Predicting Customer Churn: Machine Learning Solutions for Banking Retention

Table of Contents

Table of Tables

Table of Figures

1.1 Introduction

1.1.1 Background

Customer loyalty is important for most businesses, but the sad reality is that customers are hardly loyal. Predicting customer churn becomes a necessity for businesses wanting to maintain their customer base. The ability to predict customers at risk of leaving allows businesses to take proactive measures to retain them. Predictive machine learning models can be used in identifying at-risk customers and significantly reduce churn.

This study aims to develop a predictive model for customer churn and identify factors contributing to churn in the banking sector. Using algorithms like K-Nearest Neighbours, Random Forest and Artificial Neural Network and implementing them in both Python and Azure machine learning studio. The findings of this research should provide actionable insights and strategic recommendations that can help banks and businesses reduce churn rates.

1.1.2 Research Question

How can machine learning models like KNN, random forest and neural networks be used to predict customer churn in the banking sector, and which models are more effective?

1.1.3 Relevant Works

Several studies have applied machine learning models for predicting customer churn across various industries. For example, Lalwani et al. (2021) combined machine learning algorithms such as SVM, LR, and random forest in predicting customer churn. Vafeiadis et al. (2015), compared various machine learning techniques in customer churn prediction, their findings discovered that boosted models performed better than non-boosted models. Bilal (2016) highlighted that neural networks are effective in predicting churn in the banking sector.

This research aims to contribute to the body of work in churn prediction by applying the feature importances, model comparison and evaluation metrics to further refine the model and provide actionable business insights and recommendations.

1.1.4 Methodology

This study adopts the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology, which provides a structured and systematic approach to handling data mining tasks (Larose & Larose, 2015). This process is divided into five phases:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modelling
5. Evaluation.

1.2 Dataset

The dataset used in this study was obtained from Lloyds Bank's Forage program, a platform that provides virtual experiences from top companies. It was in the form of an Excel sheet with 5 spreadsheets containing customer demographics data, transaction history data, customer service data, online activity data and churn status data of customers.

Customer Demographics: Unique information describing 1000 bank customers, with 5 descriptors.

Table 2.1 Description of Customer Demographics Data

FEATURE	DESCRIPTION
CUSTOMER ID	Customer Identification
AGE	Age of customer
GENDER	Gender of Customer (Male or Female)
MARITAL STATUS	Married, Single, Widowed or Divorced
INCOME LEVEL	Customers income level (Low, Medium, or High)

Transaction History: contains information on the customer's transaction with the bank, with 5054 observations and 5 features.

Table 2.2 Description of Transaction History Data

FEATURE	DESCRIPTION
CUSTOMER ID	Customer Identification
TRANSACTION ID	Unique identifier for the transaction
TRANSACTION DATE	Date transaction occurred
AMOUNT SPENT	Amount in pounds the customer spent
PRODUCT CATEGORY	Type of product customer purchased (i.e. electronic, clothing)

Customer Service: information about customer interactions with the bank's customer service, with 1002 observations and 5 features.

Table 2.3 Description of Customer Service Data

FEATURE	DESCRIPTION
CUSTOMER ID	Customer Identification
INTERACTION ID	Unique identification of the interaction
INTERACTION DATE	Date interaction occurred
INTERACTION TYPE	Reason for interaction (inquiry, complaint, or feedback)
RESOLUTION STATUS	State of the interaction (resolved or unresolved)

Online Activity: information about each customer's online interaction with the bank, with 1000 observations and 4 features.

Table 2.4 Description of Online Activity Data

FEATURE	DESCRIPTION
CUSTOMER ID	Customer Identification
LAST LOGIN DATE	Date customer logged in to bank last
LOGIN FREQUENCY	Frequency at which customer logs in
SERVICE USAGE	Method of online interaction (i.e. Mobile app, website)

Churn status: information on customer status with the bank, with 1000 observations and 2 features.

Table 2.5 Description of Churn Status Data

FEATURE	DESCRIPTION
CUSTOMER ID	Customer Identification
CHURN STATUS	If customer has left the bank or inactive

All five sheets were imported and merged to form a single datasheet containing all the customer information. This dataset consists of 6812 observations and 17 features.

```
# Merging all the dataframes.
bank_df = pd.merge(customer_demographics_df, transaction_history_df, on ='CustomerID', how = 'outer' )
bank_df = pd.merge(bank_df, customer_service_df, on ='CustomerID', how = 'outer' )
bank_df = pd.merge(bank_df, online_activity_df, on ='CustomerID', how = 'outer' )
bank_df = pd.merge(bank_df, churn_status_df, on ='CustomerID', how = 'outer' )
```

```
# checking amount of rows and columns
bank_df.shape
```

```
(6812, 17)
```

Figure 2.1 Code and Result for Merging Bank Data

1.2.1 Ethical, Social, Legal Consideration

The datasets adhere to the GDPR and are anonymized mitigating privacy concerns.

1.3 Exploratory Data Analysis and Preprocessing

Exploring and analysing the features in a dataset is essential for gaining a proper understanding and developing a foundation for the modelling stage. This process will include data cleaning and visualizations that provide valuable insights.

1.3.1 Data Cleaning

Selecting Relevant Features: features that provide little to no information as to the reason for churn were dropped.

```
# Anonymizing and dropping irrelevant columns
bank_df = bank_df.drop(['CustomerID','TransactionID','TransactionDate','InteractionID',
                        'InteractionDate','LastLoginDate'], axis=1)
```

Figure 2.2 Code for dropping irrelevant columns

Duplicate Handling: The first step is to check for and handle duplicated observations appropriately. A total of 284 duplicates were found and removed.

```
#checking for duplicates
bank_df.duplicated().sum()
```

```
284
```

```
# dropping the duplicates
bank_df = bank_df.drop_duplicates()
print(f' Duplicates: {bank_df.duplicated().sum()}')
bank_df.shape
```

```
Duplicates: 0
(6528, 11)
```

Figure 2.3 Code and output for handling duplicate values

Handling Missing Values: Missing values can raise significant issues during the modelling process if not handled properly. Proper investigation and understanding of the dataset are required to handle missing values. Approximately 24.6% of the interaction type and resolution status is missing, but there were no missing values in the individual dataset.

```
#checking for the percentage of missing values
bank_df.isna().sum()/len(bank_df) * 100
```

```
Age                 0.000000
Gender              0.000000
MaritalStatus       0.000000
IncomeLevel         0.000000
AmountSpent         0.000000
ProductCategory     0.000000
InteractionType    24.632353
ResolutionStatus   24.632353
LoginFrequency      0.000000
ServiceUsage        0.000000
ChurnStatus         0.000000
```

Figure 2.4 Checking for missing values

The missing values are most likely due to customers who did not interact with customer service at all, thereby not having any records of doing so. The missing values in interaction type were imputed with "None", to describe customers who did not interact with customer service and resolution was imputed with "No issue".

```
# Handling missing values in interaction type
interactiontype_imputer = SimpleImputer(strategy='constant', fill_value='None')
bank_df[['InteractionType']] = interactiontype_imputer.fit_transform(bank_df[['InteractionType']])
```

```
# Handling missing values in Resolution Status
resolution_imputer = SimpleImputer(strategy='constant', fill_value='No Issue')
bank_df[['ResolutionStatus']] = resolution_imputer.fit_transform(bank_df[['ResolutionStatus']])
```

Figure 2.5 Code handling missing values

Formatting Datatypes: Each feature must be in the appropriate datatype (e.g. categorical columns formatted as category and numerical columns formatted as float64 or int64).

```
# numerical columns
numerical_cols = ['Age','AmountSpent','LoginFrequency']
```

```
# nominal columns
nominal_cols = ['Gender','MaritalStatus','ProductCategory','InteractionType','ResolutionStatus','ServiceUsage']
```

```
# ordinal column
ordinal_col = ['IncomeLevel']
```

```
# Target column
target = ['ChurnStatus']
```

```
# converting the data type of the nominal columns to categorical.
for col in nominal_cols:
    bank_df[col] = pd.Categorical(bank_df[col], ordered=False)
```

```
# converting the data type of the ordinal column to ordered categories
bank_df['IncomeLevel'] = pd.Categorical(bank_df['IncomeLevel'], categories=['Low', 'Medium', 'High'], ordered=True)
```

Figure 2.6 Code for formatting data types

1.3.2 Visualization

1.3.2.1 Univariate Analysis

ChurnStatus (Target): The pie chart shows that 80% of the bank's customers haven't churned. This distribution suggests that the classes are imbalanced and would need to be addressed.

```
# Pie chart to visualize target.
plt.figure(figsize=(10, 5))
bank_df['ChurnStatus'].value_counts(normalize=True).plot.pie(
    autopct='%1.1f%%',
    startangle=60,
    colors=['lightblue', 'darkblue'],
    labels=['Not Churned', 'Churned'],
    explode=[0.05, 0.05])
plt.title('Distribution of Churn Status')
plt.ylabel('')
plt.show()
```

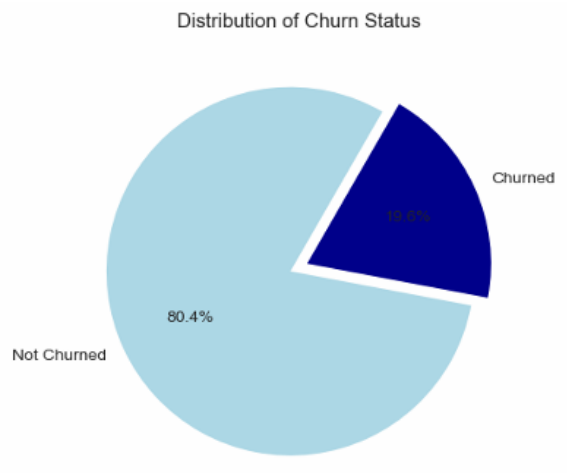

Figure 2.7 Distribution of target variable (ChurnStatus)

Distribution of numerical features: each numerical feature was visualized to understand its distribution. All the distributions appear evenly distributed.

```
# plotting distribution of numeriacal features
fig, axes = plt.subplots(1, 3, figsize=(10, 3))
for i, col in enumerate(numerical_cols):
    sns.histplot(bank_df[col], kde=True, ax=axes[i],edgecolor='black',color='#66C2A5')
    axes[i].set_title(f'Distribution of {col}')

plt.tight_layout(w_pad=3, pad=2)
plt.show()
```

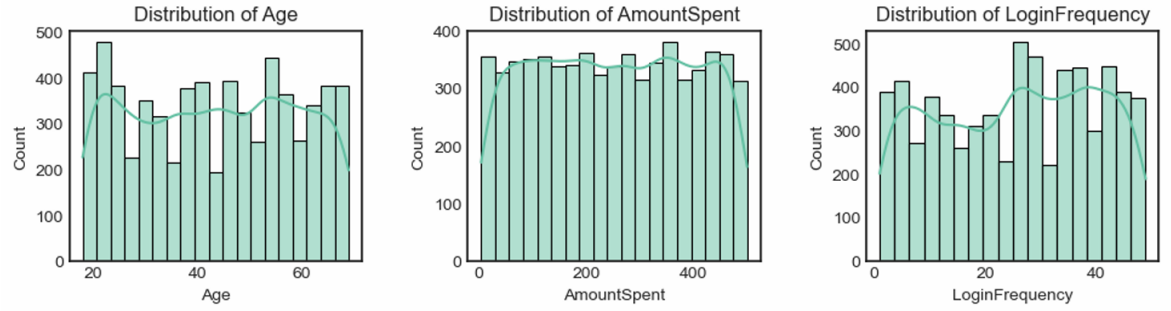

Figure 2.8 Distribution of Age, AmountSpent and LoginFrequency

Distribution of categorical features: All the features appear to be distributed evenly.

```
# distribution for age and marital status
fig, axes = plt.subplots(1,2, figsize=(8, 3))
axes = axes.flatten()
for i, col in enumerate(nominal_cols[:2]):
    sns.countplot(bank_df, x=col, ax=axes[i], color='#66C2A5')
    axes[i].set_title(f'Distribution of {col}')
plt.tight_layout(w_pad=3, pad=2)
plt.show()
```

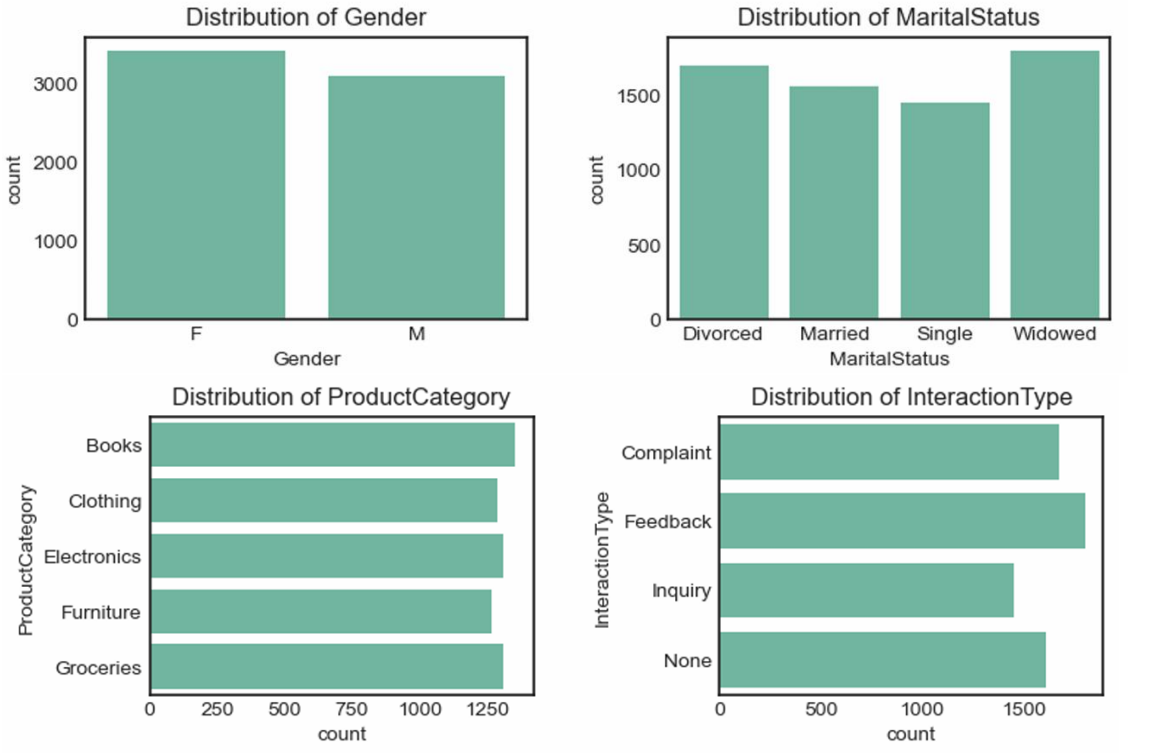

Figure 2.9 Distribution of Gender, MaritalStatus, ProductCategory and InteractionType

Figure 2.10 Distribution of ResolutionStatus and ServiceUsage

1.3.2.2 Bivariate Analysis

Distribution of numerical features and churn status: There seems to be a relationship between customer age, and login frequency with churn. The median age of customers who churn is higher than customers who stay.

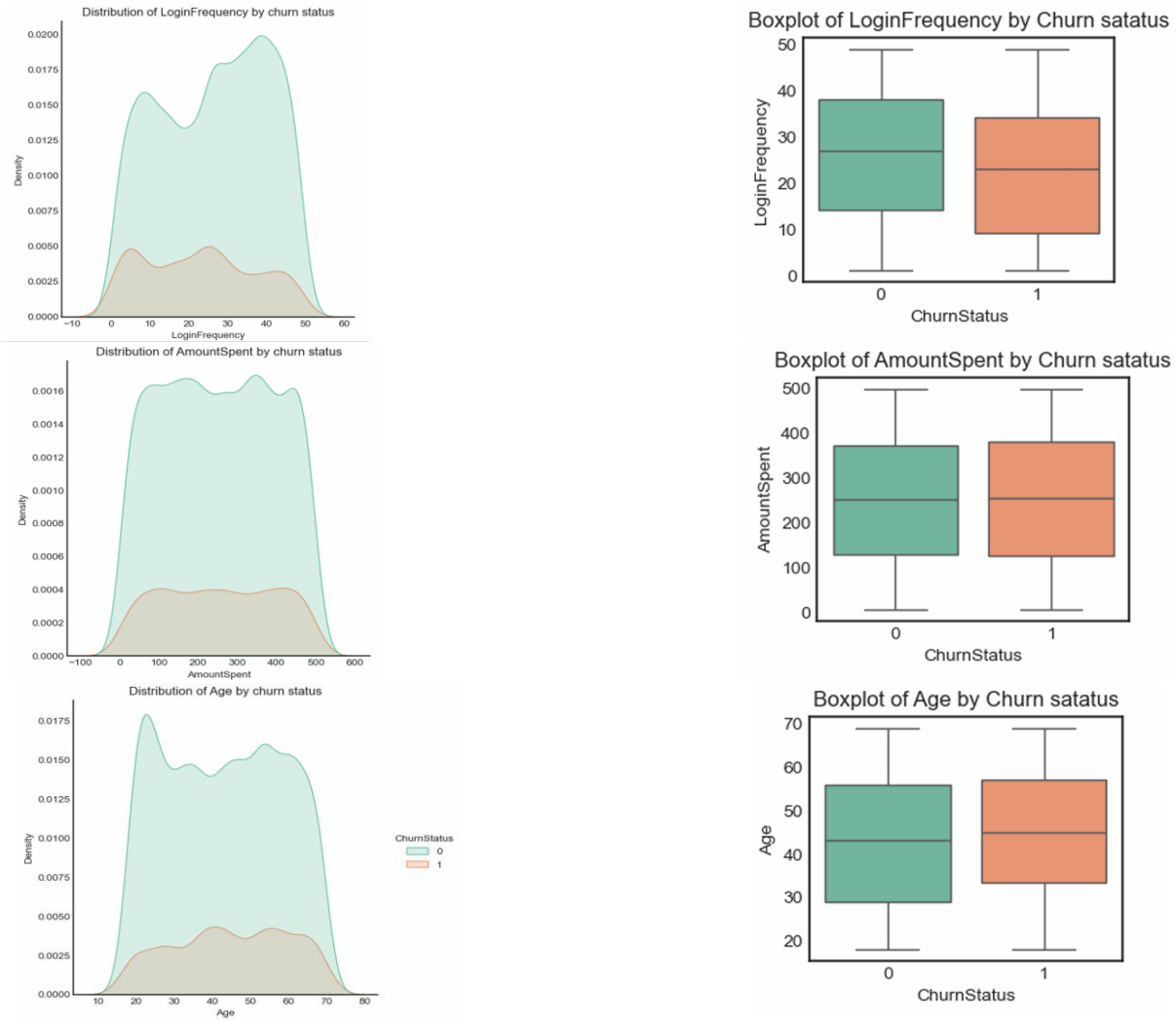

Figure 2.11 Distribution of numerical features with churn status

Distribution of categorical features and churn status: Customers using the Mobile app tend to churn more than other service users, and customers with complaints and feedback churn more than others.

```
# distribution for Resolution status  and Service usage by churn
fig, axes = plt.subplots(1,2, figsize=(8, 3))
axes = axes.flatten()
for i, col in enumerate(nominal_cols[4:6]):
    sns.countplot(bank_df, x=col, ax=axes[i], hue='ChurnStatus', palette='Set2')
    axes[i].set_title(f'Distribution of {col} by churn status')
plt.tight_layout(w_pad=3, pad=2)
plt.show()
```

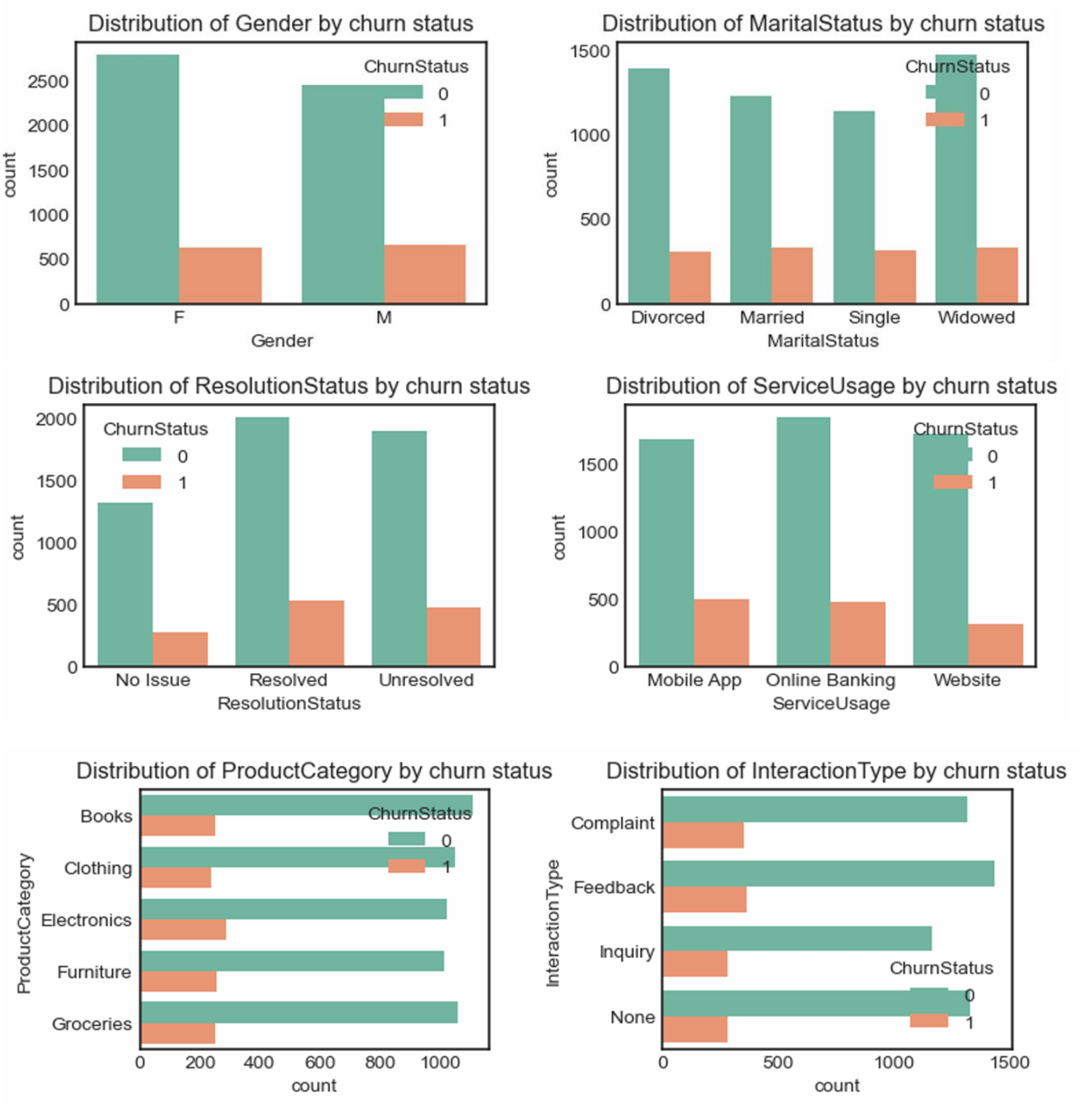

Figure 2.12 Distribution of categorical features and churn status

1.3.2.3 Multivariate Analysis

Correlation: age, amount spent, and login frequency aren't correlated.

```
# correlation analysis of numerical features
plt.figure(figsize=(6, 3))
sns.heatmap(bank_df[numerical_cols].corr(), annot=True, fmt='.3f', cmap='Blues')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

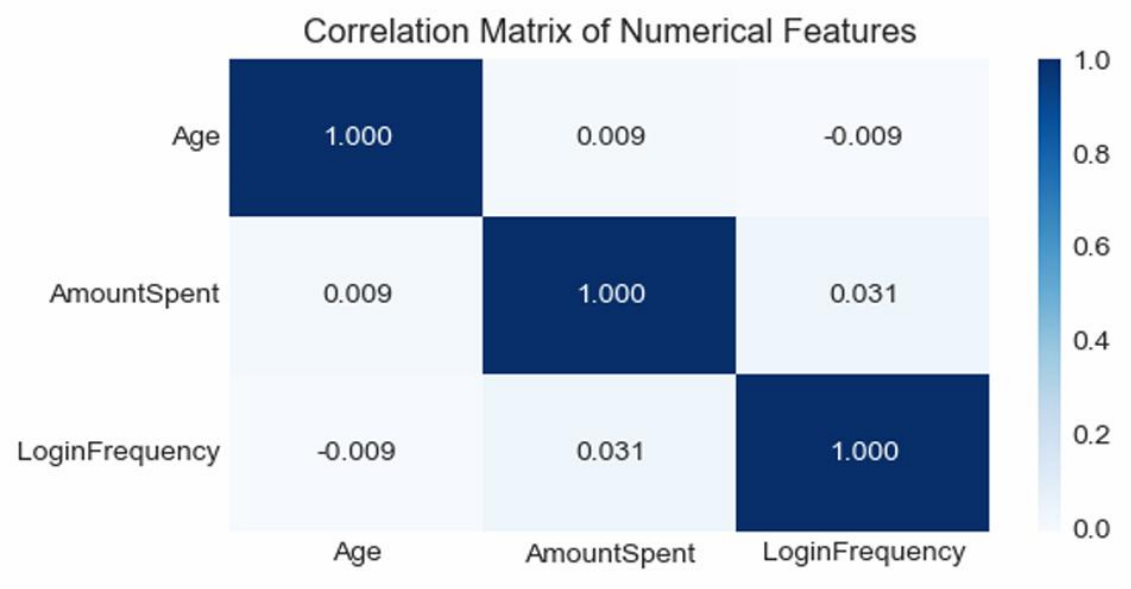

Figure 2.13 Correlation of numerical features

1.3.3 Preprocessing

To enhance the quality of the model, a few transformations and processing are required.

Assignment: The variables were assigned to their designated categories i.e. features (independent variables) and target (dependent variable). All features except for churn status were assigned to X, while churn status, was assigned to y.

```
# spliiting the target feature from the data
X = bank_df.drop('ChurnStatus', axis=1)
y = bank_df['ChurnStatus']
```

Figure 2.14 Code assigning target and features

Train-Test Split: the features and models were split into train and test sets. The train set contained 70% of the data chosen at random but stratified to make sure there is a similar distribution in both train and test sets.

```
# splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3,
    random_state=random,
    stratify=y)
```

Figure 2.15 Code splitting the data into train and test set

Standardization: Numerical features in the train and test were standardized because of differences in scale and units. StandardScaler was used to transform the data to have a mean and standard deviation of approximately 0 and 1 respectively.

```
# Scaling numerical features
X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])
```

Figure 2.16 Code for normalizing the numerical features

Encoding: The nominal and ordinal features in both train and test sets were encoded to numerical using OneHotEncoder and OrdinalEncoder.

```
# Instantiating OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories= [['Low', 'Medium', 'High']])
```

```
# Ordinal Encoding the ordinal columns- Using OrdinalEncoder
X_train[['IncomeLevel']] = ordinal_encoder.fit_transform(X_train[['IncomeLevel']])
X_test[['IncomeLevel']] = ordinal_encoder.transform(X_test[['IncomeLevel']])
```

Figure 2.17 Code for using an ordinal encoder to encode the ordinal column (IncomeLevel)

```
# Instantiating the encoder
ohe = OneHotEncoder(drop='first', sparse_output=False)
```

```
# Encoding train and test categorical features
nominal_train = ohe.fit_transform(X_train[nominal_cols])
nominal_test = ohe.transform(X_test[nominal_cols])
```

Figure 2.18 Code for using one hot encoder to encode the nominal columns

Class Imbalance: The EDA uncovered that the target variable (ChurnStatus) is overly dominated by one class. The synthetic Minority Oversampling technique (SMOTE) was used to balance the train set to achieve better model results.

```
# instantiating the oversampler
smote = SMOTE(random_state=random)
```

```
# Resampling the model
X_train, y_train = smote.fit_resample(X_train,y_train)
```

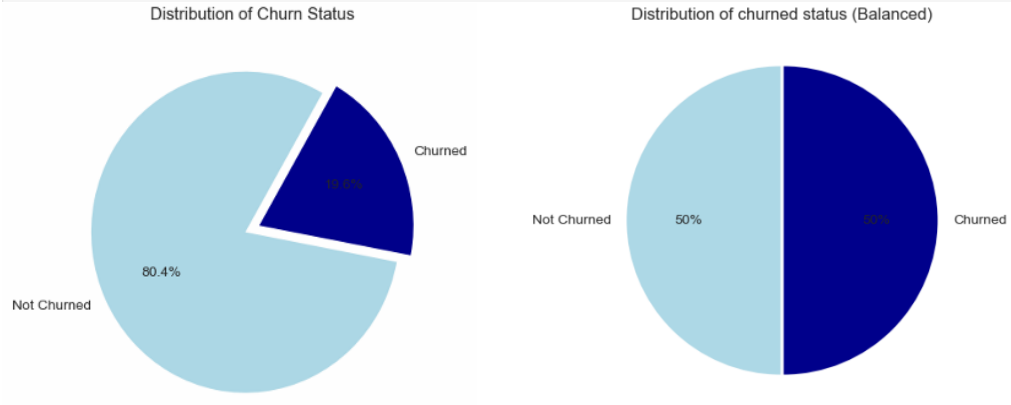

Figure 2.19 Using SMOTE to treat class imbalance

Feature Selection: RFECV (Recursive feature elimination, cross-validated) was used to select the best subsets of features needed to train the model. Random Forest was chosen as the estimator and f1-score was used as the evaluation criteria during cross-validation.

```
# Instantiating the Random Forest model as the estimator for RFECV
rf = RandomForestClassifier(random_state=random)
```

```
# Instantiating the RFECV for feature selection
rfecv = RFECV(estimator=rf,
    step=4,
    cv=3,
    scoring='f1'
)
rfecv.fit(X_train, y_train)
```

Figure 2.20 Code applying RFECV for feature selection

```
optimal_features = rfecv.n_features_
selected_features = X_train.columns[rfecv.support_]
```

```
optimal_features
```

```
19
```

Figure 2.21 The optimal number of features to train the model

Figure 1.21 shows that there are 19 optimal features, which indicates that no features were removed.

1.4 Implementation

KNN (K-Nearest Neighbours) and Random Forest algorithms were chosen for this classification task due to their different approach to handling classification tasks. KNN classifies data points based on the majority labels of its nearest neighbours, while Random Forest is an ensemble algorithm that aggregates the output of multiple decision trees (Gohil et al., 2023).

1.4.1 Python

1.4.1.1 KNN (K-Nearest Neighbours)

The KNN model was chosen for its simplicity, interpretability and effectiveness in classification tasks.

Determining the optimal number of neighbours: The KNN model was instantiated and trained on the pre-processed data by varying the number of neighbours from 1 to 12. For each k-value, the model's prediction performance was evaluated by using the F1-score metric. The value of K with the highest f1-score was selected as the optimal number of neighbours.

```
# Finding the optimal number of neighbours
f1score = []
k=12
for i in range(1,k+1):
    knn_model_2 = KNeighborsClassifier(n_neighbors=i, metric='minkowski', p=2)
    knn_model_2.fit(X_train, y_train)
    f1score.append(f1_score(y_test, knn_model_2.predict(X_test)))
```

Figure 2.22 Code to find optimal number of neighbours

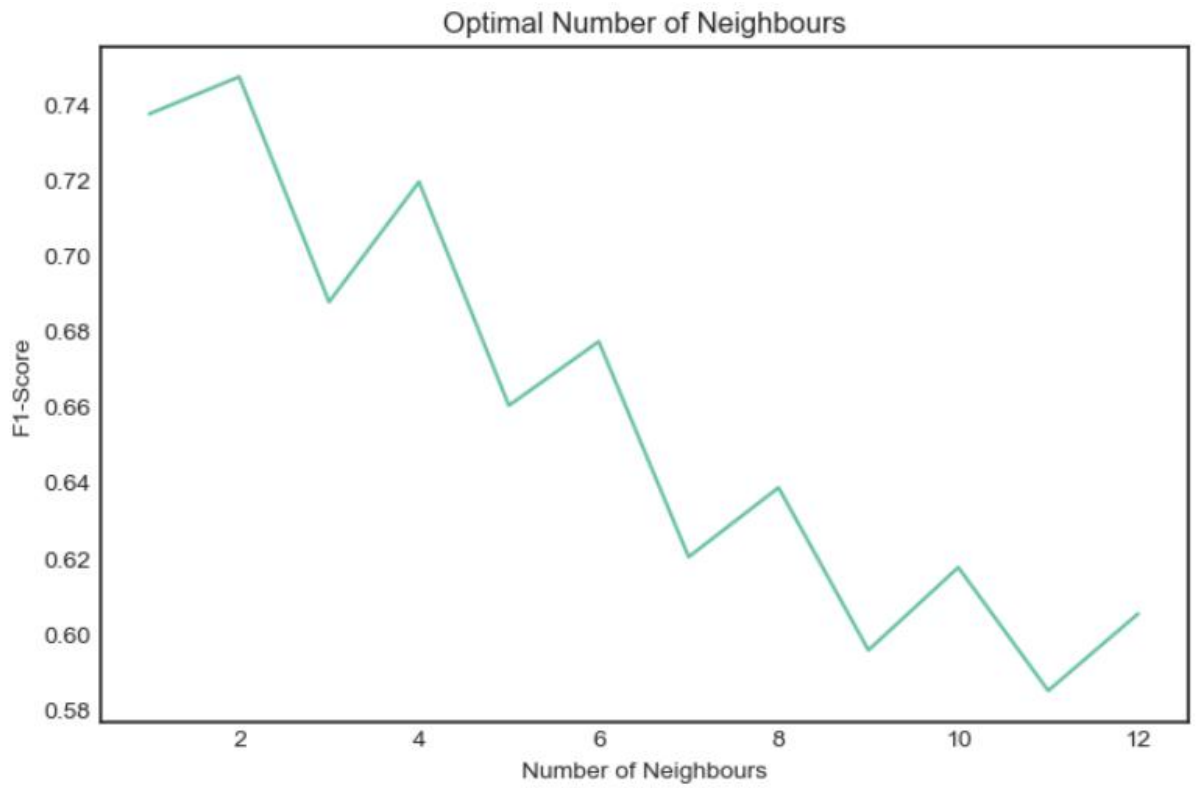

Figure 2.23 Line chart showing the optimal number of neighbours

Hyperparameter tuning:

- **n_neighbors:** the optimal number of neighbours was identified in Figure 2.23 as 2
- **metric:** The Minkowski distance was chosen with p=2 for Euclidean distance because it is the most used metric to measure the proximity between the data points.

```
# the optimal model has 2 neighhors
knn_model_optimal = KNeighborsClassifier(n_neighbors=2, metric='minkowski', p=2)
knn_model_optimal.fit(X_train, y_train)
y_pred_knn_optimal = knn_model_optimal.predict(X_test)
```

Figure 2.24 Code used to train the KNN model

Predictions: model predictions can be compared with the actual values by using a confusion matrix. This confusion matrix shows that the model correctly predicted the following in each class:

- Not churned (0): 1481 out of 1574 observations
- Churned (1): 285 out of 385 observations

```
# plotting the confusion matrix
knn_cm = confusion_matrix(y_test, y_pred_knn_optimal)
plt.figure(figsize=(8,4))
sns.heatmap(knn_cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Bank Churn classification')
plt.show()
```

Figure 2.25 Code for plotting the confusion matrix of the KNN model

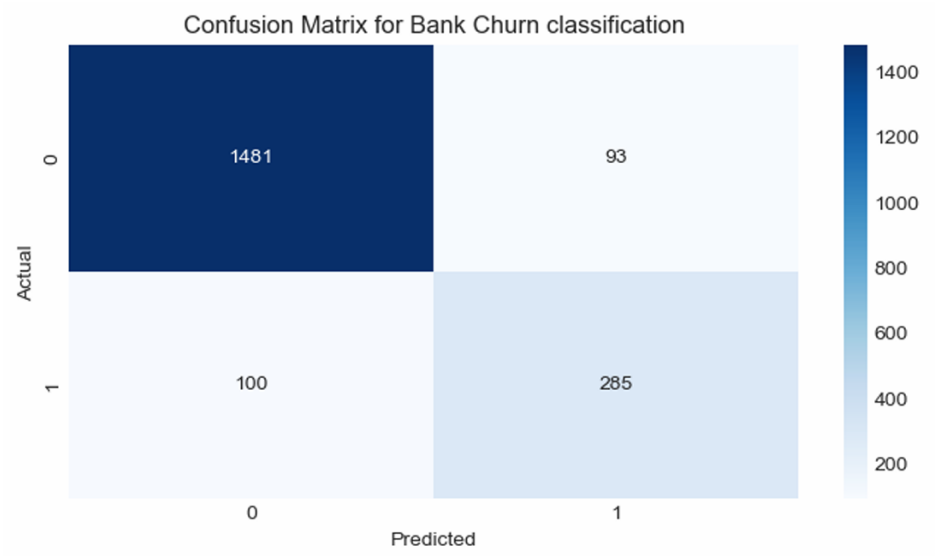

Figure 2.26 Confusion matrix of the KNN model, Predicting churn

1.4.1.2 Random Forest

The random forest model has a track record of performing better compared to other classifiers (Xie et al., 2009). It was selected for its flexibility and performance in handling structured data and its inherent feature importance score providing insight into factors affecting churn.

Implementation

The random forest model was trained on the pre-processed data (X_train and y_train) by instantiating and fitting the RandomForestClassifer with a random state value of 123 ensuring the reproducibility of results. To enhance the model's performance, the following hyperparameters were tuned:

- n_estimators: the number of trees of the random forest model was set as 250 and 260.
- max_depth: maximum depth of trees was set as 10 and 20

The hyperparameters were tuned using GridSearchCV, a technique that finds the combinations of hyperparameters that produce the best performance, using cross-validation to evaluate each combination. The implementation steps are shown below:

1. Instantiation: The GridSearchCV model was instantiated with a random forest model as its estimator and the predefined hyperparameters.
2. Cross-Validation: the GridSearchCV model was set to perform 5 cross-validations.
3. Evaluation: f1-score was used to evaluate model performance during this process, and the best hyperparameter was chosen based on this performance
4. Result: The optimal hyperparameters are applied to the model, which was then re-trained.

```
# Insatntiating random forest model
rf_model = RandomForestClassifier(random_state=random)
```

```
parameters = {
    'n_estimators': [250,260],
    'max_depth' : [10,20]
}

# instantiating GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=parameters , cv=5, verbose=1, n_jobs=-1, scoring='f1')
grid_search.fit(X_train, y_train)
```

Figure 2.27 Hyperparameter tuning code, using GridSearchCV

```
grid_search.best_params_
```

```
{'max_depth': 20, 'n_estimators': 250}
```

```
# Predict on the test optimal X_test data for random forest
y_pred_rf_tuned = grid_search.predict(X_test)
```

Figure 2.28 Using the pest parameters of the Random Forest model to predict churn

Predictions: The model correctly classified 291 out of 385 churn cases and 1561 out of 1574 no-churn cases.

```
# plotting the confusion matrix
rf_cm_tuned = confusion_matrix(y_test, y_pred_rf_tuned)
plt.figure(figsize=(8, 4))
sns.heatmap(rf_cm_tuned, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Bank Churn classification')
plt.show()
```

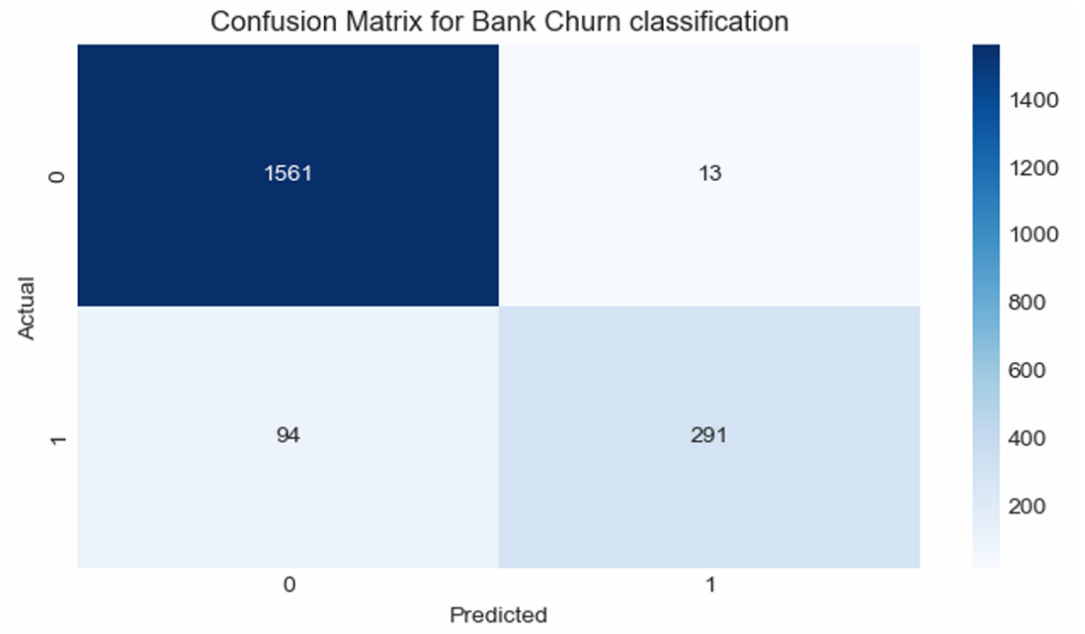

Figure 2.29 Confusion matrix showing classification results for Random Forest Model

1.4.2 Implementation in Azure Machine Learning

Random forest and Artificial Neural Networks (ANN) were the algorithms selected for implementation in Azure ML. The cleaned merged bank data without missing values and duplicates was uploaded to Azure ML studio and a design pipeline was created where the data was loaded.

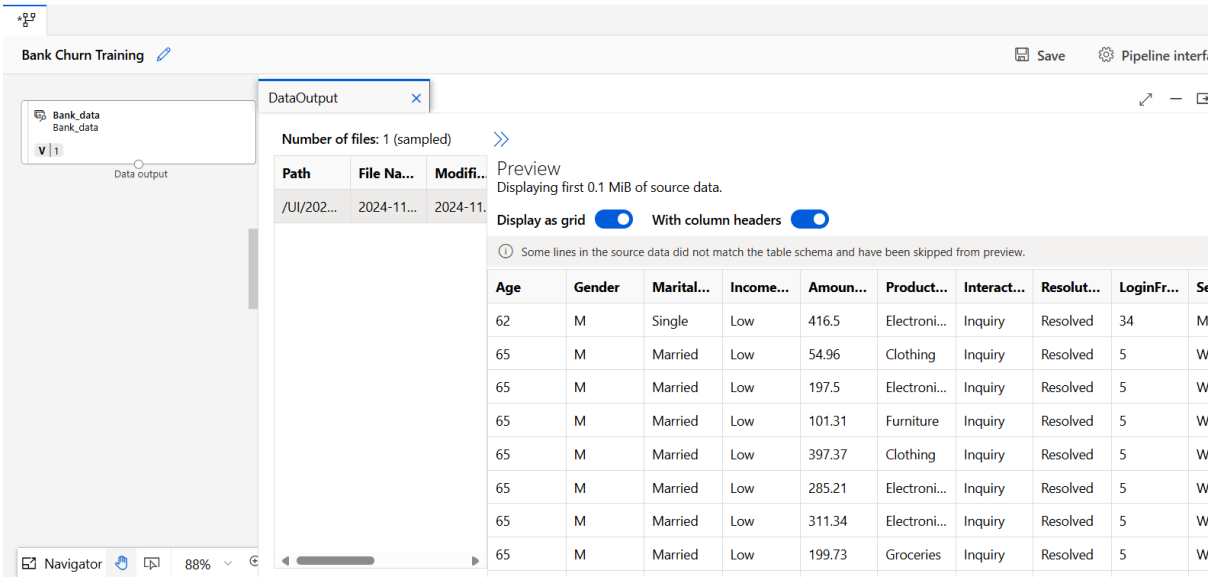

Figure 2.30 loaded data preview in Azure ML Studio

1.4.2.1 Preprocessing

Standardization: Numerical features were standardised using ZScore, which works like StandardScaler in Python.

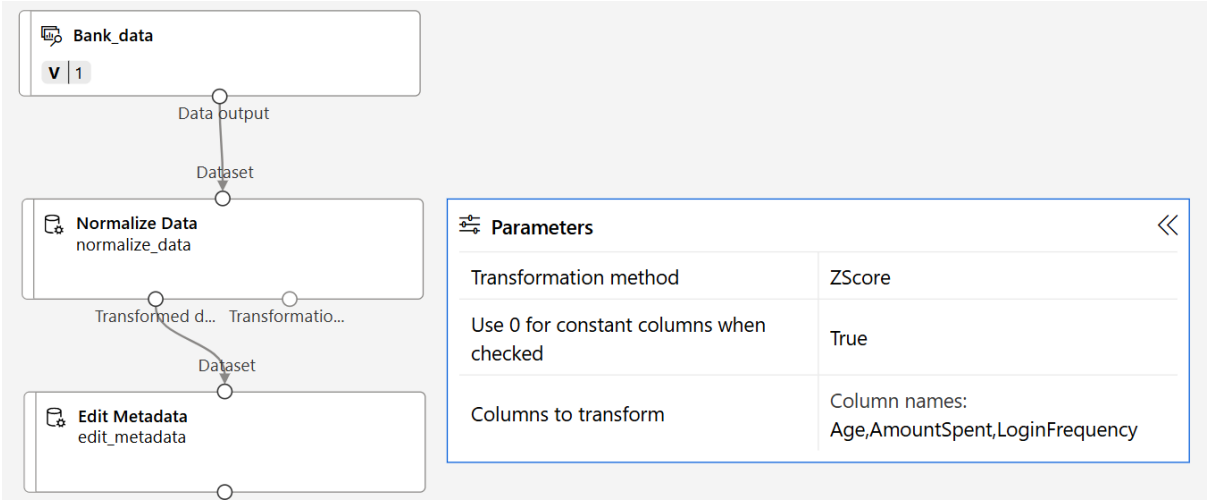

Figure 2.31 Standardizing the Age, AmountSpent and LoginFrequency features

Edit Metadata: Converted the data type of the categorical features to category.

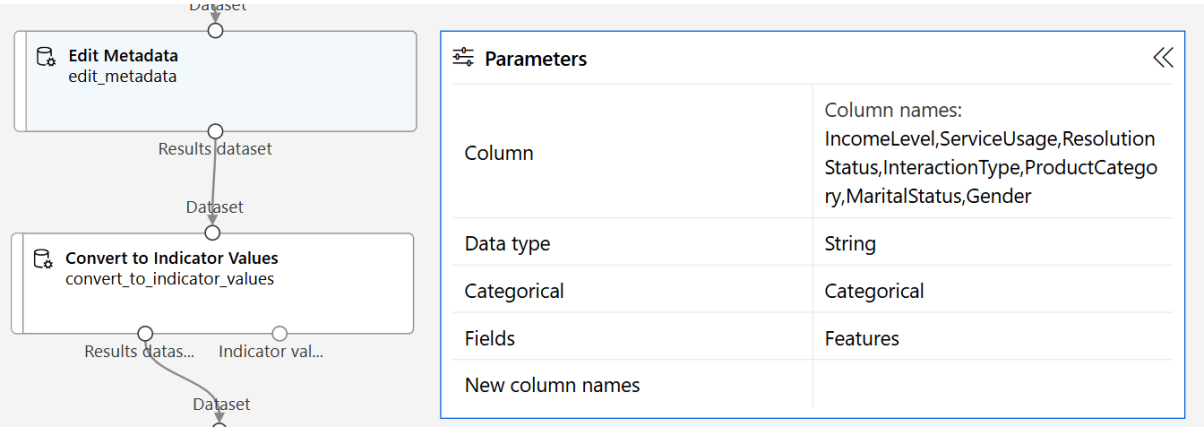

Figure 2.32 Editing metadata of categorical features

Convert to indicators: Categorical features were encoded.

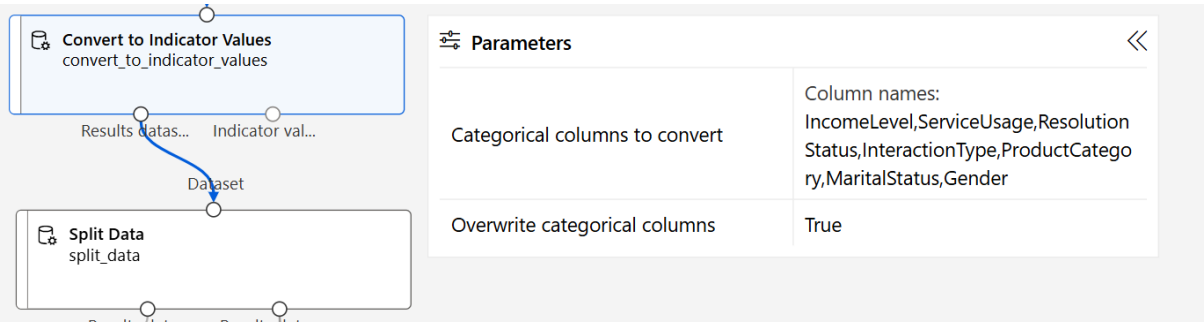

Figure 2.33 Encoding categorical features

Split data: Data was split with a 70:30 ratio, into Train and Test sets.

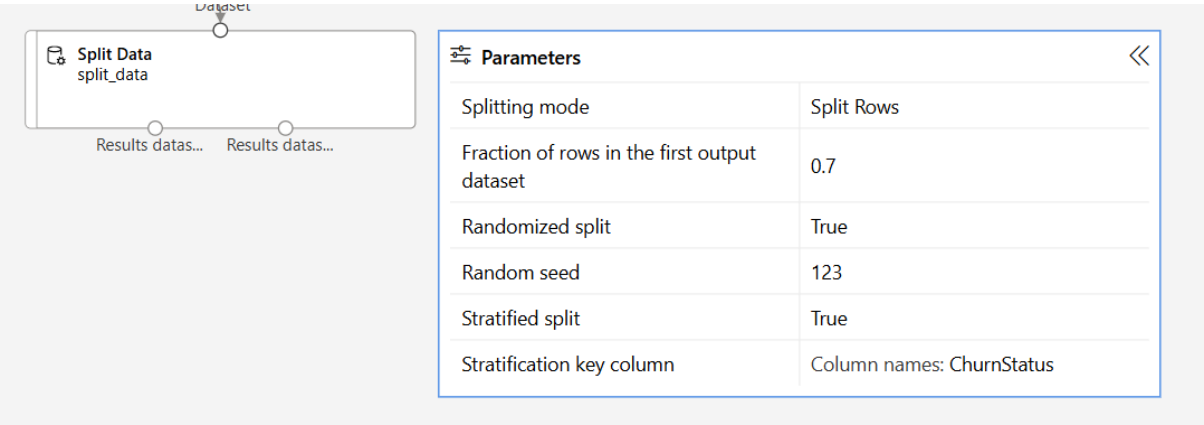

Figure 2.34 70-30 train-test split

Oversampling using SMOTE: class imbalance was addressed by oversampling the minority class of the target label (ChurnStatus).

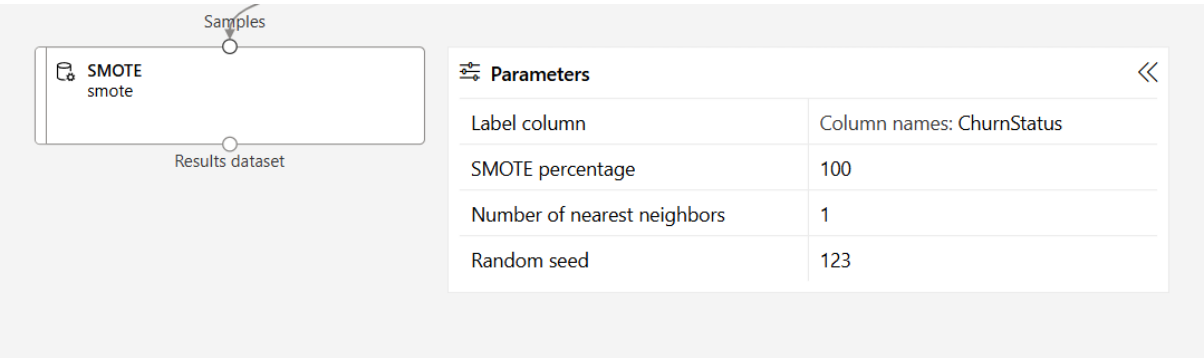

Figure 2.35 Oversampling with SMOTE

1.4.2.2 Model 3: Two-Class Decision Forest (Random Forest)

Model 3 used a two-class decision forest algorithm in Azure ML Studio to predict churn status, the following parameters were configured:

- Number of decision trees: 8
- Maximum depth of the decision trees: 32
- Random state: 123

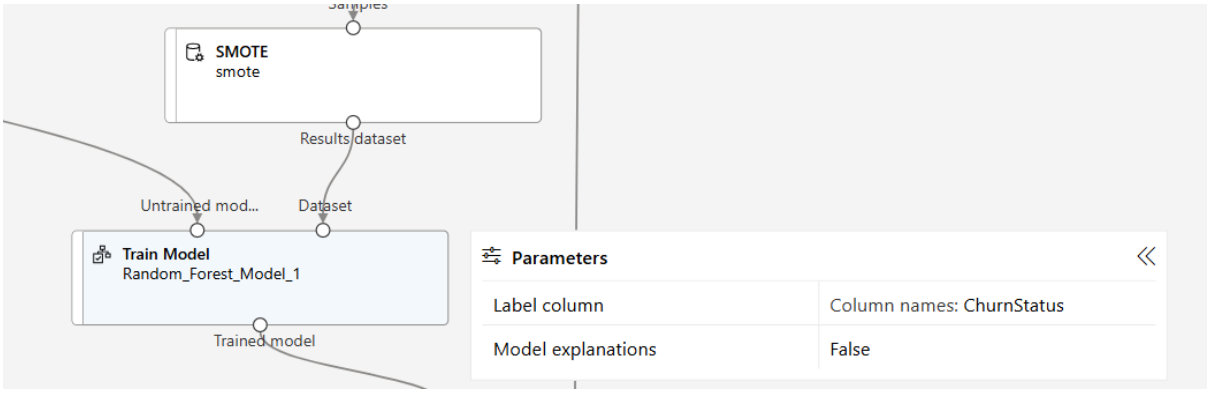

Figure 2.36 Setting ChurnStatus as the target for the two-class decision forest model (Model 3)

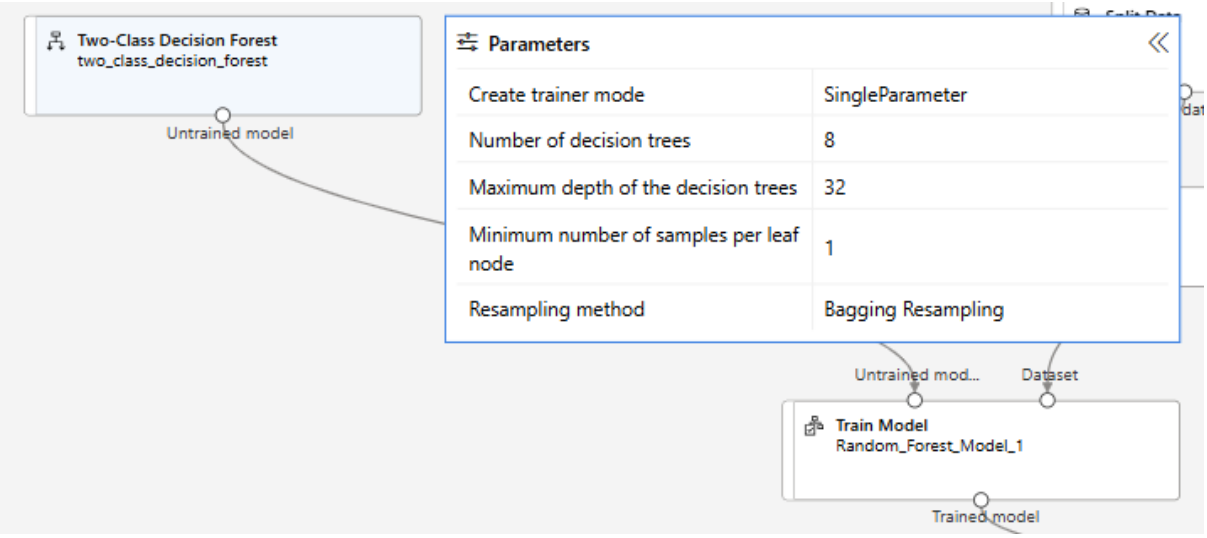

Figure 2.37 setting the parameters to train the two-class decision forest model

Predictions: The model correctly classified 301 out of 343 churn cases and 1532 out of 1616 no-churn cases.

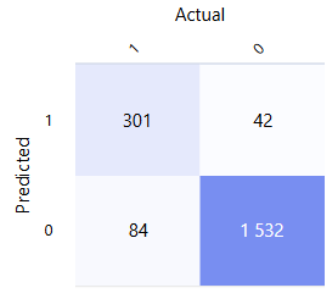

Figure 2.38 Confusion matrix for the two-class decision forest model (Azure ML Studio)

1.4.2.3 Model 4: Two-Class Neural Network (ANN)

The following parameters were configured:

- Number of hidden layers: 100
- Learning rate: 0.1
- Number of learning iterations: 100
- Random state: 123

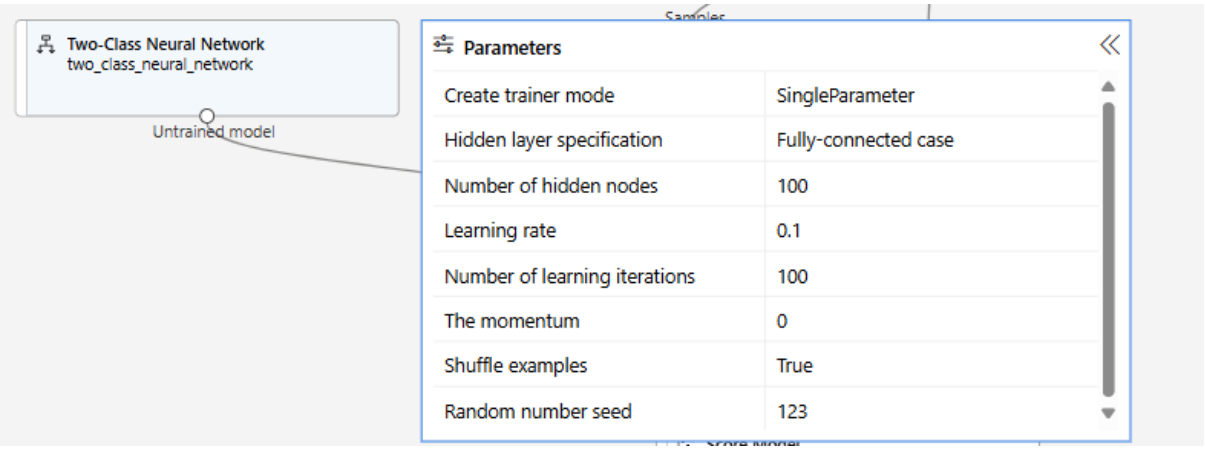

Figure 2.39 setting the parameters to train the two-class neural network model

Predictions: The model correctly classified 167 out of 343 churn cases and 1432 out of 1616 no-churn cases.

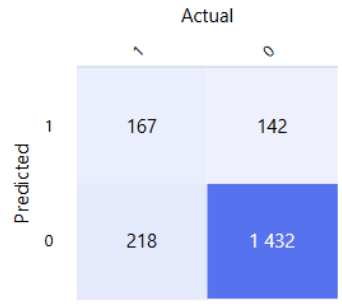

Figure 2.40 Confusion matrix for the two-class neural network model (Azure ML Studio)

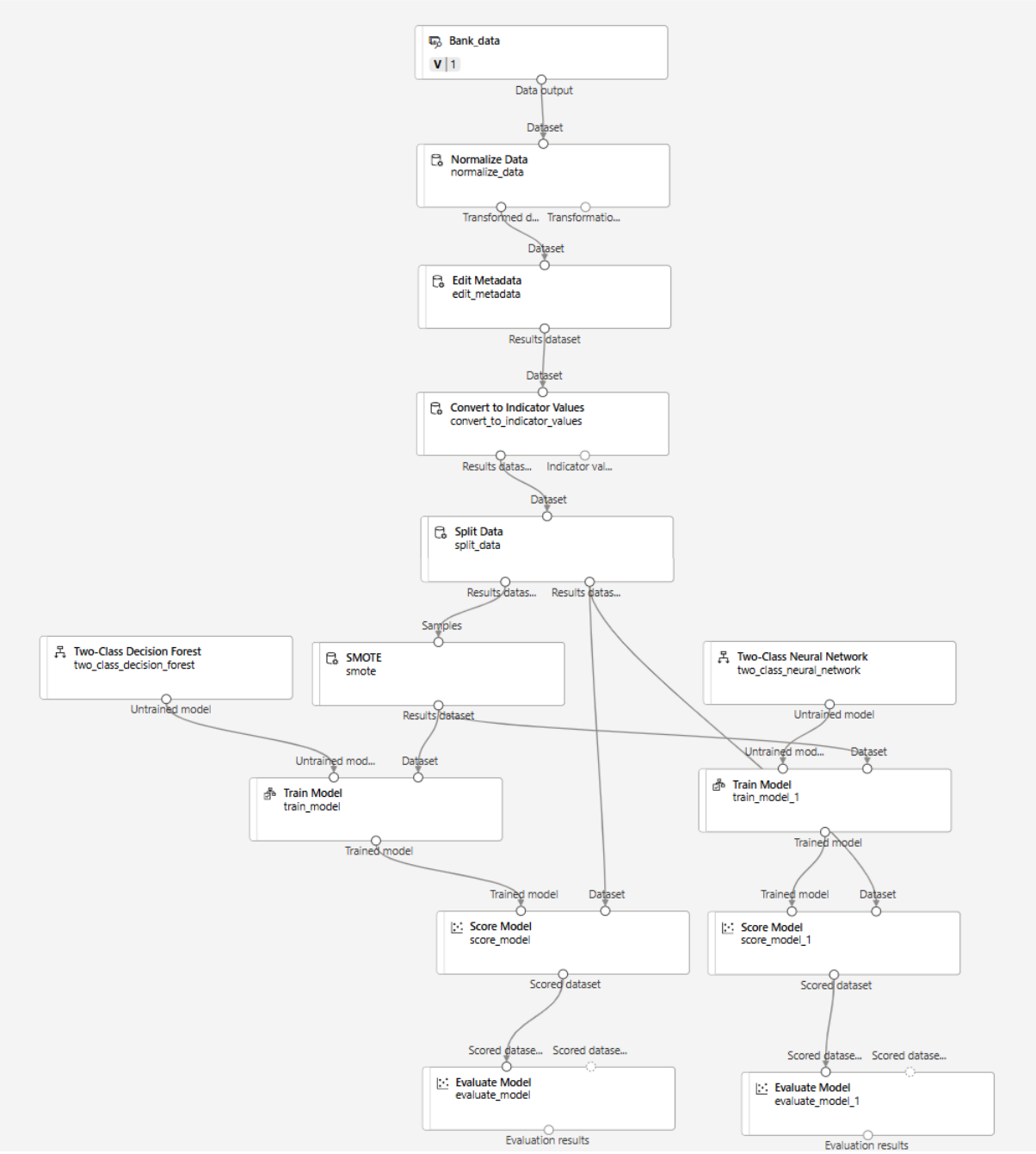

Figure 2.41 Azure ML design pipeline for churn prediction

1.5 Result Analysis and Discussion

1.5.1 Model Evaluation

Each model's performance was evaluated on different metrics, such as precision, recall, f1-score and accuracy. These metrics are calculated from the model predictions confusion matrix (Lawlani et al., 2021).

- Recall: Ratio of real churners
- Precision: Ratio of correctly predicted churners
- Accuracy: ratio of all correct predictions
- F1-score: harmonic average of precision and recall

Model 1 - KNN with Python: The classification report generates the model's evaluation metric scores. Figure 2.42 shows that the KNN model had an f1-score of 75%.

```
# classification report
print(classification_report(y_test,y_pred_knn_optimal))
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      1574
           1       0.75      0.74      0.75       385

    accuracy                           0.90      1959
   macro avg       0.85      0.84      0.84      1959
weighted avg       0.90      0.90      0.90      1959
```

Figure 2.42 Classification report for the KNN model

Model 2- Random Forest with Python: Figure 2.43 shows the random forest model had an f1-score of 84%.

```
print(classification_report(y_test,y_pred_rf_tuned))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      1574
           1       0.96      0.76      0.84       385

    accuracy                           0.95      1959
   macro avg       0.95      0.87      0.91      1959
weighted avg       0.95      0.95      0.94      1959
```

Figure 2.43 Classification report for the random forest model

Model 3 – Two-Class Decision Forest: Model 3, which also uses decision trees like the random forest model had an f1-score of 83%.

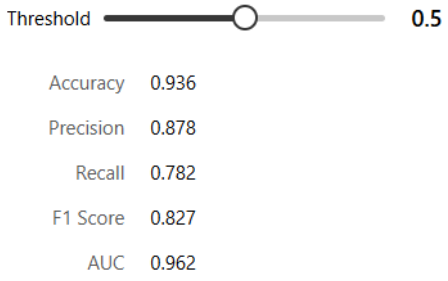

Figure 2.44 Classification report for the two-class decision forest model

Model 4 - Two-Class Neural Network: Model 4 had an f1-score of 48%

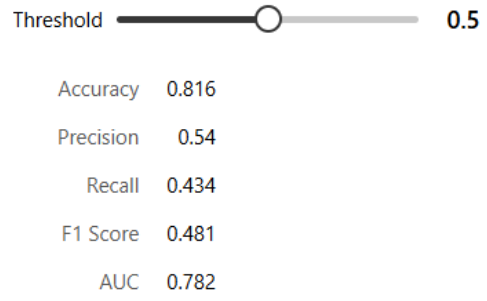

Figure 2.45 Classification report for the two-class neural network model

1.5.2 Model Comparison

Table 2.6 Comparison of all four models

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
KNN	0.90	0.75	0.74	0.75
RANDOM FOREST	**0.95**	**0.96**	0.76	**0.84**
TWO-CLASS DECISION FOREST	0.94	0.878	**0.78**	0.83
TWO-CLASS NEURAL NETWORK	0.82	0.54	0.43	0.481

The random forest model achieved the best performance in three of the evaluation metrics (accuracy, precision and f1-score). KNN showed a balanced performance, while the two-class decision forest had a result comparable with the random forest but fell slightly short. The two-class neural network had the weakest performance in all metrics.

Due to the imbalanced nature of the dataset, the f1-score was used as the primary evaluation metric because it balances both precision and recall.

1.5.3 Feature Importance

This was used to determine the significance of each feature on the predictions made by the model. The feature importance chart shows that login frequency, age and income level are the most important features when trying to predict bank customer churn.

```
# plotting feature importance
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10,6))
sns.barplot(feature_importance_df, x='Importance',y='Feature',color='#66C2A5')
plt.title('Feature Importance for Bank Churn Classification')
plt.show()
```

Figure 2.46 Code to plot the feature importance of the random forest model

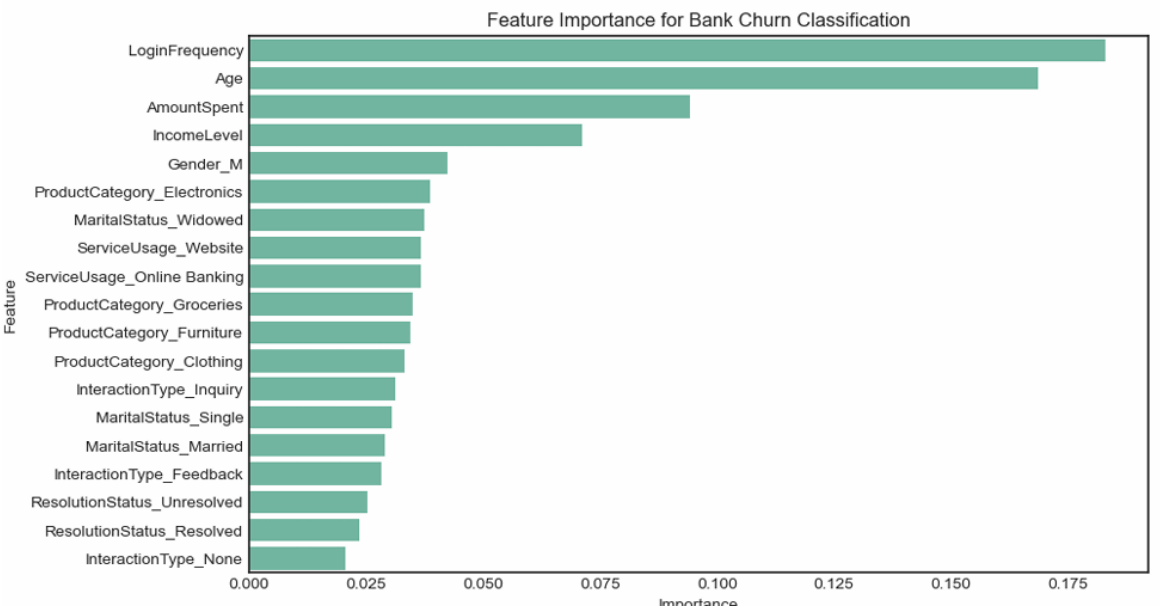

Figure 2.47 Feature importance of the random forest model

1.5.4 Business Benefits

The benefits and impact of these methods include:

- Customer retention strategies: by identifying the key factors that contribute to customer churn like login frequency, age and amount spent, the banks can focus on at-risk customers by implementing strategies specific to these groups.
- Resource allocation: by focusing on just the major factors that influence churn, banks and businesses can better allocate resources to only impactful areas.
- Optimizing processes: this analysis can help optimize operations by identifying high-risk customers and addressing the potential churn effectively.

1.6 Conclusion

This study explored the use of multiple classification algorithms with two implementation methods. The goal was to identify high-risk customers likely to churn and discover the factors that result in churn allowing effective mitigation strategies.

The study findings demonstrated the effectiveness of using a machine learning model to tackle customer churn prediction. The random forest model proved to be the best by achieving a f1-score of 84%. Some key factors such as login frequency, age and amount spent were discovered as strong factors that contribute to churn.

1.6.1 Actionable Recommendations

- Enhance customer engagement of customers with low login frequency by introducing notifications or personalized emails to increase online activity.
- Employ the use of targeted strategies based on customer income level and age group (i.e. customers between the age of 15-30 can be engaged with social media presence).
- Monitor customer base by deploying a model to flag at-risk customers.
- Invest in improving online services such as online banking apps and websites, by making them more interactive and user-friendly for all demographics.

By applying these recommendations, businesses can reduce churn rates and ultimately improve customer satisfaction and revenue.

2 References

Bilal Zoric, A. (2016). Predicting Customer Churn in Banking Industry using Neural Networks. *Interdisciplinary Description of Complex Systems*, *14*(2), 116–124. https://doi.org/10.7906/indecs.14.2.1

Côté, P.-O., Nikanjam, A., Ahmed, N., Humeniuk, D., & Khomh, F. (2024). Data cleaning and machine learning: a systematic literature review. *Automated Software Engineering*, *31*(2), 54-. https://doi.org/10.1007/s10515-024-00453-w

Gohil, S., Ameria, M., & Vergin Raja Sarobin, M. (2023). An intelligent Prediction of Customer Churn in Telecom Industry Using Business Analytics Classification Models. *2023 12th International Conference on Advanced Computing (ICoAC)*, 1–7. https://doi.org/10.1109/ICoAC59537.2023.10249776

Lalwani, P., Mishra, M. K., Chadha, J. S., & Sethi, P. (2022). Customer churn prediction system: a machine learning approach. *Computing*, *104*(2), 271–294. https://doi.org/10.1007/s00607-021-00908-y

Microsoft. (n.d.). SMOTE: Synthetic Minority Oversampling Technique. Microsoft Azure Machine Learning Documentation. Retrieved November 26, 2024, from https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/smote?view=azureml-api-2

Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. C. (2015). A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory*, *55*, 1–9. https://doi.org/10.1016/j.simpat.2015.03.003

Xie, Y., Li, X., Ngai, E. W. T., & Ying, W. (2009). Customer churn prediction using improved balanced random forests. *Expert Systems with Applications*, *36*(3), 5445–5449. https://doi.org/10.1016/j.eswa.2008.06.121