Predicting Bank Customer Churn with Random Forest Modeling

Ashley Traore

Research Question

Research question: How well can a Random Forest model predict customer churn in the banking sector using current demographic and account-level data?

This question was selected due to the critical impact customer churn has on banking profitability and long-term growth. Since retaining existing clients is typically more cost-effective than acquiring new ones, the ability to identify at-risk customers early becomes a strategic imperative. By predicting which individuals are likely to leave, banks can proactively implement retention measures that reduce churn and preserve revenue.

The problem is rooted in a practical business context, where banks routinely collect structured data such as age, tenure, account balance, and credit score. These features are practical and available, making them ideal for building a predictive model. Random Forest was chosen for its robustness across mixed data types, its resistance to overfitting, and its ability to generate feature importance scores, providing both reliable predictions and interpretability for stakeholders.

Hypothesis:

Null Hypothesis (H_0): The Random Forest model does not achieve a predictive accuracy greater than 70% in identifying customer churn.

Alternate Hypothesis (H_1): The Random Forest model achieves a predictive accuracy greater than 70% in identifying customer churn.

This threshold reflects a meaningful level of performance that would make the model useful in a business setting. The goal is not just to build a technically sound model, but one that delivers actionable insights and meets stakeholder expectations for reliability and impact.

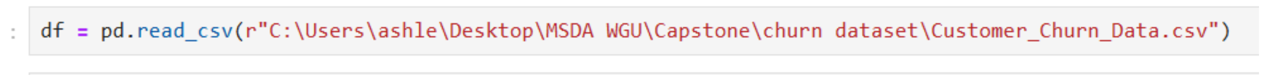
Data Collection

For this project, a publicly available bank customer churn dataset was selected that includes over 10,000 records, each containing structured features such as age, tenure, account balance, product usage, credit score, and whether the customer exited the bank (Meshram, 2023). The dataset was sourced from Kaggle, which provided immediate access to labeled data suitable for supervised machine learning. One key advantage of this data-gathering approach was its efficiency; using a pre-curated dataset allowed the focus to be on modeling and interpretation rather than sourcing raw data. However, some initial cleaning was still required to ensure consistency. This included handling missing values, removing duplicate records, verifying data types, and confirming that categorical features were properly encoded. A notable disadvantage was the presence of class imbalance, with relatively few customers labeled as churned. This imbalance can bias the model toward predicting the majority class. To mitigate this, stratified train-test splitting was applied to preserve class proportions and used ROC-AUC alongside accuracy to evaluate model performance more reliably.

Data Extraction and Preparation

Data Extraction

The dataset used in this project was sourced from Kaggle, a trusted repository for curated datasets in machine learning and analytics. It included over 10,000 customer records from a financial institution, with structured variables such as age, tenure, credit score, account balance, product usage, estimated salary, and a binary churn label (Exited). The data was downloaded in CSV format and imported into a pandas DataFrame using read_csv().



```
df = pd.read_csv(r"C:\Users\ashle\Desktop\MSDA WGU\Capstone\churn dataset\Customer_Churn_Data.csv")
```

All data exploration, cleaning, and modeling were conducted within Jupyter Notebooks, which provided a modular and interactive environment for iterative development. This setup enabled inline visualizations, step-by-step diagnostics, and reproducible documentation of the entire workflow.

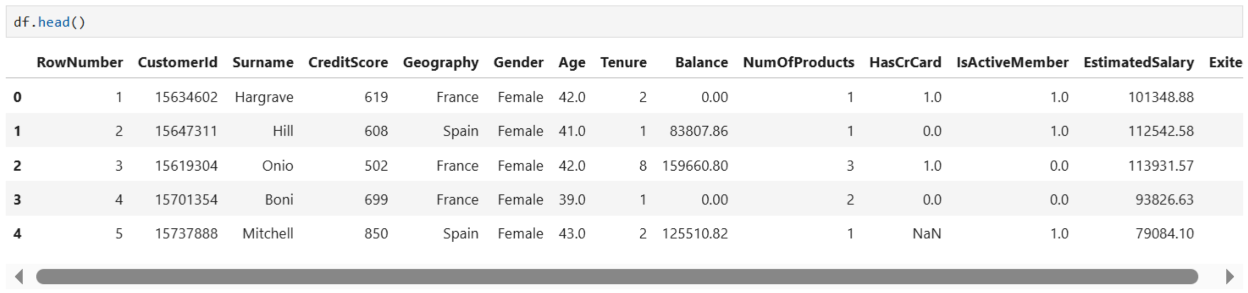
Advantage: Kaggle datasets provide structured, ready-to-use data that accelerates model development, allowing for a focus on analysis rather than data acquisition.

Disadvantage: The presence of class imbalance, with relatively few customers labeled as churned. This imbalance can bias the model toward predicting the majority class

Data Preparation

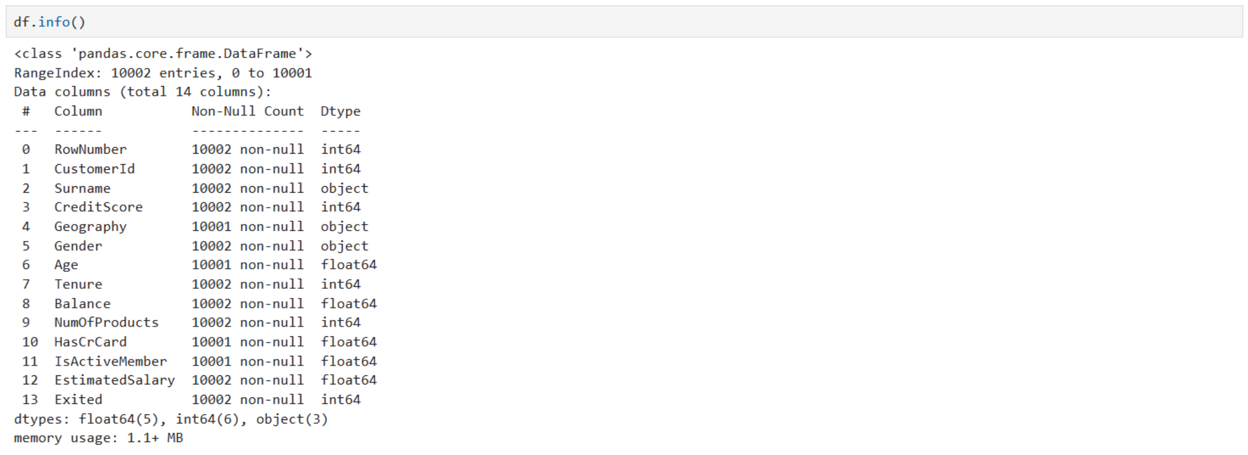
1. Initial Inspection and Visualization

Exploratory commands such as head(), info(), and describe() were used to assess structure, data types, and summary statistics. These commands provided a quick overview of column formats, missing values, and basic distributional properties, helping to flag potential preprocessing needs early in the workflow.



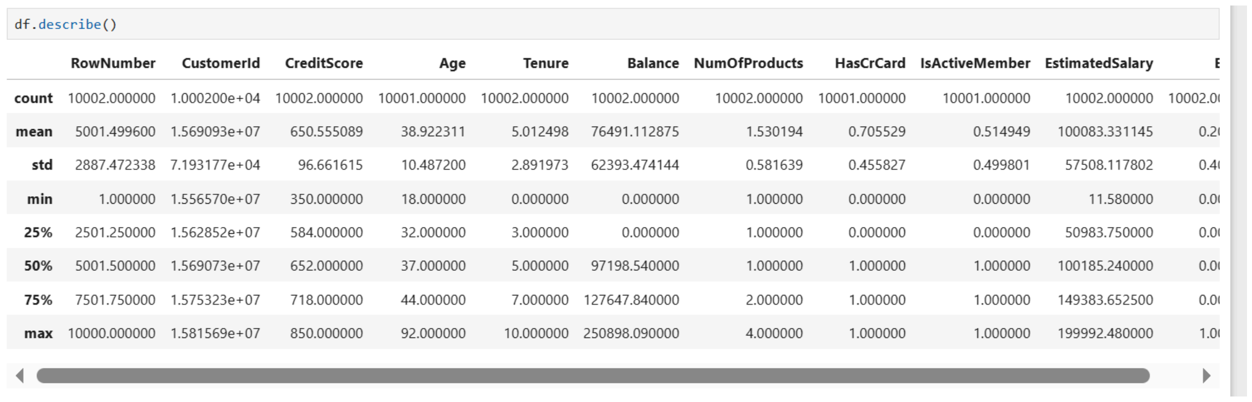
```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exite
0	1	15634602	Hargrave	619	France	Female	42.0	2	0.00	1	1.0	1.0	101348.88	
1	2	15647311	Hill	608	Spain	Female	41.0	1	83807.86	1	0.0	1.0	112542.58	
2	3	15619304	Onio	502	France	Female	42.0	8	159660.80	3	1.0	0.0	113931.57	
3	4	15701354	Boni	699	France	Female	39.0	1	0.00	2	0.0	0.0	93826.63	
4	5	15737888	Mitchell	850	Spain	Female	43.0	2	125510.82	1	NaN	1.0	79084.10	



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10002 entries, 0 to 10001
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10002 non-null  int64  
 1   CustomerId       10002 non-null  int64  
 2   Surname          10002 non-null  object 
 3   CreditScore      10002 non-null  int64  
 4   Geography        10001 non-null  object 
 5   Gender           10002 non-null  object 
 6   Age              10001 non-null  float64
 7   Tenure           10002 non-null  int64  
 8   Balance          10002 non-null  float64
 9   NumOfProducts    10002 non-null  int64  
 10  HasCrCard        10001 non-null  float64
 11  IsActiveMember   10001 non-null  float64
 12  EstimatedSalary  10002 non-null  float64
 13  Exited           10002 non-null  int64  
dtypes: float64(5), int64(6), object(3)
memory usage: 1.1+ MB
```

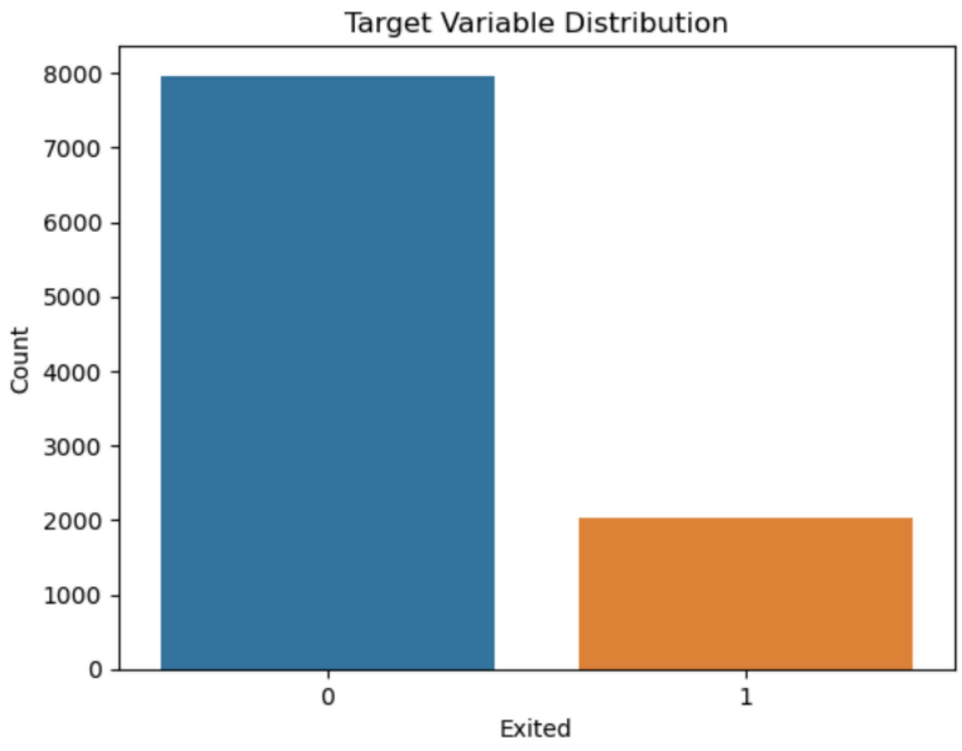


```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	E
count	10002.000000	1.000200e+04	10002.000000	10001.000000	10002.000000	10002.000000	10002.000000	10001.000000	10001.000000	10002.000000	10002.0(
mean	5001.499600	1.569093e+07	650.555089	38.922311	5.012498	76491.112875	1.530194	0.705529	0.514949	100083.331145	0.2(
std	2887.472338	7.193177e+04	96.661615	10.487200	2.891973	62393.474144	0.581639	0.455827	0.499801	57508.117802	0.4(
min	1.000000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.000000	0.000000	11.580000	0.0(
25%	2501.250000	1.562852e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.000000	0.000000	50983.750000	0.0(
50%	5001.500000	1.569073e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.000000	1.000000	100185.240000	0.0(
75%	7501.750000	1.575323e+07	718.000000	44.000000	7.000000	127647.840000	2.000000	1.000000	1.000000	149383.652500	0.0(
max	10000.000000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.000000	1.000000	199992.480000	1.0(

The distribution of the target variable (Exited) was visualized using seaborn's countplot, revealing a churn rate of approximately 20%. Visualizing the target distribution is essential for understanding class imbalance, which directly impacts model training and evaluation. A 20% churn rate suggests a moderately imbalanced classification problem, warranting consideration of stratified sampling or resampling techniques to ensure fair model performance across classes.

```python
# Print a bar chart to visualize the target variable distribution
sns.countplot(x='Exited', data=df)
plt.title('Target Variable Distribution')
plt.ylabel('Count')
plt.show()

#Print value counts
print(df['Exited'].value_counts(normalize=True))
```



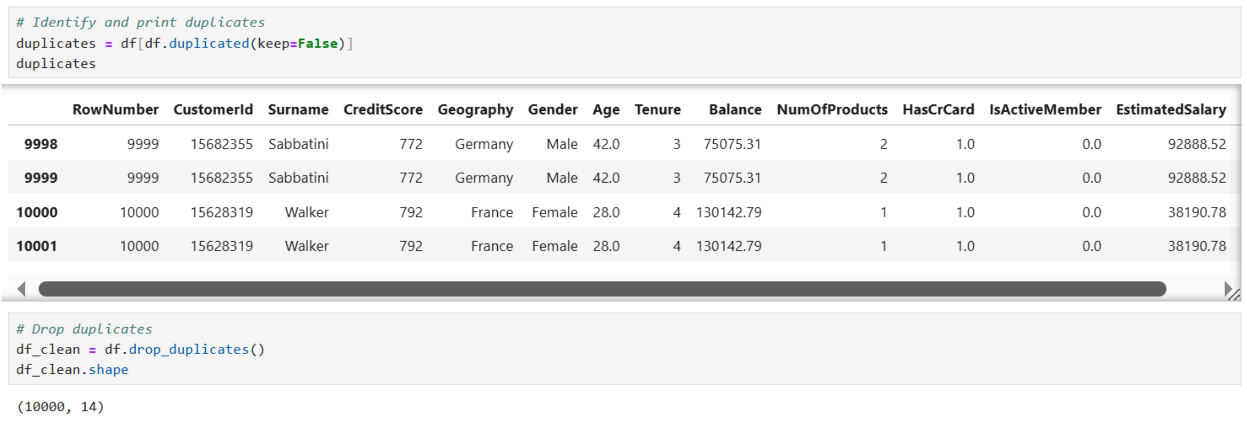
```
Exited
0    0.796241
1    0.203759
Name: proportion, dtype: float64
```

Advantage: Early inspection helps identify data quality issues and informs downstream preprocessing decisions.

Disadvantage: Summary statistics may mask underlying patterns or anomalies that require deeper analysis.

2. Duplicate and Missing Value Handling

Duplicate records were identified using duplicated() and removed with drop_duplicates() to ensure that each customer was uniquely represented in the dataset. This step was essential for maintaining the integrity of statistical summaries and preventing overrepresentation of repeated observations. Duplicate entries can distort model learning by artificially inflating the frequency of certain patterns, leading to biased predictions and misleading feature importance.



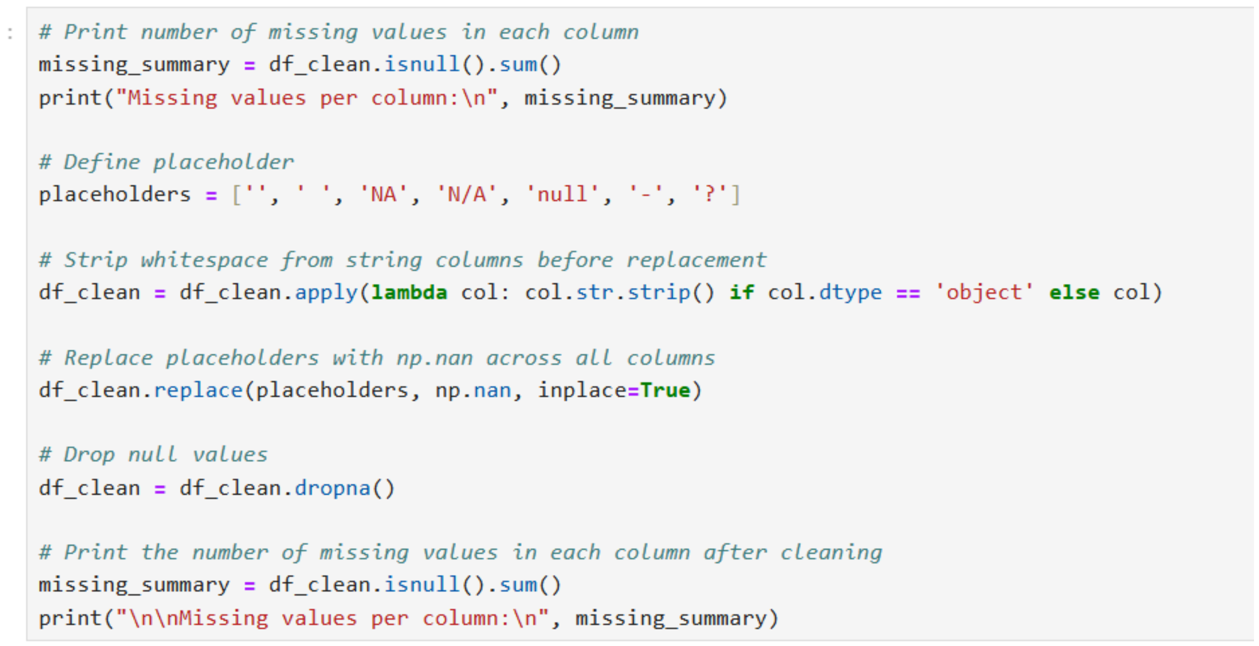
```python
# Identify and print duplicates
duplicates = df[df.duplicated(keep=False)]
duplicates
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
9998	9999	15682355	Sabbatini	772	Germany	Male	42.0	3	75075.31	2	1.0	0.0	92888.52
9999	9999	15682355	Sabbatini	772	Germany	Male	42.0	3	75075.31	2	1.0	0.0	92888.52
10000	10000	15628319	Walker	792	France	Female	28.0	4	130142.79	1	1.0	0.0	38190.78
10001	10000	15628319	Walker	792	France	Female	28.0	4	130142.79	1	1.0	0.0	38190.78

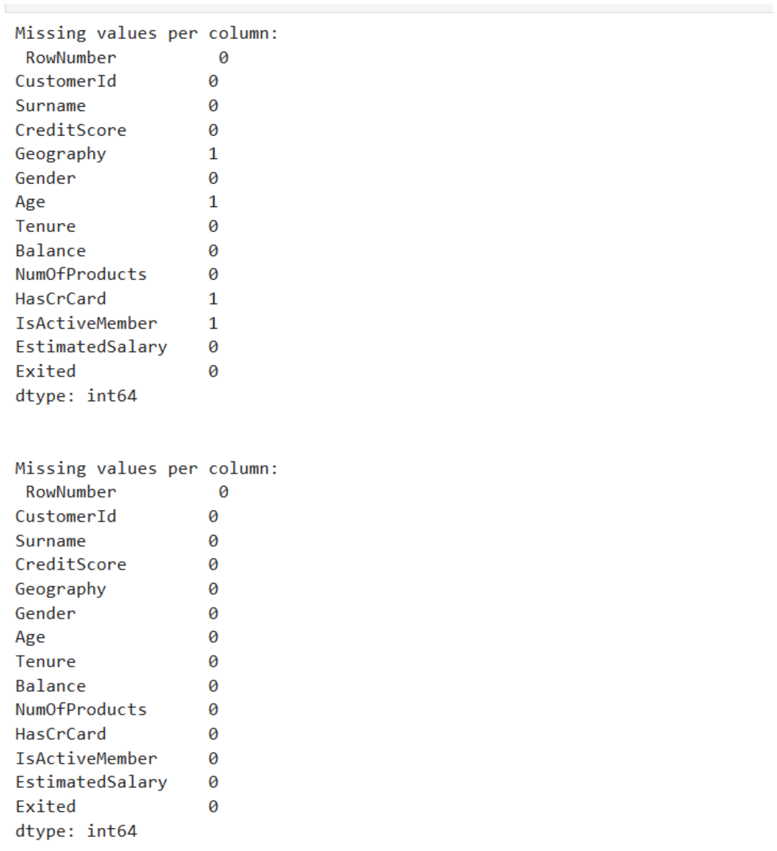
```python
# Drop duplicates
df_clean = df.drop_duplicates()
df_clean.shape
```

```
(10000, 14)
```

Missing values were addressed by first identifying placeholder entries (for example, empty strings, whitespace, or non-standard null indicators) and replacing them with np.nan to standardize detection. Rows containing null values were then dropped using dropna() to ensure that only complete records were used for modeling. This approach was chosen because the proportion of missing data was minimal and did not justify imputation, which could introduce noise or assumptions not supported by the data.



```python
# Print number of missing values in each column
missing_summary = df_clean.isnull().sum()
print("Missing values per column:\n", missing_summary)

# Define placeholder
placeholders = ['', ' ', 'NA', 'N/A', 'null', '-', '?']

# Strip whitespace from string columns before replacement
df_clean = df_clean.apply(lambda col: col.str.strip() if col.dtype == 'object' else col)

# Replace placeholders with np.nan across all columns
df_clean.replace(placeholders, np.nan, inplace=True)

# Drop null values
df_clean = df_clean.dropna()

# Print the number of missing values in each column after cleaning
missing_summary = df_clean.isnull().sum()
print("\n\nMissing values per column:\n", missing_summary)
```


```
Missing values per column:
 RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          1
Gender             0
Age                1
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          1
IsActiveMember     1
EstimatedSalary    0
Exited             0
dtype: int64


Missing values per column:
 RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Advantage: Removing duplicates and nulls improves data integrity and prevents misleading model behavior.

Disadvantage: Dropping rows with missing values can reduce sample size and potentially discard informative data.

3. Outlier Detection and Capping

Outliers were identified across all numerical features using the interquartile range (IQR) method, which calculates the spread between the 25th and 75th percentiles to flag values that fall significantly outside the typical range. This approach is robust to skewed distributions and does not assume normality, making it well-suited for exploratory data analysis in real-world datasets.

For variables where outliers were present, such as CreditScore, Age, and NumOfProducts, Z-score capping was implemented to limit extreme values to within three standard deviations from the mean. This technique preserves the overall distribution while reducing the influence of outliers that could distort model training or inflate error metrics. By capping rather than removing these values, the dataset retains potentially informative observations without allowing them to dominate the learning process.

```python
#Identifying Outliers
#Define column variable with all the numeric columns in the data frame
columns = df_clean.select_dtypes(exclude=['object']).drop('Exited', axis=1).columns

#Loop through each numeric column to find outliers
for i in columns:
    #print(")
    # Calculating the first quartile, the third quartile, and the interquartile range. [In-Text Citation: (Hackers Realm)]
    Q1 = df_clean[i].quantile(0.25)
    Q3 = df_clean[i].quantile(0.75)
    IQR = Q3 - Q1

    # Define outlier bounds
    upper_bound = Q3 + 1.5 * IQR
    lower_bound = Q1 - 1.5 * IQR

    # Identify outliers
    outliers = df_clean[(df_clean[i] < lower_bound) | (df_clean[i] > upper_bound)]
    #Printing the count and range of outliers
    print("\n\033[1m " + i + ": \033[0m" + "Count of outliers: " + str(len(outliers)))


# Cleaning Outliers
# Z-score capping to cap all columns with outliers with a threshold set at 3 standard deviations from the mean.
outlierColumns = ['CreditScore', 'Age', 'NumOfProducts']
# Loop through each column that has outliers.
for i in outlierColumns:
    # Calculate the upper and lower threshold using the Interquartile Range (IQR) method.
    upperThreshold = df_clean[i].mean() + 3 * df_clean[i].std()
    lowerThreshold = df_clean[i].mean() - 3 * df_clean[i].std()
    #Print upper and lower threshold for each column
    print("\n\033[1m" + i + ": \033[0m Upper Threshold: " + str(upperThreshold) + ' Lower Threshold: ' + str(lowerThreshold))

    #Capping outliers for each column on upper and lower limit
    df_clean.loc[(df_clean[i]>=upperThreshold), i] = float(upperThreshold)
    df_clean.loc[(df_clean[i]<=lowerThreshold), i] = float(lowerThreshold)

     # Count outliers after cleaning
    outliers_ = df_clean[(df_clean[i] < lowerThreshold) | (df_clean[i] > upperThreshold)]
    print("Count of Outliers after capping: " + str(len(outliers_)))
```

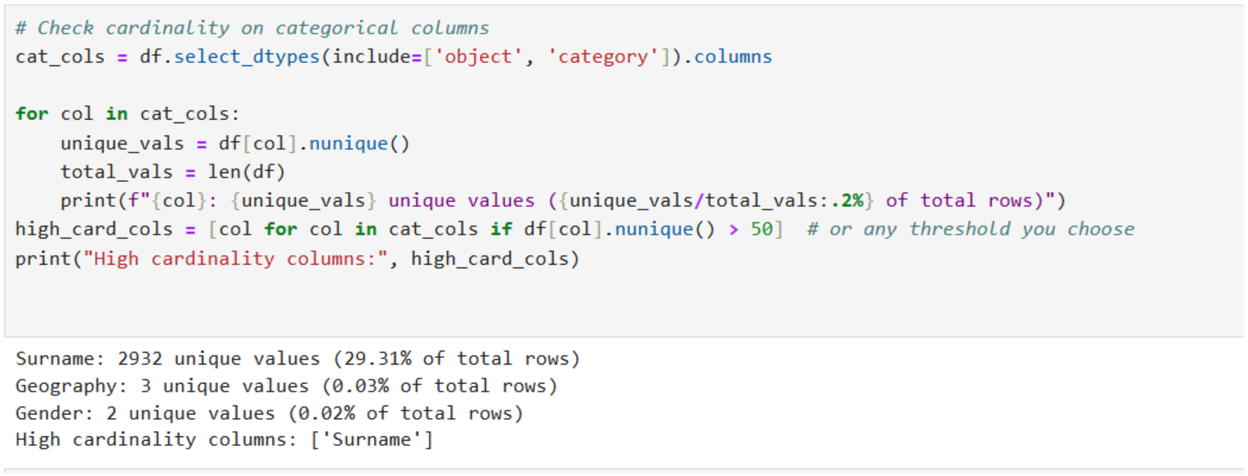
```
 RowNumber: Count of outliers: 0

 CustomerId: Count of outliers: 0

 CreditScore: Count of outliers: 16

 Age: Count of outliers: 359

 Tenure: Count of outliers: 0

 Balance: Count of outliers: 0

 NumOfProducts: Count of outliers: 60

 HasCrCard: Count of outliers: 0

 IsActiveMember: Count of outliers: 0

 EstimatedSalary: Count of outliers: 0

CreditScore:  Upper Threshold: 940.3773051066316 Lower Threshold: 360.62929753442467
Count of Outliers after capping: 0

Age:  Upper Threshold: 70.38633419345116 Lower Threshold: 7.455808663691698
Count of Outliers after capping: 0

NumOfProducts:  Upper Threshold: 3.275265367923586 Lower Threshold: -0.214841198255719
Count of Outliers after capping: 0
```

Advantage: Capping extreme values reduces noise and prevents outliers from skewing model training.

Disadvantage: Artificially adjusting values may obscure genuine patterns or behaviors in the data.

4. Cardinality Check and Categorical Encoding

Before encoding, cardinality was assessed for all categorical features using nunique() to ensure that encoding strategies were appropriate and scalable. This step helps distinguish between low-cardinality variables, where simple encoding is sufficient, and high-cardinality variables, which may require dimensionality reduction or specialized techniques to avoid overfitting and sparse representations.

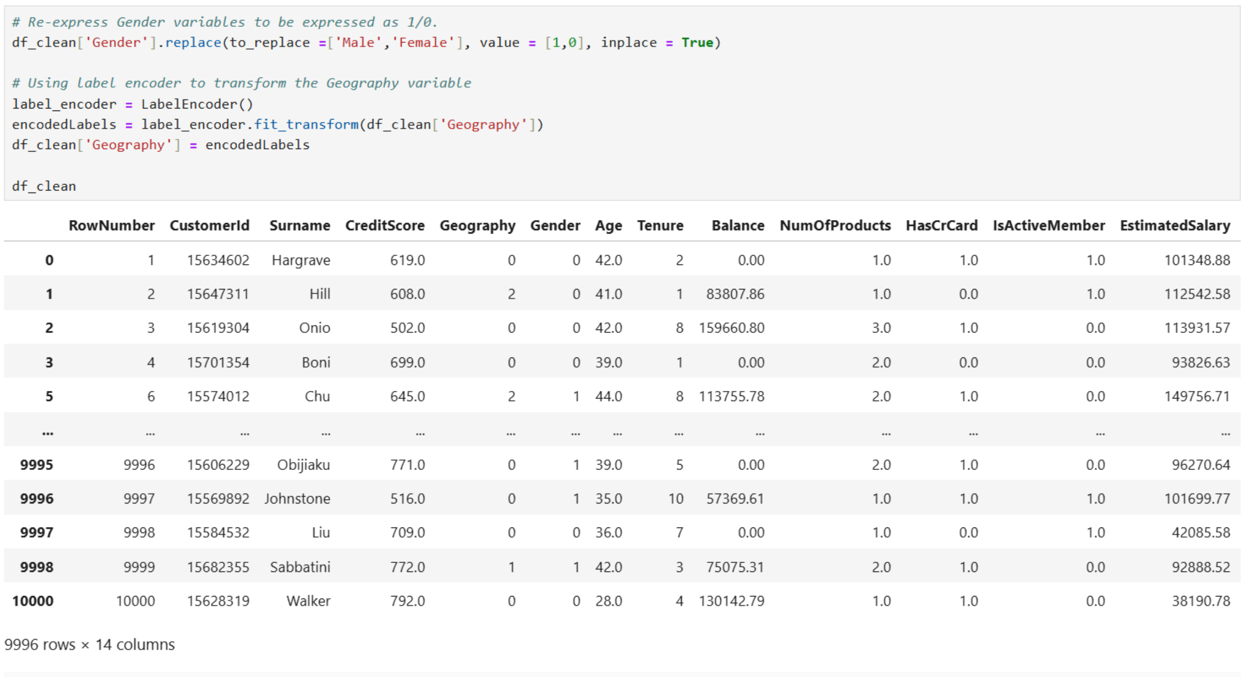


```python
# Check cardinality on categorical columns
cat_cols = df.select_dtypes(include=['object', 'category']).columns

for col in cat_cols:
    unique_vals = df[col].nunique()
    total_vals = len(df)
    print(f"{col}: {unique_vals} unique values ({unique_vals/total_vals:.2%} of total rows)")
high_card_cols = [col for col in cat_cols if df[col].nunique() > 50]  # or any threshold you choose
print("High cardinality columns:", high_card_cols)
```

```
Surname: 2932 unique values (29.31% of total rows)
Geography: 3 unique values (0.03% of total rows)
Gender: 2 unique values (0.02% of total rows)
High cardinality columns: ['Surname']
```

Binary categorical variables, such as Gender, were manually encoded to 0/1. This direct mapping maintains interpretability and avoids unnecessary complexity in the feature space. For example, encoding Male as 1 and Female as 0 allows models to treat the variable as numeric without introducing multicollinearity or inflated dimensionality.

Multi-class variables like Geography were transformed using LabelEncoder, which assigns a unique integer to each category. This approach is efficient for tree-based models (for example, Random Forest), which are insensitive to the ordinal nature of label encoding.



```python
# Re-express Gender variables to be expressed as 1/0.
df_clean['Gender'].replace(to_replace =['Male','Female'], value = [1,0], inplace = True)

# Using label encoder to transform the Geography variable
label_encoder = LabelEncoder()
encodedLabels = label_encoder.fit_transform(df_clean['Geography'])
df_clean['Geography'] = encodedLabels

df_clean
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	1	15634602	Hargrave	619.0	0	0	42.0	2	0.00	1.0	1.0	1.0	101348.88
1	2	15647311	Hill	608.0	2	0	41.0	1	83807.86	1.0	0.0	1.0	112542.58
2	3	15619304	Onio	502.0	0	0	42.0	8	159660.80	3.0	1.0	0.0	113931.57
3	4	15701354	Boni	699.0	0	0	39.0	1	0.00	2.0	0.0	0.0	93826.63
5	6	15574012	Chu	645.0	2	1	44.0	8	113755.78	2.0	1.0	0.0	149756.71
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771.0	0	1	39.0	5	0.00	2.0	1.0	0.0	96270.64
9996	9997	15569892	Johnstone	516.0	0	1	35.0	10	57369.61	1.0	1.0	1.0	101699.77
9997	9998	15584532	Liu	709.0	0	0	36.0	7	0.00	1.0	0.0	1.0	42085.58
9998	9999	15682355	Sabbatini	772.0	1	1	42.0	3	75075.31	2.0	1.0	0.0	92888.52
10000	10000	15628319	Walker	792.0	0	0	28.0	4	130142.79	1.0	1.0	0.0	38190.78

9996 rows × 14 columns

Advantage: Cardinality checks help prevent overfitting and guide the selection of efficient encoding methods (Connor, 2025).

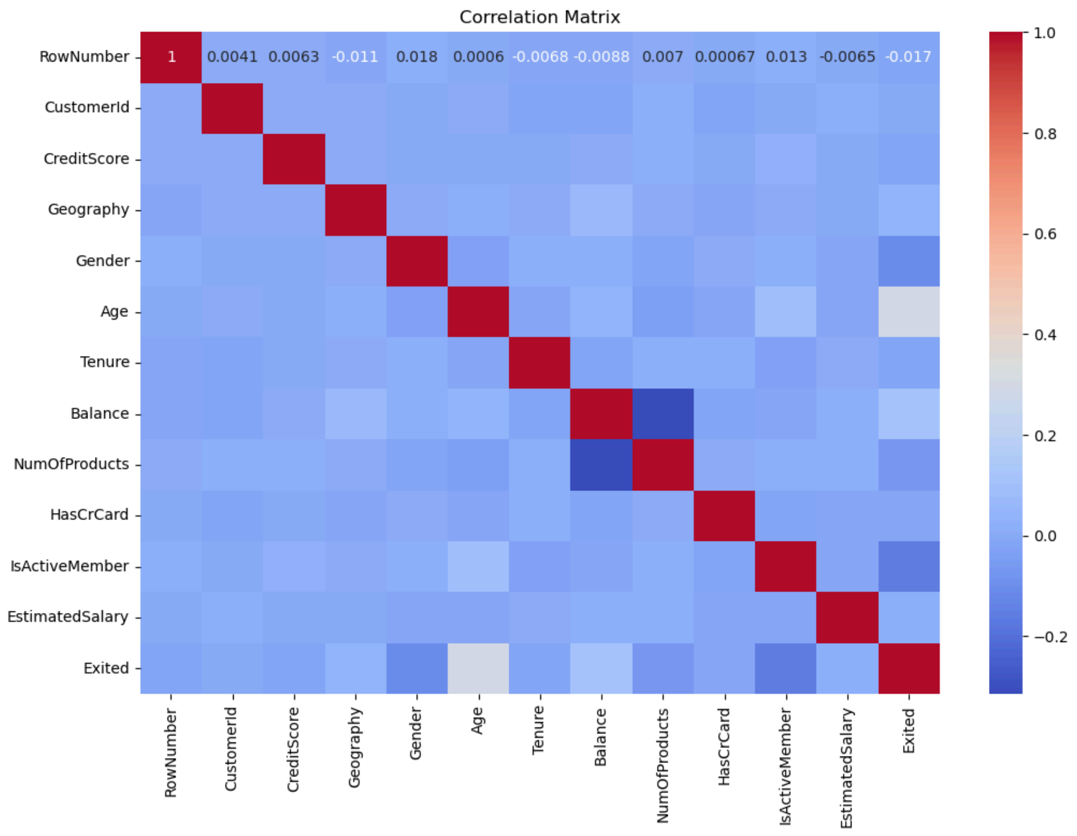
Disadvantage: Label encoding does not preserve category interpretability, which can make feature importance plots or stakeholder explanations less intuitive, especially when categories are represented as arbitrary integers.

5. Feature Selection and Multicollinearity Check

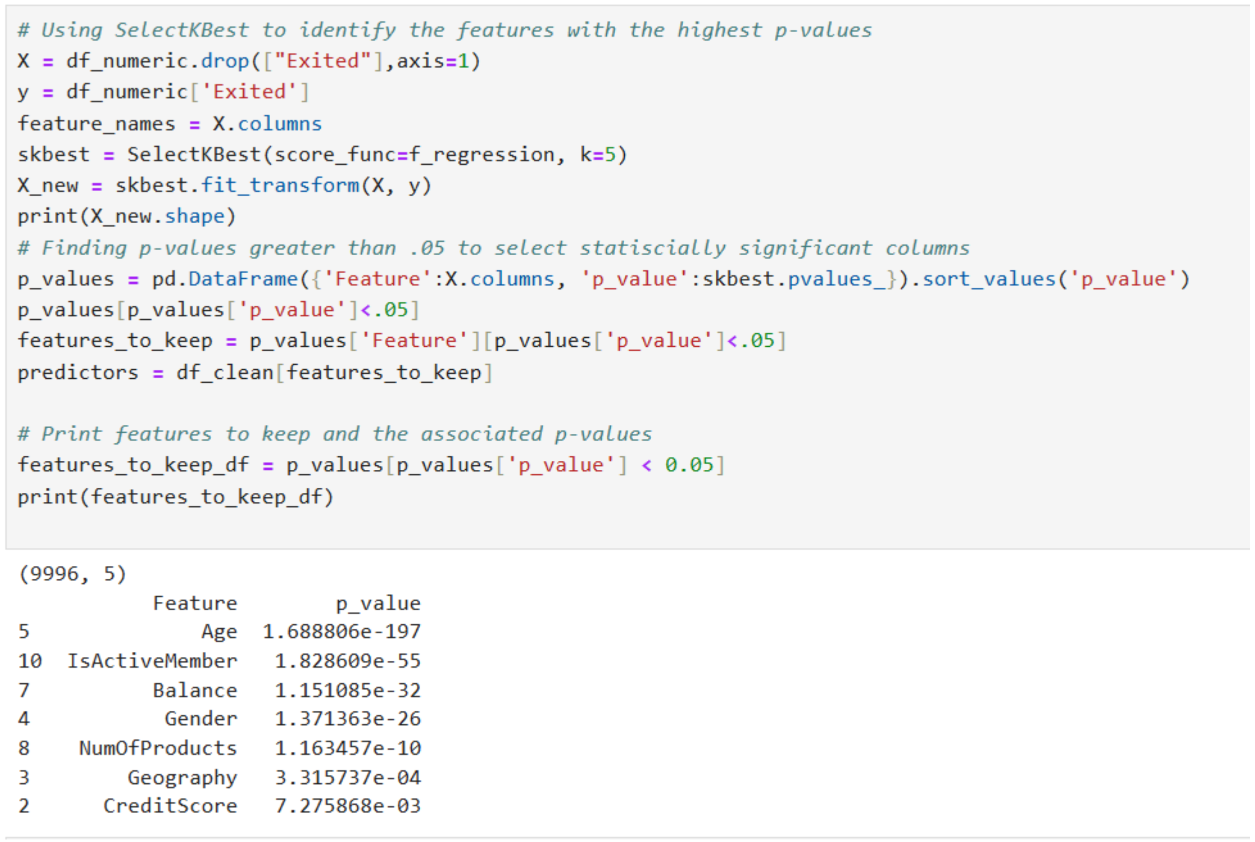
To optimize model performance and enhance interpretability, a multi-step feature selection process was conducted. This approach ensured that only the most relevant, non-redundant predictors were retained, improving both computational efficiency and stakeholder transparency.

Correlation Analysis: A correlation matrix was generated to identify highly correlated pairs among numeric features. This helped flag potential redundancy and guide the initial pruning of variables with high interdependence. Strongly correlated predictors can distort model interpretation by inflating importance scores and masking the true signal, especially in ensemble methods like Random Forest (Ampawanon, 2023). By removing or consolidating these features, the model becomes more streamlined, interpretable, and less prone to overfitting, ultimately enhancing both predictive performance and stakeholder clarity.

```python
# Select numeric columns
df_numeric = df_clean.select_dtypes(include=[np.number])

# Correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



Statistical Significance via SelectKBest: The SelectKBest method with f_regression scoring was applied to evaluate the linear relationship between each independent variable and the target. Features with p-values below 0.05 were considered statistically significant and retained for further modeling. This threshold ensured that only predictors with meaningful explanatory power were included, while also reinforcing reproducibility and stakeholder confidence by anchoring feature selection in transparent, data-driven criteria.



```
# Using SelectKBest to identify the features with the highest p-values
X = df_numeric.drop(["Exited"],axis=1)
y = df_numeric['Exited']
feature_names = X.columns
skbest = SelectKBest(score_func=f_regression, k=5)
X_new = skbest.fit_transform(X, y)
print(X_new.shape)
# Finding p-values greater than .05 to select statiscially significant columns
p_values = pd.DataFrame({'Feature':X.columns, 'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value']<.05]
features_to_keep = p_values['Feature'][p_values['p_value']<.05]
predictors = df_clean[features_to_keep]

# Print features to keep and the associated p-values
features_to_keep_df = p_values[p_values['p_value'] < 0.05]
print(features_to_keep_df)
```

```
(9996, 5)
           Feature        p_value
5              Age  1.688806e-197
10  IsActiveMember   1.828609e-55
7          Balance   1.151085e-32
4           Gender   1.371363e-26
8    NumOfProducts   1.163457e-10
3        Geography   3.315737e-04
2      CreditScore   7.275868e-03
```

Multicollinearity Check using VIF: To evaluate multicollinearity among numeric predictors, Variance Inflation Factor (VIF) scores were computed using the statsmodels library. The analysis was performed on all numeric features excluding the target variable Exited, with a constant term added to account for the intercept in regression modeling. VIF measures how much a feature overlaps with others, helping to spot when predictors are too similar, which can make the model unstable or harder to interpret. The resulting VIF values indicated that none of the features exceeded the conventional threshold of 10, suggesting that multicollinearity was not a concern in this dataset. This result is important because it helps keep the model's estimates accurate and avoids errors that could hide real patterns in the data, making the insights clearer and more trustworthy for stakeholders. As a result, no features required removal or transformation, and the model remains well-equipped to produce reliable, stakeholder-ready insights without distortion from collinearity.

```
# VIF to check multicollinearity
print("\n\033[1m VIF Check: \033[0m")
X = sm.add_constant(df_numeric.drop('Exited', axis=1))
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_data)
```

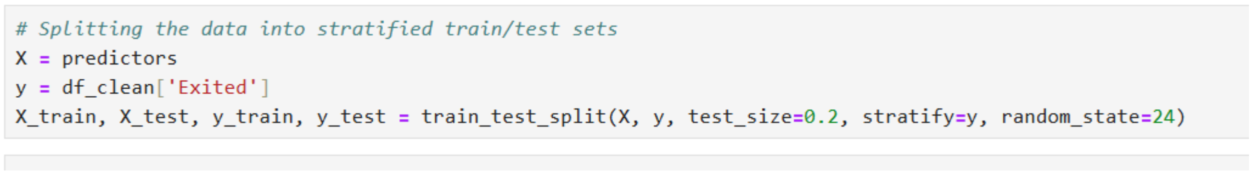
```
 VIF Check:
            feature           VIF
0             const  47695.258688
1         RowNumber      1.000840
2        CustomerId      1.001218
3       CreditScore      1.001129
4         Geography      1.006527
5            Gender      1.002618
6               Age      1.010510
7            Tenure      1.002178
8           Balance      1.117690
9     NumOfProducts      1.113456
10        HasCrCard      1.001349
11   IsActiveMember      1.009606
12  EstimatedSalary      1.001177
```

Advantage: This layered approach ensures that selected features are both statistically relevant and non-redundant, which improves model generalizability, reduces overfitting risk, and enhances interpretability for stakeholders.

Disadvantage: Sole reliance on statistical thresholds (for example, p-values or VIF cutoffs) may inadvertently exclude features that hold contextual or domain-specific importance, particularly those valued by stakeholders for business relevance, even if their statistical contribution is marginal.

6. Train-Test Split and Stratification

To preserve the class distribution of churned vs. retained customers, a stratified train-test split was performed. This approach ensures that both training and testing datasets maintain the same proportion of the target class Exited, which is especially important when dealing with imbalanced outcomes. By doing so, the model is exposed to representative patterns during training and evaluated on a realistic sample during testing, reducing bias, improving generalizability, and supporting more reliable performance metrics such as ROC-AUC.



```
# Splitting the data into stratified train/test sets
X = predictors
y = df_clean['Exited']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=24)
```

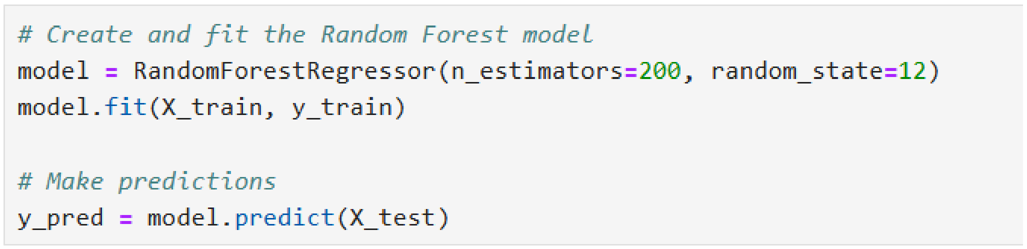
Advantage: Stratification ensures balanced representation of target classes in both training and test sets, improving evaluation reliability.

Disadvantage: A stratified train-test split may reduce flexibility in cross-validation strategies or introduce bias if the original class distribution is heavily skewed, potentially limiting the model’s ability to generalize across underrepresented patterns.

Analysis

Random Forest Modeling

A Random Forest classification model was applied to the cleaned dataset to predict whether a customer exited the bank. This technique was selected for its ability to capture nonlinear relationships, accommodate mixed data types, and remain robust in the presence of outliers and noise. The model was configured with 200 estimators and a fixed random seed to ensure reproducibility across runs and consistency in performance evaluation.



```
# Create and fit the Random Forest model
model = RandomForestRegressor(n_estimators=200, random_state=12)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Advantage: High predictive accuracy and resilience to noisy or complex data.

Disadvantage: Limited transparency, individual decision paths within trees are difficult to interpret, which can challenge explainability in regulated settings (Kundu, 2022).

After training on a stratified sample that preserved the original class distribution, the model generated predicted probabilities on the test set. These probabilities were then converted into binary classifications using two thresholds: a standard threshold of 0.5 for general performance assessment, and a stricter threshold of 0.75 to identify high-risk customers for targeted retention strategies. This dual-threshold approach enabled both broad churn detection and focused business intervention, aligning technical outputs with operational priorities.

Advantage: Dual thresholding allows flexible decision-making, balancing broad detection with focused intervention.

Disadvantage: A stricter threshold (for example, 0.75) may reduce recall, potentially missing customers who are likely to churn but fall below the cutoff.

Model Evaluation

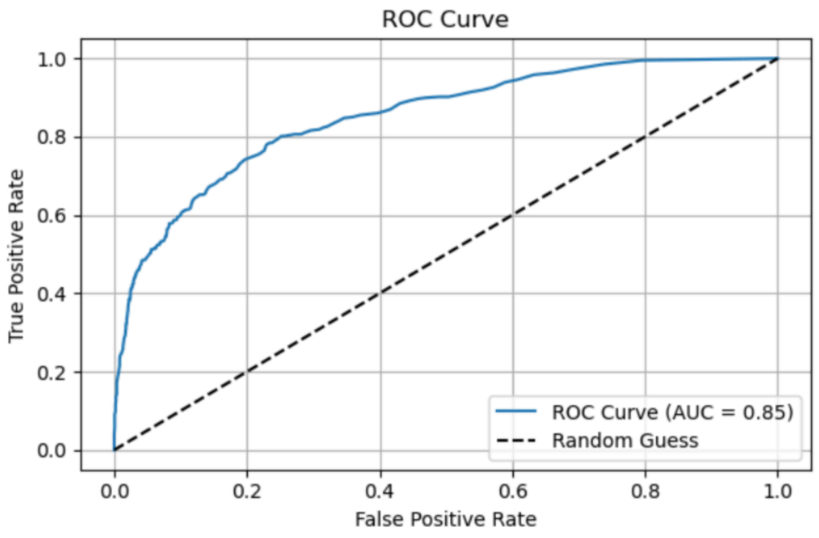
Model performance was evaluated using multiple metrics. The model achieved an accuracy of 85.2%, indicating that the majority of predictions matched the actual outcomes in the test set. Additionally, the model produced a ROC-AUC score of 0.8511, reflecting strong discriminatory power in distinguishing between exited and retained customers across varying probability thresholds. To support interpretation, a ROC-AUC curve was also plotted to visually represent the model's discriminatory power.

Advantage: ROC-AUC is especially valuable for imbalanced datasets, as it evaluates the model's ability to rank predictions correctly regardless of class proportions.

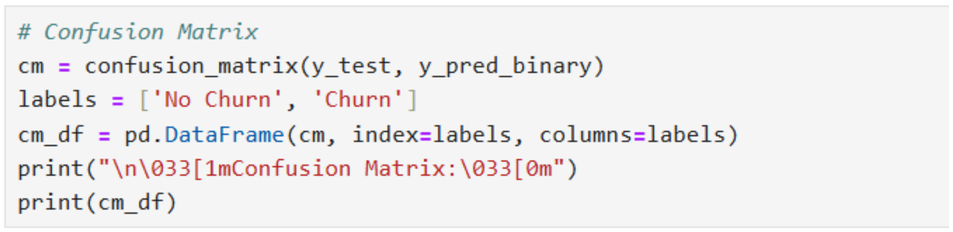
Disadvantage: Using accuracy alone may be misleading in imbalanced datasets, as it doesn't reflect how well the model handles minority classes.

```
# Convert predicted probabilities to binary labels using 0.5 threshold
y_pred_binary = [1 if prob >= 0.5 else 0 for prob in y_pred]

# Accuracy Score
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"\n\033[1mAccuracy:\033[0m {accuracy:.4f}")

# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred)
print(f"\033[1mROC-AUC:\033[0m {roc_auc:.4f}")

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

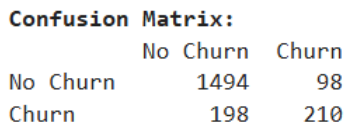
```
Accuracy: 0.8520
ROC-AUC: 0.8511
```



The confusion matrix provided further insight into classification behavior, revealing 1494 true negatives and 210 true positives, alongside 98 false positives and 198 false negatives. These results suggest that the model effectively identified most customers in both classes. However, the presence of 198 false negatives, churned customers misclassified as retained, highlights an opportunity for further refinement.



```python
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_binary)
labels = ['No Churn', 'Churn']
cm_df = pd.DataFrame(cm, index=labels, columns=labels)
print("\n\033[1mConfusion Matrix:\033[0m")
print(cm_df)
```



```
Confusion Matrix:
          No Churn  Churn
No Churn      1494     98
Churn          198    210
```

Advantage: Confusion matrix offers granular insight into classification behavior, supporting targeted model improvements.

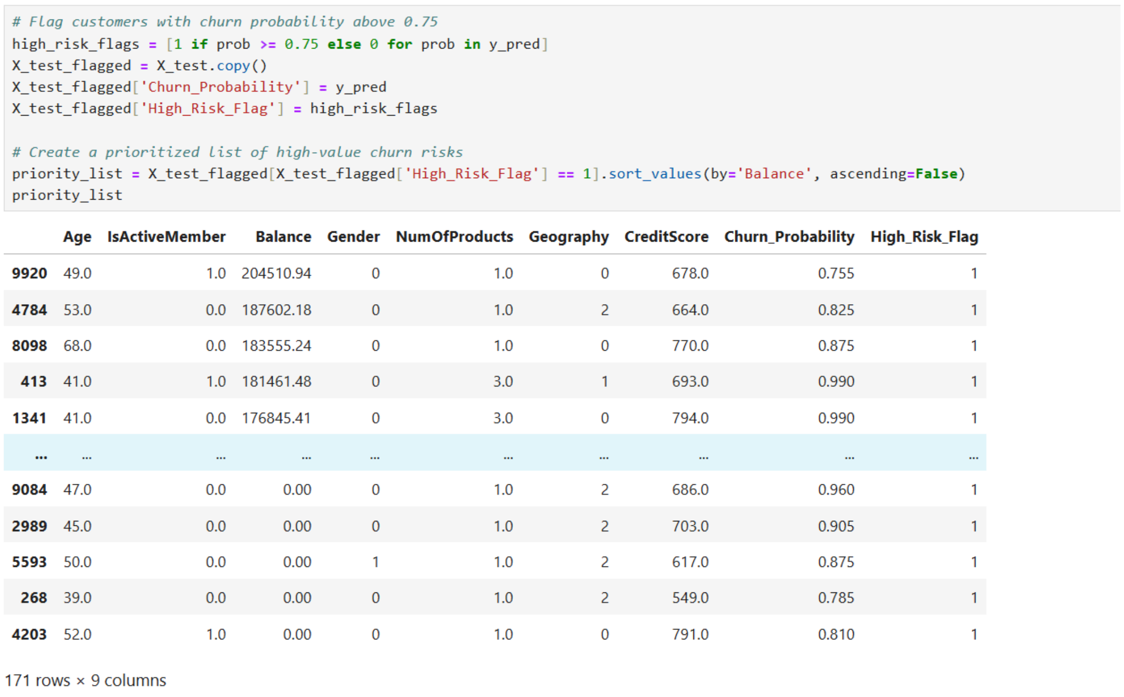
Disadvantage: Stakeholders may struggle to interpret raw counts without context. For example, knowing there are 198 false negatives is less actionable than understanding the cost or impact of those misclassifications

These metrics indicate that the model was able to correctly classify a substantial majority of both churned and retained customers. While the overall accuracy remained strong at 85.2%, the presence of 198 false negatives suggests that some churned customers were misclassified as retained, which may warrant further threshold tuning or feature refinement. Nonetheless, the model demonstrated solid generalizability and precision, with a ROC-AUC score of 0.8511 confirming its ability to effectively distinguish between the two classes across varying probability thresholds.

Risk-Based Targeting

To support targeted retention efforts, the model's predicted probabilities were used to flag customers with a high likelihood of churn. Specifically, any customer with a predicted churn probability greater than 0.75 was assigned a binary High_Risk_Flag. This threshold was chosen to emphasize precision over recall, ensuring that flagged individuals represent the most confident churn predictions.

The flagged dataset was then enriched with each customer's predicted probability and sorted by account balance in descending order. This produced a priority list of high-value customers at elevated risk of churn, those whose departure would likely have the greatest financial impact. By combining churn likelihood with account value, this approach enables stakeholders to allocate retention resources more effectively, focusing on customers whose retention offers the highest ROI. This prioritization framework transforms raw model output into actionable intelligence, bridging the gap between analytics and operational decision-making.



```python
# Flag customers with churn probability above 0.75
high_risk_flags = [1 if prob >= 0.75 else 0 for prob in y_pred]
X_test_flagged = X_test.copy()
X_test_flagged['Churn_Probability'] = y_pred
X_test_flagged['High_Risk_Flag'] = high_risk_flags

# Create a prioritized list of high-value churn risks
priority_list = X_test_flagged[X_test_flagged['High_Risk_Flag'] == 1].sort_values(by='Balance', ascending=False)
priority_list
```

	Age	IsActiveMember	Balance	Gender	NumOfProducts	Geography	CreditScore	Churn_Probability	High_Risk_Flag
9920	49.0	1.0	204510.94	0	1.0	0	678.0	0.755	1
4784	53.0	0.0	187602.18	0	1.0	2	664.0	0.825	1
8098	68.0	0.0	183555.24	0	1.0	0	770.0	0.875	1
413	41.0	1.0	181461.48	0	3.0	1	693.0	0.990	1
1341	41.0	0.0	176845.41	0	3.0	0	794.0	0.990	1
...	...	...	...	...	...	...	...	...	...
9084	47.0	0.0	0.00	0	1.0	2	686.0	0.960	1
2989	45.0	0.0	0.00	0	1.0	2	703.0	0.905	1
5593	50.0	0.0	0.00	1	1.0	2	617.0	0.875	1
268	39.0	0.0	0.00	0	1.0	2	549.0	0.785	1
4203	52.0	1.0	0.00	0	1.0	0	791.0	0.810	1

171 rows × 9 columns

Advantage: Aligns predictive insights with business strategy by surfacing high-risk, high-value customers for proactive outreach.

Disadvantage: May overlook lower-balance customers who churn frequently or influence broader customer sentiment, limiting holistic retention strategies.

Data Summary and Implications

Implications of the Analysis

The Random Forest model demonstrated strong predictive capability in identifying customers at elevated risk of churn, with particular sensitivity to those holding high account balances, enabling a strategically targeted retention approach. The classifier achieved an accuracy of 85.2%, indicating that a substantial majority of predictions aligned with actual outcomes in the test set. Additionally, the model yielded a ROC-AUC score of 0.8511, reflecting robust discriminatory power in distinguishing between retained and exited customers across varying probability thresholds. A ROC-AUC curve was plotted to visually reinforce this performance.

Based on these results, we reject the null hypothesis, which suggested that the model would not exceed 70% predictive accuracy, in favor of the alternative hypothesis: The Random Forest model achieves a predictive accuracy greater than 70% in identifying customer churn.

This outcome provides compelling statistical evidence that the model significantly outperforms the baseline threshold, validating its utility for operational deployment.

To further align predictive insights with business impact, a dual-threshold classification strategy was implemented (0.5 for general detection, 0.75 for high-risk flagging). This approach balanced broad churn awareness with precision-targeted intervention, surfacing individuals whose potential departure would likely result in the greatest financial loss. The framework not only enhances model interpretability but also strengthens its relevance to stakeholder decision-making, bridging technical performance with strategic retention priorities.

Limitation

While the model demonstrated strong predictive accuracy, its interpretability remains limited. Random Forests are complex, and their inner workings aren’t easy to understand, which can be a challenge in regulated industries or when sharing results with non-technical teams.

Recommended Course of Action

Implement a retention strategy that prioritizes outreach to customers flagged with high churn probability and substantial account value. This approach maximizes ROI by focusing resources on individuals whose retention would yield the greatest financial benefit. Additionally, stakeholder-facing dashboards should be developed to visualize churn risk and account impact, enhancing transparency and operational decision-making.

Future Directions

Segment-Specific Modeling: Build separate models for different customer segments (for example, by geography, tenure, or product usage). By tailoring interventions to the unique behaviors and risks of each segment, it enhances targeting precision and improves interpretability for stakeholders through contextualized churn insights. Segment-specific modeling also supports strategic prioritization by aligning retention efforts with segment-level business value and helps mitigate bias and improve fairness, especially when dominant segments distort global performance.

Behavioral and Sentiment Enrichment: Augment the dataset with customer interaction logs, support tickets, or sentiment scores from surveys and reviews. This allows the model to capture emotional and behavioral signals that traditional metrics like account balance or tenure often miss. This enrichment helps uncover latent churn drivers such as frustration, dissatisfaction, or disengagement, factors that may precede transactional changes. By integrating these qualitative dimensions, the model gains a more holistic view of customer risk, enabling earlier and more accurate detection of at-risk users. It also empowers the business to design empathetic, context-aware retention strategies that go beyond financial incentives, fostering stronger customer relationships and long-term loyalty.

Sources

Meshram, S. (2023a, April 27). Bank customer churn prediction. Kaggle. https://www.kaggle.com/datasets/shubhammeshram579/bank-customer-churn-prediction

Connor, M. (2025, February 8). Understanding cardinality in detail: Its definition, importance, and impact on data analysis and... Medium. https://medium.com/@azizozmen/understanding-cardinality-in-detail-its-definition-importance-and-impact-on-data-analysis-and-1f080db41978

Ampawanon, N. (2023, August 27). The pitfall of correlated features: Diluted feature importance in Data Science. Medium. https://medium.com/@nuntaputampawanon/the-pitfall-of-correlated-features-diluted-feature-importance-in-data-science-68d71d3521f9

Kundu, N. (2022, December 27). The advantages and disadvantages of Random Forest: A comprehensive look at the machine learning... Medium. https://medium.com/@nitishkundu1993/the-advantages-and-disadvantages-of-random-forest-a-comprehensive-look-at-the-machine-learning-36c3417003e7