = explainer_rf.shap_values(X_test)
```

```
# Verify the shapes of the SHAP values
print("Shape of SHAP values:", np.array(shap_values_rf).shape)
shap_values = shap_values_rf[:, :, 1]

# Verify the shapes
print("Shape of SHAP values (positive class):", shap_values.shape)
print("Shape of data matrix:", X_test.shape)
```

Figure 16: SHAP on Random Forest Model

Random Forest: LIME

```
# Create a LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_res.values,
    feature_names=X.columns,
    class_names=['No Churn', 'Churn'],
    mode='classification'
)

instance_idx = 0  # Index of the instance in X_test
instance = X_test.iloc[instance_idx].values
```

```
# Run LIME
explanation_lime = explainer.explain_instance(instance, clf.predict_proba, num_features=len(X.columns))
explanation_lime.show_in_notebook(show_table=True, show_all=False)
```

Figure 17: LIME on Random Forest Model

5.1.2 XGBoost

XGBoost ML algorithm was another model used as baseline model and in XAI-integration experiment. Code snippets below is process of XGBoost model training with its default parameters and applying proposed XAI methods to access feature important.

```
xgb_clf = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb_clf.fit(X_train_res, y_train_res)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=42, ...)
```

```
y_pred = xgb_clf.predict(X_test)
print("Classification Report: XGBoost")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Classification Report: XGBoost
              precision    recall  f1-score   support

           0       0.83      0.88      0.85      1546
           1       0.59      0.50      0.54       557

    accuracy                           0.78      2103
   macro avg       0.71      0.69      0.70      2103
weighted avg       0.77      0.78      0.77      2103
```

Figure 18: Baseline XGBoost Model

XGBoost: Feature Important

```
# Get the feature importance
booster = xgb_clf.get_booster()
importance_dict = booster.get_score(importance_type='weight')

# Convert to a DataFrame
feature_importance_df_xgb = pd.DataFrame({'Feature': list(importance_dict.keys()), 'Importance': list(importance_dict.values())})
feature_importance_df_xgb = feature_importance_df_xgb.sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df_xgb)
plt.title('Feature Importance')
plt.show()
```

Figure 19: Feature Important on XGBoost Model

XGBoost: SHAP

```
explainer_xgb = shap.TreeExplainer(xgb_clf)
shap_values_xgb = explainer_xgb.shap_values(X_test)

shap.summary_plot(shap_values_xgb, X_test, plot_type="bar")
```

Figure 20: SHAP on XGBoost Model

XGBoost: LIME

```
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_res.values,
    feature_names=X.columns,
    class_names=['No Churn', 'Churn'],
    mode='classification'
)

instance_idx = 0
instance = X_test.iloc[instance_idx].values
```

```
# Run LIME
explanation_lime = explainer.explain_instance(instance, xgb_clf.predict_proba, num_features=len(X.columns))
explanation_lime.show_in_notebook(show_table=True, show_all=False)
```

Figure 21: LIME on XGBoost Model

5.1.3 Logistic Regression

Lastly, Logistic Regression algorithm Was applied which the same approach as prior models.

Logistic Regression

```
log_clf = LogisticRegression(random_state=42, max_iter=1000)
log_clf.fit(X_train_res, y_train_res)
y_pred_log = log_clf.predict(X_test)

print("Logistic Regression model")
print(classification_report(y_test, y_pred_log))
print(f"Accuracy: {accuracy_score(y_test, y_pred_log):.4f}")
```

```
Logistic Regression model
              precision    recall  f1-score   support

           0       0.91      0.74      0.82      1546
           1       0.53      0.80      0.63       557

    accuracy                           0.76      2103
   macro avg       0.72      0.77      0.73      2103
weighted avg       0.81      0.76      0.77      2103
```

Figure 22: Logistic Regression Model

Logistic Regression: Feature Important

```
feature_names = X_train_res.columns.tolist()

# Get the coefficients of the features
coefficients = log_clf.coef_[0]
# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})
# Sort the DataFrame by absolute value of coefficients
feature_importance_df['Absolute Coefficient'] = feature_importance_df['Coefficient'].abs()
feature_importance_df = feature_importance_df.sort_values(by='Absolute Coefficient', ascending=False)
print(feature_importance_df)
```

Figure 23: Feature Important on Logistic Regression Model

Logistic Regression: SHAP

```
explainer = shap.Explainer(log_clf, X_train_res)
shap_values = explainer(X_test)
```

Figure 24: SHAP on Logistic Regression Model

Logistic Regression: LIME

```
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_res.values,
    feature_names=X.columns,
    class_names=['No Churn', 'Churn'],
    mode='classification'
)

instance_idx = 0
instance = X_test.iloc[instance_idx].values
```

```
explanation_lime = explainer.explain_instance(instance, log_clf.predict_proba, num_features=len(X.columns))
explanation_lime.show_in_notebook(show_table=True, show_all=False)
```

Figure 25: LIME on Logistic Regression Model

5.1.4 Feature Selection

Following the training of these baseline models and XAI application, the next step was to extract and select the top five features from each algorithm. Top ranked features and their importance scores were recorded in a new dataframe for each XAI approach. The top five features were selected as follows: top_5_features_importance, which includes the most important features from the feature importance analysis; top_5_features_shap, which ranks features based on their mean SHAP values; and top_5_features_LIME, which identifies the top features from the LIME analysis. These selected features were then prepared for retraining in experiments.

Random Forest: Select Top 5 Features

```
# Select top 5 features
top_5_features_importance = feature_importances_df.head(5)
top_5_features_shap = mean_shap_values_df.sort_values(by='Mean |SHAP Value|', ascending=False).head(5)
top_5_features_LIME = lime_features_df.head(5)
```

Figure 26: Selecting top 5 features from each XAI applied on Random Forest model

```
top_5_features_importance.head()
```

	Feature	Importance
0	TotalCharges	0.097101
1	tenure	0.088320
2	MonthlyCharges	0.079223
3	Contract_Month-to-month	0.072229
4	PaymentMethod_Electronic check	0.067569

```
top_5_features_LIME.head()
```

	Feature	Importance Value
0	Contract_Month-to-month <= 0.00	-0.158717
1	0.00 < OnlineSecurity_No <= 1.00	0.074065
2	0.00 < PaymentMethod_Electronic check <= 1.00	0.073717
3	0.00 < TechSupport_No <= 1.00	0.069181
4	0.00 < PaperlessBilling_Yes <= 1.00	0.049151

```
top_5_features_shap.head()
```

	Feature	Mean |SHAP Value|
0	Contract_Month-to-month	0.062125
1	PaymentMethod_Electronic check	0.048206
2	OnlineSecurity_No	0.043793
3	tenure	0.040466
4	TechSupport_No	0.037369

Figure 27: Top 5 features from each XAI applied on Random Forest model

XGBoost: Select Top 5 Features

```
# Select top 5 features from each XAI
top_5_features_importance_xgb = feature_importance_df_xgb.head(5)
top_5_features_shap_xgb = mean_shap_values_df_xgb.sort_values(by='Mean |SHAP Value|', ascending=False).head(5)
top_5_features_LIME_xgb = lime_features_df_xgb.head(5)
```

Figure 28: Selecting top 5 features from each XAI applied on XGBoost model

```
top_5_features_importance_xgb.head()
```

	Feature	Importance
1	MonthlyCharges	719.0
2	TotalCharges	649.0
0	tenure	392.0
3	gender_Female	146.0
31	PaperlessBilling_No	106.0

```
top_5_features_shap_xgb.head()
```

	Feature	Mean |SHAP Value|
0	Contract_Month-to-month	0.890333
1	tenure	0.654639
2	MonthlyCharges	0.464351
3	TotalCharges	0.384966
4	PaperlessBilling_Yes	0.338612

```
top_5_features_LIME_xgb.head()
```

	Feature	Importance Value
0	Contract_Month-to-month <= 0.00	-0.063309
1	0.00 < OnlineSecurity_No <= 1.00	0.026520
2	tenure > 0.85	-0.025281
3	0.00 < OnlineBackup_Yes <= 1.00	0.020094
4	gender_Female <= 0.00	-0.019692

Figure 29: Top 5 features from each XAI applied on XGBoost model

Logistic Regression: Select Top 5 Features

```
# Select top 5 features from each XAI
top_5_features_importance_log = feature_importance_df_log.head(5)
```

```
top_5_features_LIME_log = lime_features_df_log.head(5)
```

```
top_5_features_shap_log = mean_shap_values_df_log.sort_values(by='Mean |SHAP Value|', ascending=False).head(5)
```

Figure 30: Selecting top 5 features from each XAI applied on Logistic Regression model

```
top_5_features_LIME_log.head()
```

	Feature	Importance Value
0	tenure > 0.85	-0.477547
1	Contract_Month-to-month <= 0.00	-0.097497
2	Contract_Two year <= 0.00	0.083380
3	InternetService_Fiber optic <= 0.00	-0.044248
4	0.00 < InternetService_DSL <= 1.00	-0.039271

```
top_5_features_importance_log.head()
```

	Feature	Coefficient	Absolute Coefficient
0	tenure	-1.648048	1.648048
2	TotalCharges	1.042179	1.042179
37	Contract_Month-to-month	0.702564	0.702564
39	Contract_Two year	-0.670594	0.670594
17	InternetService_Fiber optic	0.358915	0.358915

```
top_5_features_shap_log.head()
```

	Feature	Mean |SHAP Value|
0	tenure	1.529266
1	TotalCharges	0.937313
2	Contract_Month-to-month	0.349949
3	Contract_Two year	0.235226
4	InternetService_Fiber optic	0.177803

Figure 31: Top 5 features from each XAI applied on Logistic Regression model

6 Experiments

6.1 Experiment 1: Retraining with Random Forest

The code snippets below demonstrate the process of retraining the Random Forest model with features selected by LIME, SHAP, and Feature Importance. To determine if selected features lead to improved model accuracy and interpretability.

Random Forest Retrain: Feature Important

```
# Using top 5 features from feature importance
top_5_features_fi_list = top_5_features_importance['Feature'].tolist()
X_train_fi = X_train_res[top_5_features_fi_list]
X_test_fi = X_test[top_5_features_fi_list]

clf_fi = RandomForestClassifier(random_state=42)
clf_fi.fit(X_train_fi, y_train_res)
y_pred_fi = clf_fi.predict(X_test_fi)

print("Random Forest using top 5 features from feature importance")
print(classification_report(y_test, y_pred_fi))
print(f"Accuracy: {accuracy_score(y_test, y_pred_fi):.4f}")
```

```
Random Forest using top 5 features from feature importance
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      1546
           1       0.52      0.55      0.53       557

    accuracy                           0.75      2103
   macro avg       0.68      0.68      0.68      2103
weighted avg       0.75      0.75      0.75      2103
```

Figure 32: Retrain Random Forest with top features from Feature Important

Random Forest Retrain: SHAP

```
# Using top 5 features from SHAP values
top_5_features_shap_list = top_5_features_shap['Feature'].tolist()
X_train_shap = X_train_res[top_5_features_shap_list]
X_test_shap = X_test[top_5_features_shap_list]

clf_shap = RandomForestClassifier(random_state=42)
clf_shap.fit(X_train_shap, y_train_res)
y_pred_shap = clf_shap.predict(X_test_shap)

print("\nRandom Forest using top 5 features from SHAP values")
print(classification_report(y_test, y_pred_shap))
print(f"Accuracy: {accuracy_score(y_test, y_pred_shap):.4f}")
```

```
Random Forest using top 5 features from SHAP values
              precision    recall  f1-score   support

           0       0.84      0.81      0.82      1546
           1       0.52      0.58      0.55       557

    accuracy                           0.75      2103
   macro avg       0.68      0.69      0.68      2103
weighted avg       0.76      0.75      0.75      2103
```

Figure 33: Retrain Random Forest with top features from SHAP

Random Forest Retrain: LIME

```
# Using top 5 features from LIME explanation
top_5_features_lime_list = top_5_features_LIME['Feature'].tolist()
X_train_lime = X_train_res[top_5_features_lime_list]
X_test_lime = X_test[top_5_features_lime_list]

clf_lime = RandomForestClassifier(random_state=42)
clf_lime.fit(X_train_lime, y_train_res)
y_pred_lime = clf_lime.predict(X_test_lime)

print("\nRandom Forest using top 5 features from LIME explanation")
print(classification_report(y_test, y_pred_lime))
print(f"Accuracy: {accuracy_score(y_test, y_pred_lime):.4f}")
```

```
Random Forest using top 5 features from LIME explanation
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      1546
           1       0.57      0.57      0.57       557

    accuracy                           0.77      2103
   macro avg       0.71      0.71      0.71      2103
weighted avg       0.77      0.77      0.77      2103
```

Figure 34: Retrain Random Forest with top features from LIME

6.2 Experiment 2: Retraining with XGBoost

XGBoost Retrain: Feature Important

```
# Using top 5 features from feature importance
top_5_features_fi_list = top_5_features_importance_xgb['Feature'].tolist()
X_train_fi = X_train_res[top_5_features_fi_list]
X_test_fi = X_test[top_5_features_fi_list]

xgb_clf_fi = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb_clf_fi.fit(X_train_fi, y_train_res)
y_pred_fi = xgb_clf_fi.predict(X_test_fi)

print("XGBoost using top 5 features from feature importance")
print(classification_report(y_test, y_pred_fi))
print(f"Accuracy: {accuracy_score(y_test, y_pred_fi):.4f}")
```

```
XGBoost using top 5 features from feature importance
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      1546
           1       0.52      0.53      0.53       557

    accuracy                           0.75      2103
   macro avg       0.68      0.68      0.68      2103
weighted avg       0.75      0.75      0.75      2103
```

Figure 35: Retrain XGBoost with top features from Feature Important

XGBoost Retrain: SHAP

```
# Using top 5 features from SHAP values
top_5_features_shap_list = top_5_features_shap_xgb['Feature'].tolist()
X_train_shap = X_train_res[top_5_features_shap_list]
X_test_shap = X_test[top_5_features_shap_list]

xgb_clf_shap = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb_clf_shap.fit(X_train_shap, y_train_res)
y_pred_shap = xgb_clf_shap.predict(X_test_shap)

print("XGBoost using top 5 features from SHAP values")
print(classification_report(y_test, y_pred_shap))
print(f"Accuracy: {accuracy_score(y_test, y_pred_shap):.4f}")
```

```
XGBoost using top 5 features from SHAP values
              precision    recall  f1-score   support

           0       0.86      0.82      0.84      1546
           1       0.55      0.62      0.58       557

    accuracy                           0.76      2103
   macro avg       0.70      0.72      0.71      2103
weighted avg       0.78      0.76      0.77      2103
```

Figure 36: Retrain XGBoost with top features from SHAP

XGBoost Retrain: LIME

```
# Using top 5 features from LIME explanation
top_5_features_lime_list = top_5_features_LIME_xgb['Feature'].tolist()
X_train_lime = X_train_res[top_5_features_lime_list]
X_test_lime = X_test[top_5_features_lime_list]

xgb_clf_lime = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb_clf_lime.fit(X_train_lime, y_train_res)
y_pred_lime = xgb_clf_lime.predict(X_test_lime)

print("XGBoost using top 5 features from LIME explanation")
print(classification_report(y_test, y_pred_lime))
print(f"Accuracy: {accuracy_score(y_test, y_pred_lime):.4f}")
```

```
XGBoost using top 5 features from LIME explanation
              precision    recall  f1-score   support

           0       0.84      0.81      0.83      1546
           1       0.53      0.58      0.55       557

    accuracy                           0.75      2103
   macro avg       0.69      0.70      0.69      2103
weighted avg       0.76      0.75      0.75      2103
```

Figure 37: Retrain XGBoost with top features from LIME

6.3 Experiment 3: Retraining with Logistic Regression

Logistic Regression Retrain: Feature Important

```
# Using top 5 features from feature importance
top_5_features_fi_list = top_5_features_importance_log['Feature'].tolist()
X_train_fi = X_train_res[top_5_features_fi_list]
X_test_fi = X_test[top_5_features_fi_list]

log_clf_fi = LogisticRegression(random_state=42, max_iter=1000)
log_clf_fi.fit(X_train_fi, y_train_res)
y_pred_fi = log_clf_fi.predict(X_test_fi)

print("XGBoost using top 5 features from feature importance")
print(classification_report(y_test, y_pred_fi))
print(f"Accuracy: {accuracy_score(y_test, y_pred_fi):.4f}")
```

```
XGBoost using top 5 features from feature importance
              precision    recall  f1-score   support

           0       0.93      0.69      0.79      1546
           1       0.50      0.85      0.63       557

    accuracy                           0.73      2103
   macro avg       0.71      0.77      0.71      2103
weighted avg       0.81      0.73      0.75      2103

Accuracy: 0.7332
```

Figure 38: Retrain Logistic Regression with top features from Feature Important

Logistic Regression Retrain: SHAP

```
# Using top 5 features from SHAP values
top_5_features_shap_list = top_5_features_shap_log['Feature'].tolist()
X_train_shap = X_train_res[top_5_features_shap_list]
X_test_shap = X_test[top_5_features_shap_list]

log_clf_shap = LogisticRegression(random_state=42, max_iter=1000)
log_clf_shap.fit(X_train_shap, y_train_res)
y_pred_shap = log_clf_shap.predict(X_test_shap)

print("XGBoost using top 5 features from SHAP values")
print(classification_report(y_test, y_pred_shap))
print(f"Accuracy: {accuracy_score(y_test, y_pred_shap):.4f}")
```

```
XGBoost using top 5 features from SHAP values
              precision    recall  f1-score   support

           0       0.93      0.69      0.79      1546
           1       0.50      0.85      0.63       557

    accuracy                           0.73      2103
   macro avg       0.71      0.77      0.71      2103
weighted avg       0.81      0.73      0.75      2103

Accuracy: 0.7332
```

Figure 39: Retrain Logistic Regression with top features from SHAP

Logistic Regression Retrain: LIME

```
# Using top 5 features from LIME explanation
top_5_features_lime_list = top_5_features_LIME_log['Feature'].tolist()
X_train_lime = X_train_res[top_5_features_lime_list]
X_test_lime = X_test[top_5_features_lime_list]

log_clf_lime = LogisticRegression(random_state=42, max_iter=1000)
log_clf_lime.fit(X_train_lime, y_train_res)
y_pred_lime = log_clf_lime.predict(X_test_lime)

print("XGBoost using top 5 features from LIME explanation")
print(classification_report(y_test, y_pred_lime))
print(f"Accuracy: {accuracy_score(y_test, y_pred_lime):.4f}")
```

```
XGBoost using top 5 features from LIME explanation
              precision    recall  f1-score   support

           0       0.92      0.69      0.79      1546
           1       0.50      0.84      0.63       557

    accuracy                           0.73      2103
   macro avg       0.71      0.77      0.71      2103
weighted avg       0.81      0.73      0.75      2103

Accuracy: 0.7337
```

Figure 40: Retrain Logistic Regression with top features from LIME

References