Final Project Capstone:

Bank Customer Churn Analysis

Ruchel Weissman
Date: 6/3/24
Project Duration: 5/7/42 - 7/25/24

Introduction

Customer churn, or the loss of clients, is a significant concern for banks and financial institutions. Retaining customers is crucial for sustained business growth and profitability, as acquiring new customers is often more expensive than keeping existing ones. This project aims to predict customer churn using advanced machine learning techniques. By accurately predicting which customers are likely to leave, banks can take proactive measures to improve customer satisfaction and loyalty, thereby reducing churn rates.

In this project, we utilize a dataset containing various customer attributes, such as demographics, account information, and transaction history. Our goal is to build and compare several machine learning models, including Random Forest, XGBoost, and a CNN deep learning model, to identify the most effective approach for predicting churn. We will also explore the impact of different preprocessing steps, such as feature selection and data balancing, on model performance.

By the end of this project, we aim to provide actionable insights and a robust model that can help banks identify at-risk customers and implement targeted retention strategies to enhance customer satisfaction and reduce churn.

Project Brief

High-Level Overview

The goal of this project is to create a robust churn model to better understand why customers leave a bank and to develop effective strategies to retain these customers. By leveraging advanced machine learning and deep learning techniques, this project aims to predict customer churn accurately and provide actionable insights for customer retention.

Objective

The objective of this project is to understand the factors causing customer churn. It aims to develop and compare multiple predictive models to find the best-performing one. Additionally, the project seeks to interpret the results of these models to provide a clear understanding of the impact of various features on churn predictions. By achieving these objectives, the project will provide actionable insights to help the bank retain customers more effectively and efficiently.

Business Value

Keeping existing customers from churning adds value to the business as acquiring new customers is much more expensive than retaining current ones. By understanding and mitigating factors leading to customer churn, the bank can enhance customer satisfaction and loyalty, thereby increasing overall profitability and business sustainability.

High-Level Approach (Methodology)

1. **Data Processing**: Preprocess data ensuring quality and handling any imbalances.
2. **EDA**: Conduct exploratory data analysis to uncover patterns, correlations, and insights in the data.
3. **Data Splitting**: Split data into training and testing sets.

4. **Model Development**: Try multiple predictive models, including XGBoost, KNN algorithm, and CNN.
5. **Model Evaluation**: Evaluate models and validate their accuracy in predicting churn.
6. **Interpretation of Results**: Identify top feature values to interpret and explain model predictions, making the results understandable and actionable.
7. **Validation and Testing**: Test the models and strategies on the testing dataset to validate their effectiveness and refine them as needed.

Key Milestones

- **Project Kickoff**: Define project scope, objectives, and deliverables.
- **Data Collection and Preprocessing**: Gather data and prepare it for analysis.
- **Exploratory Data Analysis**: Perform EDA to gain initial insights.
- **Model Development**: Build and train predictive models.
- **Model Evaluation**: Assess the performance of different models.
- **Interpretation of Results**: Use top features for model interpretation.
- **Validation and Testing**: Validate the models and strategies in a real-world setting.
- **Final Review and Report**: Compile findings, review results, and present recommendations.

In-Scope

- Development and comparison of multiple predictive models.
- Testing and validation of models and strategies.

Out-of-Scope

- Implementation of deep reinforcement learning for real-time decision-making.
- Time series data from each customer over time would be extremely helpful to help determine when individuals churn over a series of years

Discovery (Exploratory Data Analysis)

Data Overview

The dataset includes 1,000 rows and 18 columns, covering customer demographics, account information, and transaction history. The data spans three countries: Spain, France, and Germany.

Data Quality

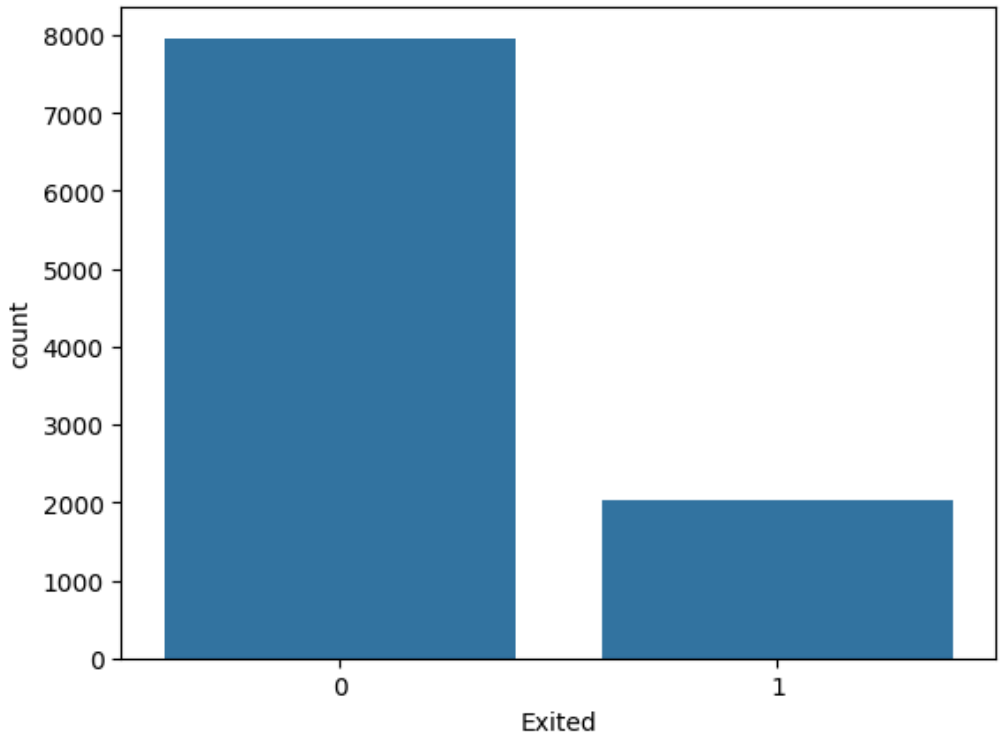
The data is clean, with no missing values and appropriate data types. However, the target variable "Exited" is imbalanced, with only about a quarter of customers having churned.

Descriptive Statistics

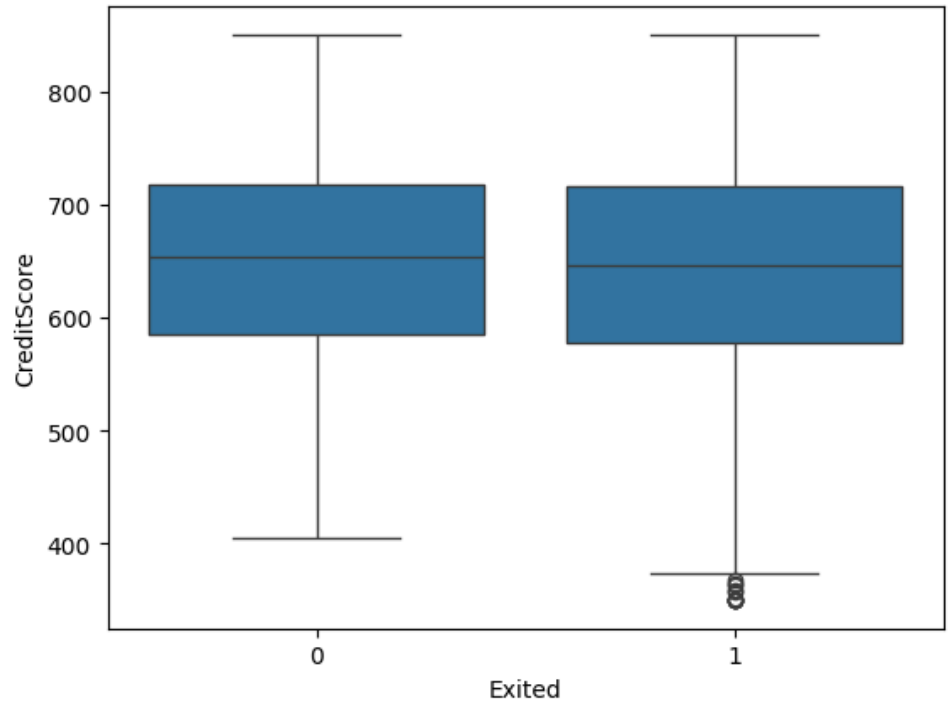
- **Age**: Mean = 39, STD = 10, Min = 18, Max = 92.
- **Tenure**: Mean = 5 years, Max = 10 years.

Target Variable

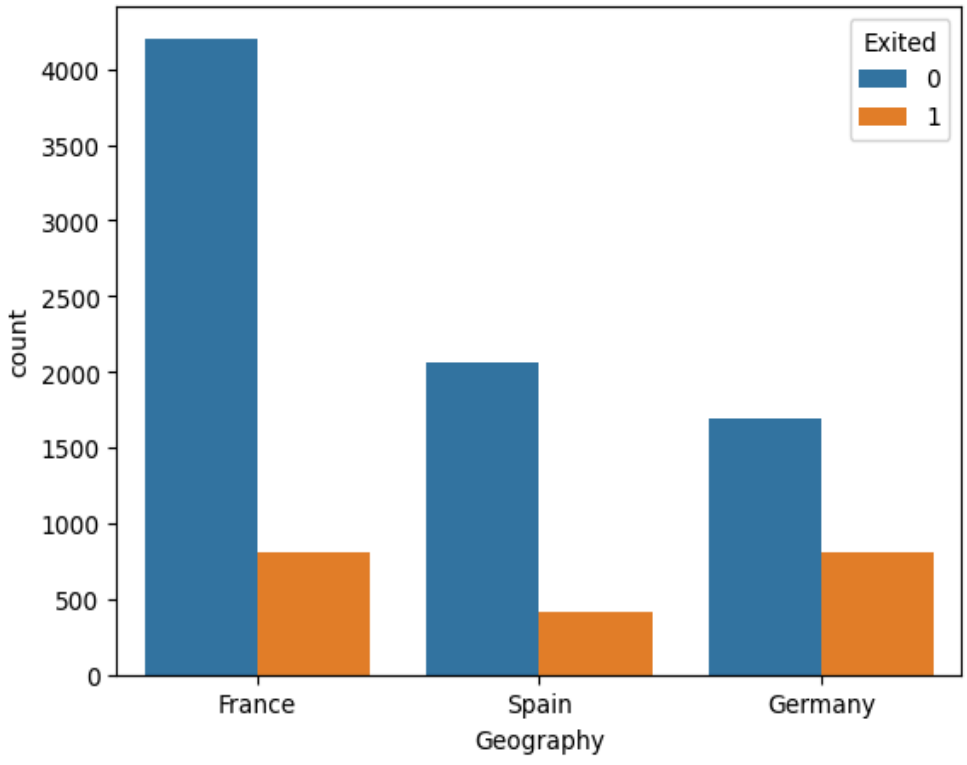
The target variable is "Exited," indicating whether a customer has churned (1) or not (0). The distribution is imbalanced, necessitating techniques to handle this imbalance.



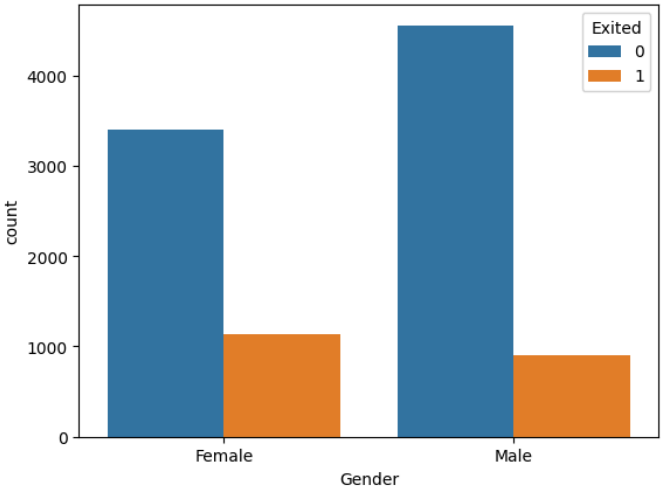
Credit Score: Customers who churn have slightly lower median credit scores and lower outliers compared to those who don't churn. score might not be the best indicator of churn.

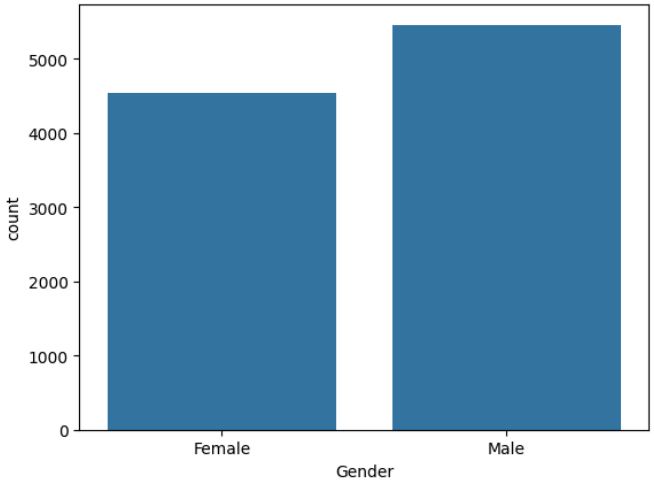


Geography: Most customers are from France, but Germany has the highest churn percentage.

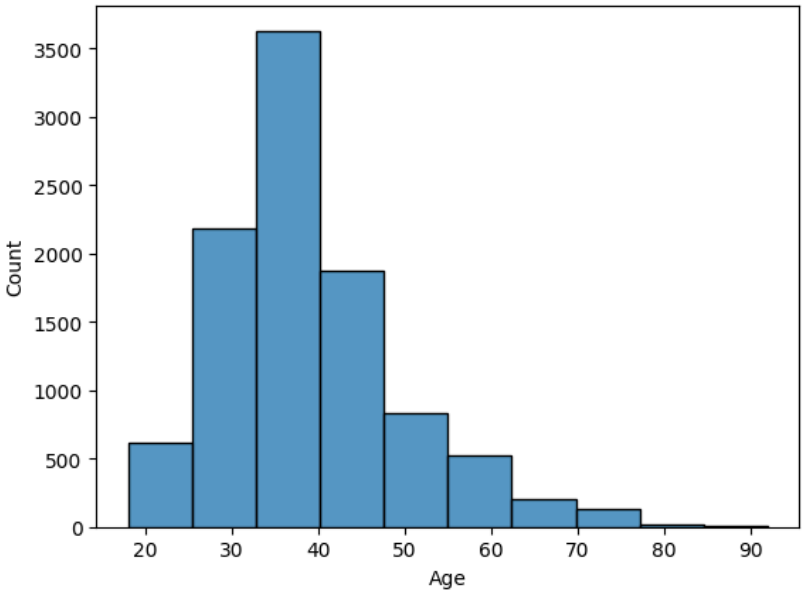


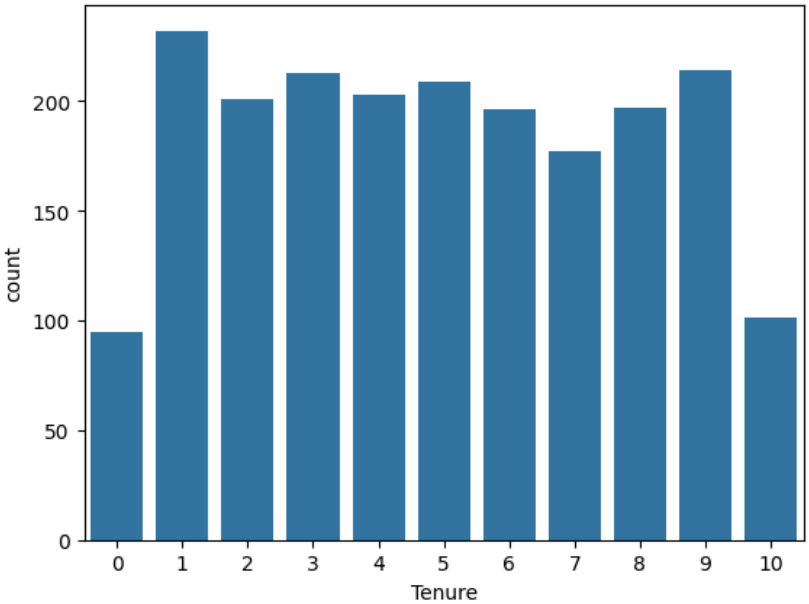
Gender: Slightly more males than females, but females tend to churn more.





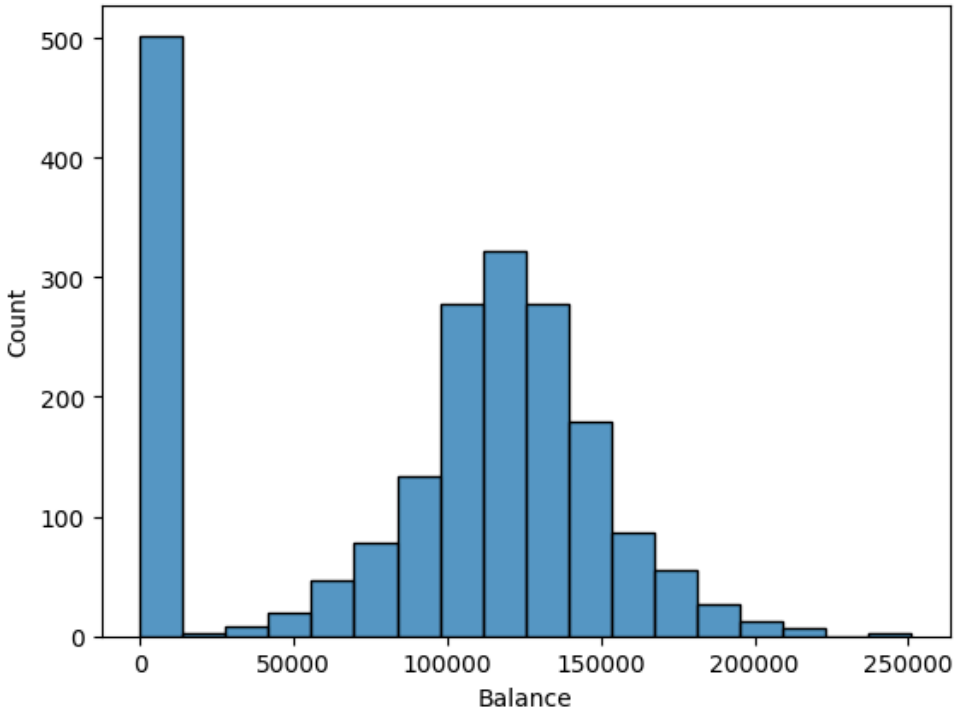
Age: Highest concentration of customers between 30 and 40.





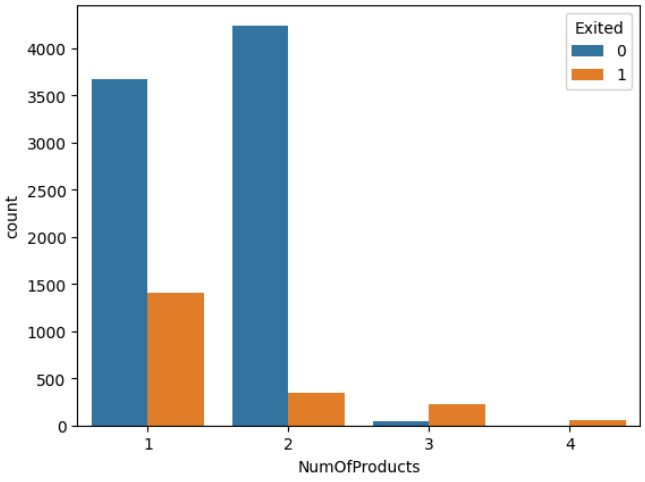
Tenure: One-year tenure has the highest churn count, followed by nine years.

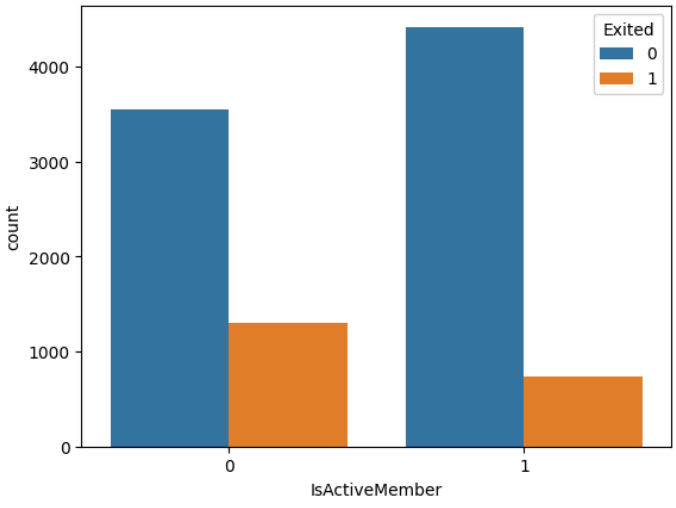
Balance: Customers with a zero balance have a higher churn rate. Maybe we can incentivize customers keeping money in account as it seems to reduce churn.



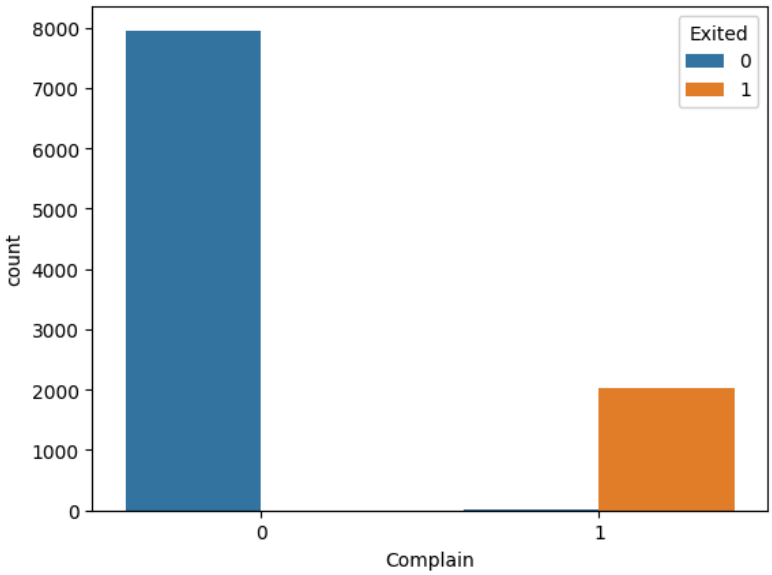
Number of Products: More customers with 3 or 4 products churned compared to those with fewer products. Maybe limit the number of products to 2.

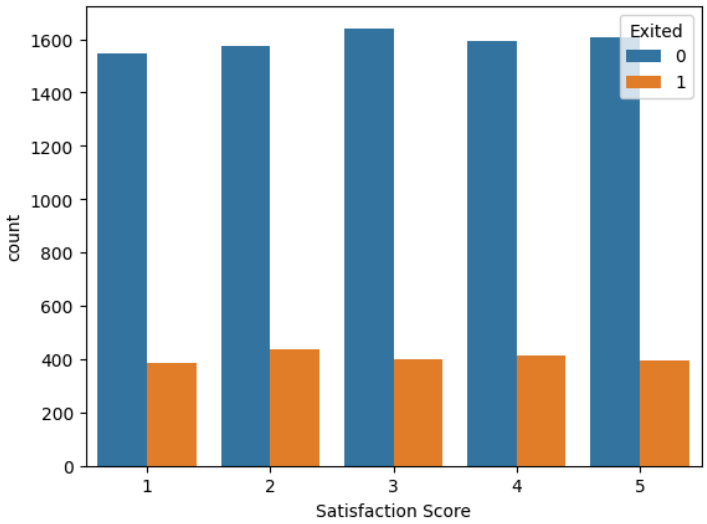
Active Member: Seems like over half of the members aren't active. Try to see why that is and how they measure if the customer is active or not.



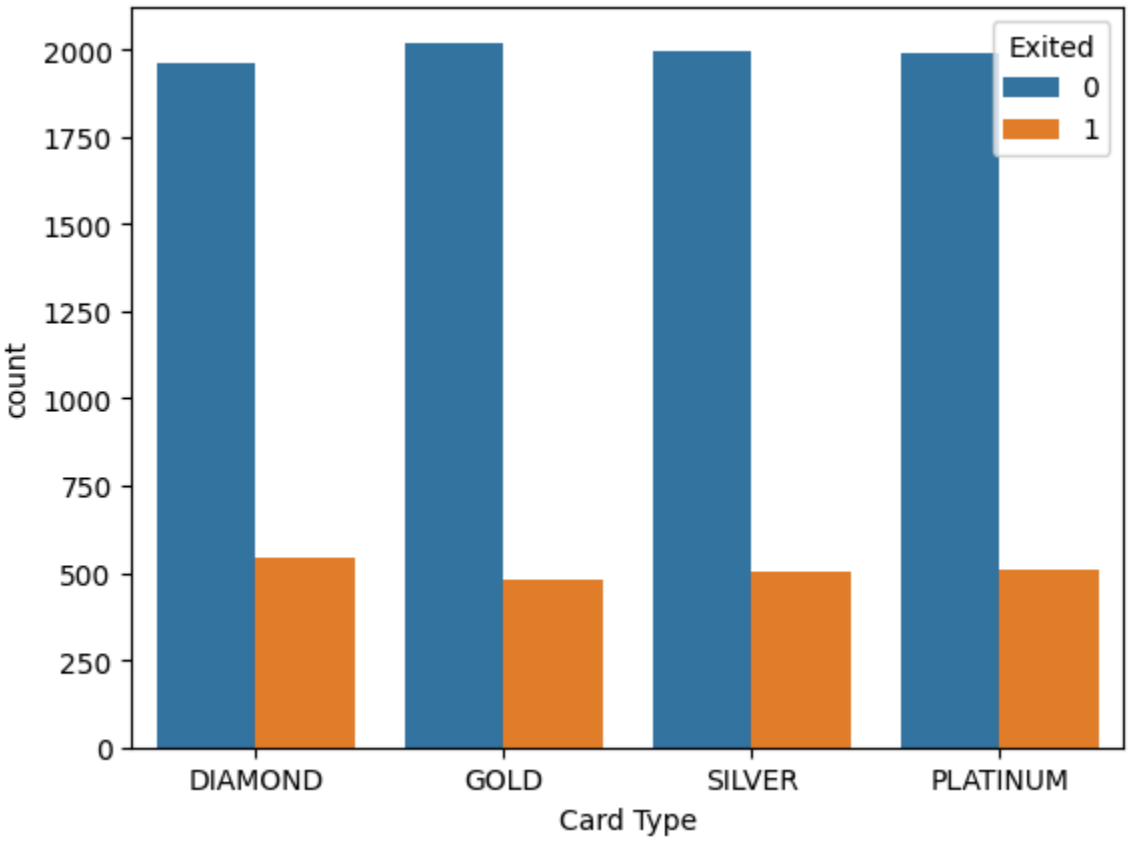


Complaints: Most customers who complain end up churning. Satisfaction score does not significantly indicate churn.

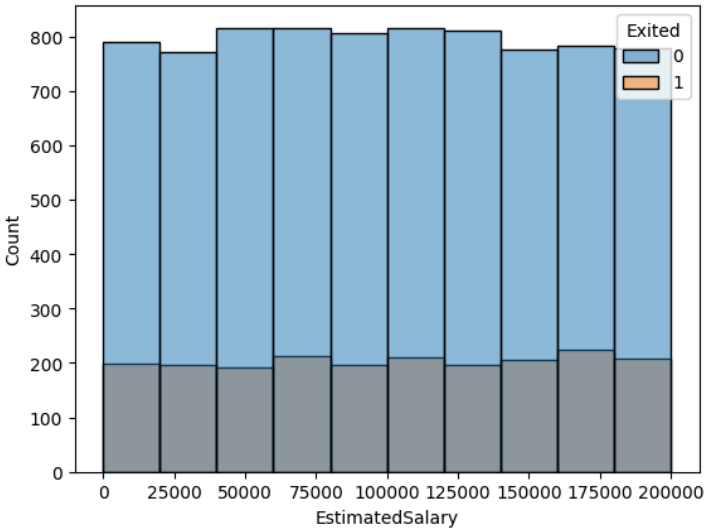


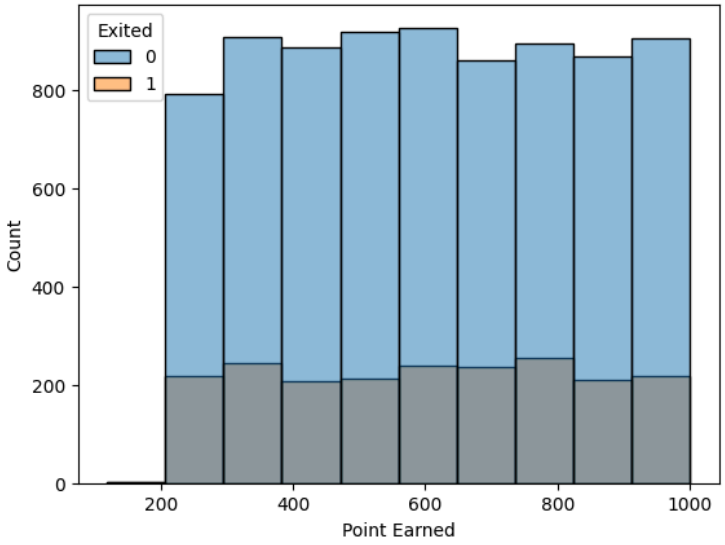


Card Types: Diamond card members churn more than others, which is surprising.

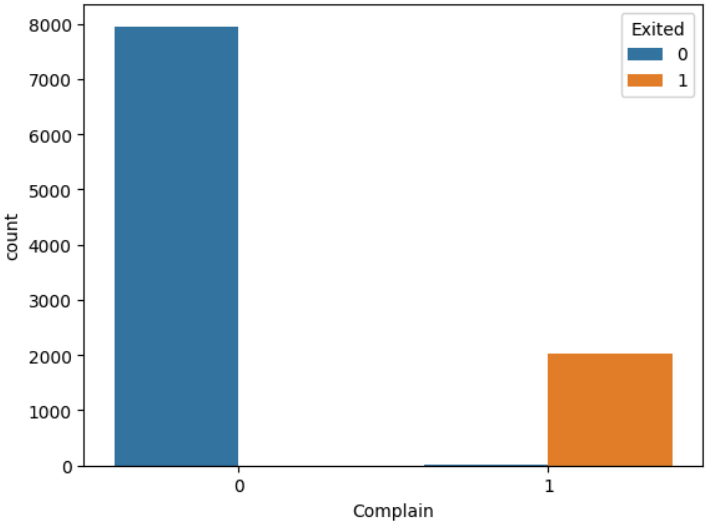


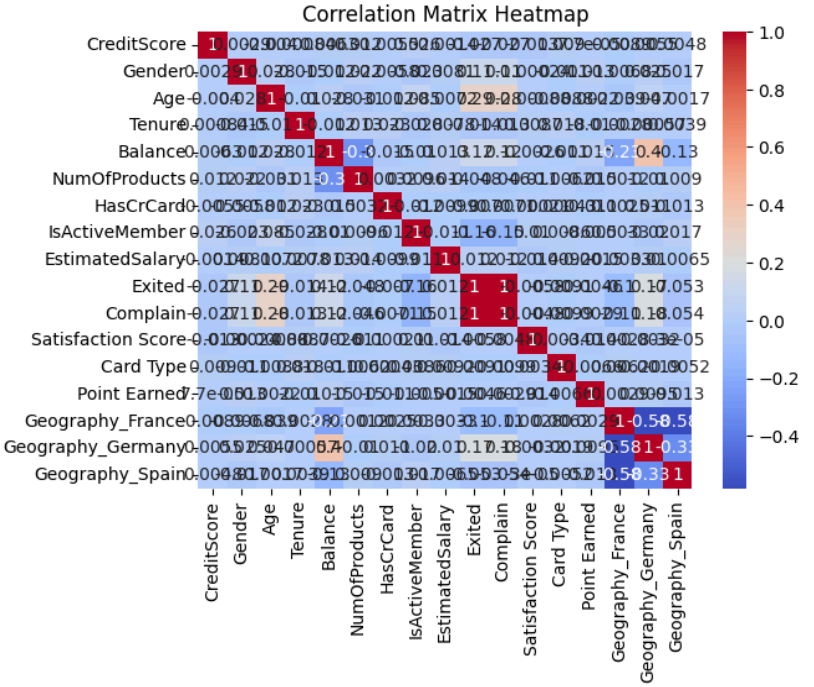
Salary and Points Earned: Do not seem to affect churn significantly.





The only significant correlation is between complaints and exited.





Design - Approach and Methodology Proposal

Relevant Python Packages

pandas: For data manipulation and analysis.

numpy: For numerical operations.

scikit-learn: For implementing machine learning algorithms and evaluation metrics.

xgboost: For implementing the XGBoost algorithm.

matplotlib/seaborn: For data visualization.

TensorFlow: For implementing the CNN model.

imblearn: For SMOTE to balance the dataset and reduce oversampling.

Benefits of This Approach

- **Accuracy:** Using a variety of models, especially advanced ones, to achieve high prediction accuracy.
- **Comprehensive Analysis:** Thoroughly understanding the factors leading to churn through feature importance and interpretability methods.

Limitations and Considerations

- **Complexity**: Advanced models may require significant computational resources and time for training and deployment.
- **Data Quality**: The accuracy of predictions is highly dependent on the quality and completeness of the input data.
- **Interpretability vs. Accuracy Trade-off**: Balancing the need for highly accurate models with the requirement that they be interpretable.
- **Implementation Costs**: High complexity and computational requirements may lead to increased implementation costs.

Predictor(s) and Features

- **Customer demographics**: Age, gender, geography.
- **Account information**: Balance, number of products, tenure.
- **Transactional behavior**: Frequency of transactions, types of transactions.
- **Customer engagement**: Participation in loyalty programs, usage of bank services.
- **Historical data**: Previous complaints.

Target Variable(s)

- **Exited**: A binary variable indicating whether a customer has churned (1) or not (0).

Feature Engineering

- **Complaint**: was removed since it is too highly correlated with the target variable for any of the other variables to matter.
- **Geography**: One-hot encoding for different geographical regions.
- **Gender/ Card Type**: Converted categorical variables to numeric.
- **Row Number/ Customer ID/ Surname**: Removed since not relevant to churn.

Overview

The Bank Customer Churn Analysis project aims to identify the factors contributing to customer churn and develop predictive models to assist the bank in retaining its customers. The project involves analyzing a dataset containing information from three countries—Spain, France, and Germany—tracking various personal and account-related details for 1,000 customers. The target variable is the "Exited" column, indicating whether a customer has left the bank.

Preliminary Model Results

Models Considered

Several machine learning models were evaluated to predict customer churn, including:

Gaussian Naive Bayes (GaussianNB): A probabilistic classifier based on Bayes' theorem with strong (naive) independence assumptions.
Support Vector Classifier (SVC): A supervised learning model used for classification by finding the hyperplane that best separates the data into classes.
K-Nearest Neighbors (KNeighborsClassifier): A simple, instance-based learning algorithm that classifies a data point based on the majority class among its k-nearest neighbors.
Logistic Regression: A statistical model that estimates the probability of a binary outcome based on one or more predictor variables.
Random Forest Classifier (RandomForestClassifier): An ensemble learning method that builds multiple decision trees and merges them to get a more accurate and stable prediction.
Extreme Gradient Boosting (XGBClassifier): A powerful ensemble machine learning algorithm based on the gradient boosting framework.
Gaussian Mixture: A probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.
Cnn deep learning model: A deep learning model that uses convolutional layers to capture spatial hierarchies in the data.

Model Performance

The data was divided into training and testing sets to ensure the model's performance on unseen data. Various preprocessing techniques, feature selection methods, and machine learning models were explored to achieve the best performance. Initially, the dataset included a feature, complaint, which was highly correlated with the target variable exited. Including this feature led

to an overfitted model with nearly 99.9% accuracy, indicating that the model was only using the complaint data rather than learning patterns. Consequently, the complaint feature was removed, which resulted in more realistic model performance.

The dataset was imbalanced, with only about a quarter of the data representing churned customers. To address this, the Synthetic Minority Over-sampling Technique (SMOTE) from the imblearn library was applied to balance the dataset by generating synthetic examples of the minority class (churned customers). This helped improve model performance and reduce bias towards the majority class. This approach improved F1 scores and recall, which are critical metrics for a churn prediction model.

I prepared 3 versions of the data set to test which one performs best.

Regular version, scaled version, balanced version. I created a function that tries it on a list of models.

Model Evaluation

- **Confusion Matrix**: Used to evaluate model performance, showing true positives, false positives, true negatives, and false negatives.
- **Precision-Recall- F1 Score**: Metrics that measure the accuracy, completeness, and balance of the model's positive predictions.
- **Accuracy**: The proportion of correctly predicted instances among the total instances.

Initial Model Performance

Regular Data

model	cm	f1	recall	precision	accuracy	auc
GaussianNB	[[1555 52] [366 27]]	0.114407	0.068702	0.341772	0.791000	0.518172
SVC	[[1607 0] [393 0]]	0.000000	0.000000	0.000000	0.803500	0.500000
KNeighborsClassifier	[[1494 113] [357 36]]	0.132841	0.091603	0.241611	0.765000	0.510643
LogisticRegression	[[1569 38] [363 30]]	0.130152	0.076336	0.441176	0.799500	0.526345
RandomForestClassifier	[[1550 57] [219 174]]	0.557692	0.442748	0.753247	0.862000	0.703639
XGBClassifier	[[1511 96] [192 201]]	0.582609	0.511450	0.676768	0.856000	0.725856
GaussianMixture	[[1607 0] [393 0]]	0.000000	0.000000	0.000000	0.803500	0.500000

Scaled Data

model	cm	f1	recall	precision	accuracy	auc
GaussianNB	[[1503 104] [240 153]]	0.470769	0.389313	0.595331	0.828000	0.662298
SVC	[[1563 44] [243 150]]	0.511073	0.381679	0.773196	0.856500	0.677150
KNeighborsClassifier	[[1522 85] [258 135]]	0.440457	0.343511	0.613636	0.828500	0.645309
LogisticRegression	[[1544 63] [313 80]]	0.298507	0.203562	0.559441	0.812000	0.582179
RandomForestClassifier	[[1553 54] [219 174]]	0.560386	0.442748	0.763158	0.863500	0.704573
XGBClassifier	[[1511 96] [192 201]]	0.582609	0.511450	0.676768	0.856000	0.725856
GaussianMixture	[[1607 0] [393 0]]	0.000000	0.000000	0.000000	0.803500	0.500000

Balanced Data

model	cm	f1	recall	precision	accuracy	auc
GaussianNB	[[1073 534] [117 276]]	0.458853	0.702290	0.340741	0.674500	0.684997
SVC	[[606 1001] [96 297]]	0.351271	0.755725	0.228814	0.451500	0.566413
KNeighborsClassifier	[[983 624] [230 163]]	0.276271	0.414758	0.207116	0.573000	0.513229
LogisticRegression	[[1013 594] [126 267]]	0.425837	0.679389	0.310105	0.640000	0.654878
RandomForestClassifier	[[1458 149] [168 225]]	0.586701	0.572519	0.601604	0.841500	0.739900
XGBClassifier	[[1437 170] [158 235]]	0.588972	0.597964	0.580247	0.836000	0.746089
GaussianMixture	[[1607 0] [393 0]]	0.000000	0.000000	0.000000	0.803500	0.500000

Results Summary

Balanced data had highest scores on f1 score and recall on almost all models which is what matters most for churn prediction.
SVC: Highest recall at 0.75, but lower F1 score.

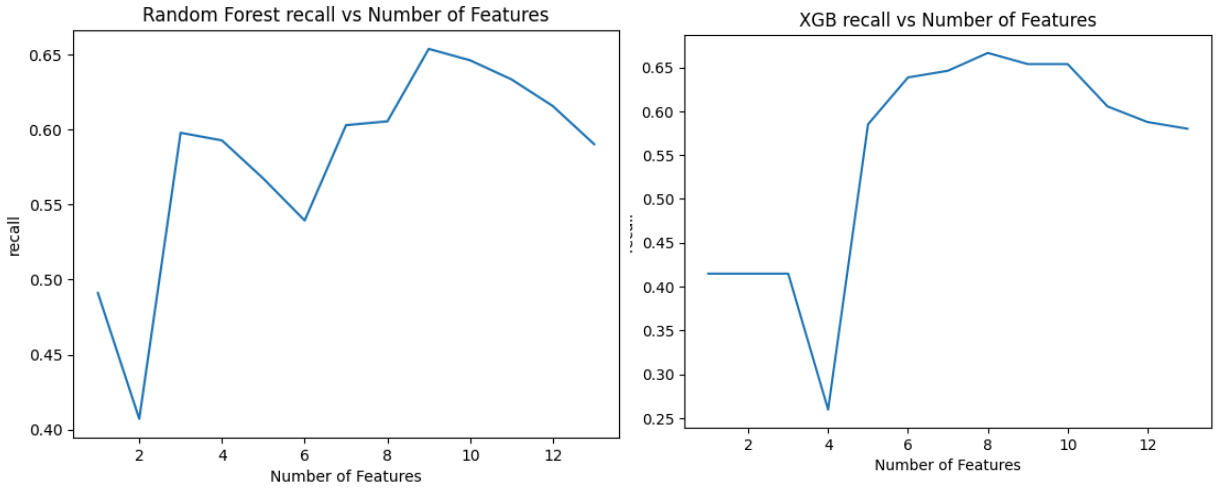
Random Forest: Balanced performance with an F1 score of around 0.58 and accuracy of 0.84.

XGBoost: good recall and F1 scores, very similar f1 and accuracy as random forest model but Highest AUC score of 0.74.

Final Model Selection and Optimization

Focusing on the top-performing models (SVC, Random Forest, and XGBoost), further optimizations were made using Recursive Feature Elimination (RFE) and Principal Component Analysis (PCA).

RFE: Used to identify the top features for the Random Forest and XGBoost models. The optimal number of features was determined using a line plot analysis:



Random Forest: 9 features provided the highest F1 score of 0.59, with an accuracy of 0.82.

XGBoost: 8 features resulted in the best performance with a recall of 0.66 and an F1 score of 0.61.

PCA: Tested to reduce dimensionality. While PCA improved the accuracy of the SVC model to 0.84, the recall dropped to 0.56, indicating a trade-off between precision and recall.

	rfe random forest	pca random forest	rfe xgb	pca xgb	weg svc	pca svc
cm	[[1388 219] [136 257]]	[[1485 122] [181 212]]	[[1411 196] [131 262]]	[[1448 159] [172 221]]	[[606 1001] [96 297]]	[[1461 146] [170 223]]
f1	0.591484	0.583219	0.615746	0.571798	0.351271	0.585302
recall	0.653944	0.539440	0.666667	0.562341	0.755725	0.567430
precision	0.539916	0.634731	0.572052	0.581579	0.228814	0.604336
accuracy	0.822500	0.848500	0.836500	0.834500	0.451500	0.842000

The recall was still the highest at 0.75 for the original weighted svc model. Looking at recall, the SVC model on the regular dataset still performs the best, with the optimized XGBoost second.

If we look at the F1 score, the optimized XGBoost performs best with SVC on PCA data second. Therefore, we will go with the optimized XGBoost model as it is one of the top two on both metrics and performs much better than SVC on other metrics like accuracy and precision.

A grid search with cross-validation was conducted to find the best parameters for SVC, Random Forest, and XGBoost models. The grid search did not significantly increase the scores.

I tried running a more advanced CNN model on various versions of the data as well, but the model did not perform better than the previous models. The highest recall obtained was 0.51, and the highest F1 score was only 0.49. So I decided that this model wasn't worth exploring further.

Final Review and Report

Findings

- **Key Factors**: Age, balance, number of products, has credit card, is active member and geography were identified as significant predictors of customer churn - most seem more personal related and not necessarily something going on from the banks part that they need to improve. So we can use the model to predict who might be likely to churn and offer them products, services to incentivize them to stay.
- **Model Effectiveness**: XGBoost was the most effective model, providing the highest accuracy and F1 score.
- **Actionable Insights**: The model provides valuable insights for developing targeted retention strategies and improving customer satisfaction.

Recommendations

- **Implement the Model**: Deploy the XGBoost model for real-time churn prediction and integrate it with the bank's customer management systems for real time decision making.
- **Monitor and Update**: Regularly monitor model performance and update it with new data to maintain accuracy.
- **Proactive Retention Strategies**: Utilize churn predictions to implement proactive retention strategies and enhance customer satisfaction.
- **Time Series Data:** Time series data from each customer over time would be extremely helpful to help determine when individuals churn over a series of time.

Lessons Learned from the Project

Importance of Feature Selection:
Removing highly correlated features (e.g., the "complaint" column) can prevent overfitting and lead to more realistic model performance.
Feature selection techniques such as Recursive Feature Elimination (RFE) can help in identifying the most significant features, improving both model performance and interpretability.

Handling Imbalanced Data:
Imbalanced datasets can significantly impact model performance.
Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) are effective in balancing the dataset and improving the model's ability to identify minority class instances (e.g., churned customers).

Evaluating Model Performance:
It's important to evaluate models on multiple metrics (accuracy, precision, recall, F1 score, AUC) to get a comprehensive understanding of their performance.

Conclusion

This project successfully developed and evaluated several predictive models for customer churn, with XGBoost emerging as the best-performing model. The insights gained from the model will help the bank implement effective retention strategies, ultimately reducing customer churn and improving overall business performance.

Appendix

Python Code

```
# -*- coding: utf-8 -*-
"""capstone organized.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/14odNFtqtNI-elcMtXyAVu0eOJAXthxfA
"""

#connect to drive
from google.colab import drive
drive.mount('/content/drive')

#import librarys
#import things plan on using
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
#import models want to use including GMM-ASVM, XGBoost, elastic net
methods, KNN algorithm, and the BiLSTM-CNN.
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import ElasticNet
from sklearn.cluster import KMeans
```

```
from sklearn.mixture import GaussianMixture
import tensorflow as tf
from tensorflow.keras.layers import Bidirectional, LSTM, Conv1D,
MaxPooling1D, Dense, Flatten, Dropout
#import pca and other scaling selecting tools
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, f1_score,
classification_report
from sklearn.metrics import precision_score, recall_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import roc_auc_score
# import SMOTE module from imblearn library to balance the yes and no
exited
from imblearn.over_sampling import SMOTE

#load data set
churn = pd.read_csv('drive/MyDrive/datascience/semester 6/capstone
project/Customer-Churn-Records.csv')

print(churn.shape)

print(churn.info())

print(churn.head())

print(churn.isna().sum())

print(churn.describe())

#see how many people churned
sns.countplot(data=churn, x='Exited')
plt.show()
```

```
#only about a quater churned data is imbalanced will have to work around
that

sns.boxplot(data=churn, x='Exited', y= 'CreditScore')
plt.show()
#people who churns credit scores are very similar to the ones that don't
churn but have a slightly lower median and lower outliers as well

#from where are most bank customers
print(churn['Exited'].unique())
sns.countplot(data=churn, x='Geography', hue='Exited')
plt.show()
#most are from france but germany has the highest churn percentage

sns.countplot(data=churn, x='Gender')
plt.show()

#which gender churns most
sns.countplot(data=churn, x='Gender', hue='Exited')
plt.show()
#more male customers overall but females churn more

#which age most clients
sns.histplot(data=churn, x='Age',bins=10)
plt.show()

#divide dataset in two- one that did churn and one that didn't
yes_churn = churn[churn['Exited']== 1]
no_churn = churn[churn['Exited']== 0]

#which age most clients who churn
sns.histplot(data=yes_churn, x='Age', bins=10)
plt.show()

#which age most clients who don't churn
sns.histplot(data=no_churn, x='Age', bins=10)
plt.show()

#tenure of most clients who  churn
sns.countplot(data=yes_churn, x='Tenure')
```

```
plt.show()
#one year tenure has the highest churn count with after that 9 years
highest. seems like maybe some customers try it out for 1 year and leave
and
# other highest get bored or move on after 9 years.

sns.histplot(data=yes_churn, x='Balance')
plt.show()
#largest # of people who churn have a 0 balance
#maybe incentivize customers keeping money in account as seems to reduce
churn

sns.countplot(data=churn, x='NumOfProducts', hue='Exited')
plt.show()
#intrestingly more customers with 3 or 4 products churned than without.
maybe limit # of products to 2.

sns.countplot(data=churn, x='HasCrCard', hue='Exited')
plt.show()

sns.countplot(data=churn, x='IsActiveMember', hue='Exited')
plt.show()
#seems like over half members aren't active. Try to see why that is and
how do the measure if customer is active or not.

sns.countplot(data=churn, x='Complain', hue='Exited')
plt.show()
#most customers who complain end up churning. need to adress their
complaints.

sns.countplot(data=churn, x='Satisfaction Score', hue='Exited')
plt.show()
#satisfaction score dosen't seem to be much of an indicator of churn. A
satisfaction rate of 1 does not have
#visibly greater churn than a 5 sr.

sns.countplot(data=churn, x='Card Type', hue='Exited')
plt.show()
#card types also don't seem to be much of difference although diamond card
members do seem
```

```
#to churn bit more than others which is surprising

sns.histplot(data=churn, x='EstimatedSalary', hue='Exited', bins=10)
plt.show()
#salary is pretty evenly distributed and does not seem to make a big
difference on churn

sns.histplot(data=churn, x='Point Earned', hue='Exited', bins=10)
plt.show()
#points earned does not seem to effect churn very much

#see correlation
#remove non applicable columns to do the correlation
churnCorr = churn.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
churnCorr = pd.get_dummies(churnCorr, columns=['Geography'])
#change male and female for 0 and 1
churnCorr['Gender'] = churn['Gender'].replace({'Male': 0, 'Female': 1})

#change each cardtype for a differnet number
print(churn['Card Type'].unique())
churnCorr['Card Type'] = churn['Card Type'].replace({'PLATINUM': 1,
'DIAMOND': 2, 'GOLD': 3, 'SILVER': 4})
# Calculate the correlation matrix
correlation_matrix = churnCorr.corr()
# Heatmap of the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()

#preprocess notebook

#remove columns not using
#remove non applicable columns to do the correlation also remove complain
since it's too highly correlated
churn = churn.drop(['RowNumber', 'CustomerId', 'Surname', 'Complain'],
axis=1)

#map categorical variables to #s maybe do one hot encoding for georaphy
cause it's not ordinal
#do one hot encoding for geography
```

```
churn = pd.get_dummies(churn, columns=['Geography'])

#change male and female for 0 and 1
churn['Gender'] = churn['Gender'].replace({'Male': 0, 'Female': 1})

#change each cardtype for a differnet number
print(churn['Card Type'].unique())
churn['Card Type'] = churn['Card Type'].replace({'PLATINUM': 1, 'DIAMOND':
2, 'GOLD': 3, 'SILVER': 4})
print(churn.head())

#Split data into testing and training sets.
X = churn.drop(['Exited'], axis=1)
y = churn['Exited']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#do seperate variable of balanced data
# import SMOTE module from imblearn library to balance the yes and no
exited
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

#create function that tries all models

def try_models(X_train, y_train,X_test, y_test):
    scores = []
    models = [
        GaussianNB(),
        SVC(random_state=42),
        KNeighborsClassifier(),
        LogisticRegression(random_state=42),
        RandomForestClassifier(random_state=42),
        XGBClassifier(random_state=42),
        GaussianMixture(random_state=42)
```

```
    ]

    for model in models:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        cm = confusion_matrix(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)
        auc = roc_auc_score(y_test, y_pred)
        score = {
            'model': model.__class__.__name__,
            'cm': cm,
            'f1': f1,
            'recall': recall,
            'precision': precision,
            'accuracy': accuracy,
            'auc' : auc
        }
        scores.append(score)

    df = pd.DataFrame(scores)
    # Apply table styles
    styles = [{'selector': 'th', 'props': [('text-align', 'center')]}]
    styled_df = df.style.set_table_styles(styles)
    # Display the styled DataFrame
    styled_df

    return styled_df

try_models(X_train, y_train, X_test, y_test)

try_models(X_train_res, y_train_res, X_test, y_test)

try_models(X_train_scaled, y_train, X_test_scaled, y_test)

#try rfe and pca on weighted data
#pca
pca = PCA()
```

```
#scale for pca
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_res)
X_test_scaled = scaler.transform(X_test)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

#test dropping features from model using rfe 1-13 and graph the accurracy
after droping features
f1_scores = []
recall_scores = []
precision_scores = []
accuracy_scores = []
num_features = []
for x in range(1,14):
  rfe = RFE(estimator=RandomForestClassifier(random_state=42),
n_features_to_select=x)
  rfe.fit(X_train_res, y_train_res)
  selected_features = X_train.columns[rfe.support_]
  X_train_selected = rfe.transform(X_train_res)
  X_test_selected = rfe.transform(X_test)
  model= RandomForestClassifier(random_state=42)
  model.fit(X_train_selected, y_train_res)
  y_pred = model.predict(X_test_selected)
  cm = confusion_matrix(y_test, y_pred)
  f1 = f1_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
  num_features.append(x)
  f1_scores.append(f1)
  recall_scores.append(recall)
  precision_scores.append(precision)
  accuracy_scores.append(accuracy)

# Create a DataFrame for plotting recall
results_f1 = pd.DataFrame({'num_features': num_features, 'f1': f1_scores})
#line graph of score
plt.plot(results_f1['num_features'], results_f1['f1'])
plt.xlabel('Number of Features')
```

```
plt.ylabel('f1')
plt.title('Random Forest f1 score vs Number of Features')
plt.show()

# Create a DataFrame for plotting f1
results_recall = pd.DataFrame({'num_features': num_features, 'recall':
recall_scores})
#line graph of score
plt.plot(results_recall['num_features'], results_recall['recall'])
plt.xlabel('Number of Features')
plt.ylabel('recall')
plt.title('Random Forest recall vs Number of Features')
plt.show()

#test dropping features from xgb model using rfe 1-13 and graph the
accurracy after droping features
f1_scores = []
recall_scores = []
precision_scores = []
accuracy_scores = []
num_features = []
for x in range(1,14):
  rfe = RFE(estimator=XGBClassifier(random_state=42),
n_features_to_select=x)
  rfe.fit(X_train_res, y_train_res)
  selected_features = X_train.columns[rfe.support_]
  X_train_selected = rfe.transform(X_train_res)
  X_test_selected = rfe.transform(X_test)
  model= XGBClassifier(random_state=42)
  model.fit(X_train_selected, y_train_res)
  y_pred = model.predict(X_test_selected)
  cm = confusion_matrix(y_test, y_pred)
  f1 = f1_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
  num_features.append(x)
  f1_scores.append(f1)
  recall_scores.append(recall)
  precision_scores.append(precision)
```

```
  accuracy_scores.append(accuracy)

# Create a DataFrame for plotting recall
results_f1 = pd.DataFrame({'num_features': num_features, 'f1': f1_scores})
#line graph of score
plt.plot(results_f1['num_features'], results_f1['f1'])
plt.xlabel('Number of Features')
plt.ylabel('f1')
plt.title('XGB f1 score vs Number of Features')
plt.show()

# Create a DataFrame for plotting f1
results_recall = pd.DataFrame({'num_features': num_features, 'recall':
recall_scores})
#line graph of score
plt.plot(results_recall['num_features'], results_recall['recall'])
plt.xlabel('Number of Features')
plt.ylabel('recall')
plt.title('XGB recall vs Number of Features')
plt.show()

#try3 models on pca data and xgb and random forest with dropped columns

#random forest
randomforestmodel = RandomForestClassifier(random_state=42)



#try on 2 versions of dataset
#regular data
rf_rfe_scores = []
rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=9)
selector = rfe.fit(X_train_res, y_train_res)
#transform data using only top features
X_train_selected_rf = rfe.transform(X_train_res)
X_test_selected_rf = rfe.transform(X_test)
#run model and see if dosen't overfit anymore
randomforestmodel.fit(X_train_selected_rf, y_train_res)

#get metrics and add to table
y_pred = randomforestmodel.predict(X_test_selected_rf)
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
rf_rfe_scores.append(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
rf_rfe_scores.append(f1)
recall = recall_score(y_test, y_pred)
print(recall)
rf_rfe_scores.append(recall)
precision = precision_score(y_test, y_pred)
print(precision)
rf_rfe_scores.append(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
rf_rfe_scores.append(accuracy)

selected_features_mask = rfe.support_
selected_features = X_train_res.columns[selected_features_mask]
print("Selected Features:", selected_features)

# Create a DataFrame
df = pd.DataFrame(selected_features)

# Apply table styles
styles = [{'selector': 'th', 'props': [('text-align', 'center')]}]
styled_df = df.style.set_table_styles(styles)

# Display the styled DataFrame
styled_df

#pca data
rf_pca_scores = []
#run model and see if dosen't overfit anymore
randomforestmodel.fit(X_train_pca, y_train_res)

#get metrics and add to table
y_pred = randomforestmodel.predict(X_test_pca)
cm = confusion_matrix(y_test, y_pred)
print(cm)
rf_pca_scores.append(cm)
```

```
f1 = f1_score(y_test, y_pred)
print(f1)
rf_pca_scores.append(f1)
recall = recall_score(y_test, y_pred)
print(recall)
rf_pca_scores.append(recall)
precision = precision_score(y_test, y_pred)
print(precision)
rf_pca_scores.append(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
rf_pca_scores.append(accuracy)

#xgboost
xgbmodel = XGBClassifier(random_state=42)

#rfe data
rfe = RFE(xgbmodel, n_features_to_select=8)
selector = rfe.fit(X_train_res, y_train_res)
#transform data using only top features
X_train_selected_xgb = rfe.transform(X_train_res)
X_test_selected_xgb = rfe.transform(X_test)
#run model and see if dosen't overfit anymore
xgbmodel.fit(X_train_selected_xgb, y_train_res)

#get metrics and add to table
y_pred = xgbmodel.predict(X_test_selected_xgb)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
rfe_scores_xgb = [cm, f1, recall, precision, accuracy]

#output consusion matrix in nice format
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
#label positive and negitive instead of 0 and 1
#align the labels to each box

plt.xticks([0.5, 1.5], ['Negative', 'Positive'])
plt.yticks([0.5, 1.5], ['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.show()

selected_features_mask = rfe.support_
selected_features = X_train_res.columns[selected_features_mask]
print("Selected Features:", selected_features)

# Create a DataFrame
df = pd.DataFrame(selected_features)

# Apply table styles
styles = [{'selector': 'th', 'props': [('text-align', 'center')]}]
styled_df = df.style.set_table_styles(styles)

# Display the styled DataFrame
styled_df

#pca data
xgbmodel.fit(X_train_pca, y_train_res)

#get metrics and add to table
y_pred = xgbmodel.predict(X_test_pca)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
pca_scores_xgb = [cm, f1, recall, precision, accuracy]

#svc
svc = SVC(random_state=42)

#pca data
svc.fit(X_train_pca, y_train_res)
#get metrics and add to table
y_pred = svc.predict(X_test_pca)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
pca_scores_svc = [cm, f1, recall, precision, accuracy]

#try on weighted data
svc.fit(X_train_res, y_train_res)
#get metrics and add to table
y_pred = svc.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
weg_scores_svc = [cm, f1, recall, precision, accuracy]

titles = ['cm', 'f1', 'recall', 'precision','accuracy' ]
```

```
# Create a DataFrame
modelscores = pd.DataFrame(index=titles)
modelscores.insert(0, 'rfe random forest', rf_rfe_scores)
modelscores.insert(1, 'pca random forest', rf_pca_scores)
modelscores.insert(2, 'rfe xgb', rfe_scores_xgb)
modelscores.insert(3, 'pca xgb', pca_scores_xgb)
modelscores.insert(4, 'pca svc', pca_scores_svc)
modelscores.insert(4, 'weg svc', weg_scores_svc)

# Apply table styles
styles = [{'selector': 'th', 'props': [('text-align', 'center')]}]
modelscores = modelscores.style.set_table_styles(styles)

# Display the styled DataFrame
modelscores

#parameter selection
#random forest
rfmodel = RandomForestClassifier(random_state = 42)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search_rf = GridSearchCV(rfmodel, param_grid, cv=5)
grid_search_rf.fit(X_train_selected_rf, y_train_res)
best_params_randomforest = grid_search_rf.best_params_
print(best_params_randomforest)
best_score = grid_search_rf.best_score_
print(best_score)
#evaluate on test set
y_pred = grid_search_rf.predict(X_test_selected_rf)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
```

```
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
rf_scores_gridsearch = [cm, f1, recall, precision, accuracy]

best_params_randomforest =
pd.DataFrame.from_dict(best_params_randomforest, orient='index')
#set the index as randomforest instead of 0
best_params_randomforest.columns = ['random forest']
print(best_params_randomforest)

# grid search cv on xgboost
xgbmodel = XGBClassifier(random_state = 42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9]
}
grid_search_xgb = GridSearchCV(xgbmodel, param_grid, cv=5)
grid_search_xgb.fit(X_train_pca, y_train_res)
best_params_xgb = grid_search_xgb.best_params_
print(best_params_xgb)
best_score_xgb = grid_search_xgb.best_score_
print(best_score_xgb)
#evaluate on test set
y_pred = grid_search_xgb.predict(X_test_pca)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
xgb_scores_gridsearch = [cm, f1, recall, precision, accuracy]
```

```
best_params_xgb = pd.DataFrame.from_dict(best_params_xgb, orient='index')
#set the index as randomforest instead of 0
best_params_xgb.columns = ['xgb']
print(best_params_xgb)

#grid search on svc
svcmodel = SVC(random_state=42)
param_grid = {
    'gamma': [1, 0.1, 0.01],
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10]
}
grid_search_svc = GridSearchCV(svcmodel, param_grid, cv=5)
grid_search_svc.fit(X_train_res, y_train_res)
best_params_svc = grid_search_svc.best_params_
print(best_params_svc)
best_score_svc = grid_search_svc.best_score_
print(best_score_svc)
#evaluate on test set
y_pred = grid_search_svc.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
svc_scores_gridsearch = [cm, f1, recall, precision, accuracy]

best_params_svc = pd.DataFrame.f
rom_dict(best_params_svc, orient='columns')
#set the index as randomforest instead of 0
best_params_svc.columns = ['svc']
print(best_params_svc)

# Concatenate DataFrames along columns (axis=1)
```

```
combined_df = pd.concat([best_params_randomforest, best_params_xgb,
best_params_svc],axis=1)

# Display the combined DataFrame
#print(combined_df)
combined_df_cleaned = combined_df.fillna('')

print(combined_df_cleaned)

print(rf_scores_gridsearch)
print(xgb_scores_gridsearch)
print(svc_scores_gridsearch)

# Define the CNN model
tfmodel = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, 5, activation='relu',
input_shape=(X_train_res.shape[1], 1)),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model with the Adam optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
tfmodel.compile(loss='binary_crossentropy', optimizer=optimizer,
metrics=['accuracy'])

# Check class distribution in the training data
unique, counts = np.unique(y_train_res, return_counts=True)
print("Training data distribution:", dict(zip(unique, counts)))

# Compute class weights to handle class imbalance
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train_res), y=y_train_res)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)

# Reshape X_train to (n_samples, n_features, 1) for 1D convolution
```

```
X_train_reshaped = np.expand_dims(X_train_res, axis=-1)

# Fit the model on training data with class weights
history = tfmodel.fit(X_train_reshaped, y_train_res, epochs=25,
batch_size=32, validation_split=0.2, class_weight=class_weights)

# Evaluate the model on test data
X_test = np.array(X_test, dtype=np.float32)
y_test = np.array(y_test, dtype=np.int32)
X_test_reshaped = np.expand_dims(X_test, axis=-1)  # Ensure the test data
is reshaped correctly

# Predict using the model
y_pred_probs = tfmodel.predict(X_test_reshaped)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()  # Convert
probabilities to binary class labels

# Compute and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Compute and print other evaluation metrics
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)

# Store the metrics
weg_scores_tf = [cm, f1, recall, precision, accuracy]

#scaled tf model
# Check class distribution in the training data
unique, counts = np.unique(y_train, return_counts=True)
print("Training data distribution:", dict(zip(unique, counts)))

# Compute class weights to handle class imbalance
```

```
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)

# Reshape X_train to (n_samples, n_features, 1) for 1D convolution
X_train_reshaped = np.expand_dims(X_train_scaled, axis=-1)

# Fit the model on training data with class weights
history = tfmodel.fit(X_train_reshaped, y_train, epochs=25, batch_size=32,
validation_split=0.2, class_weight=class_weights)

# Evaluate the model on test data
X_test = np.array(X_test_scaled, dtype=np.float32)
y_test = np.array(y_test, dtype=np.int32)
X_test_reshaped = np.expand_dims(X_test, axis=-1)  # Ensure the test data
is reshaped correctly

# Predict using the model
y_pred_probs = tfmodel.predict(X_test_reshaped)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()  # Convert
probabilities to binary class labels

# Compute and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Compute and print other evaluation metrics
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)

# Store the metrics
scaled_scores_tf = [cm, f1, recall, precision, accuracy]
```

```
#regular tf model
# Check class distribution in the training data
unique, counts = np.unique(y_train, return_counts=True)
print("Training data distribution:", dict(zip(unique, counts)))

# Compute class weights to handle class imbalance
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)

# Reshape X_train to (n_samples, n_features, 1) for 1D convolution
X_train_reshaped = np.expand_dims(X_train, axis=-1)

# Fit the model on training data with class weights
history = tfmodel.fit(X_train_reshaped, y_train, epochs=25, batch_size=32,
validation_split=0.2, class_weight=class_weights)

# Evaluate the model on test data
X_test = np.array(X_test, dtype=np.float32)
y_test = np.array(y_test, dtype=np.int32)
X_test_reshaped = np.expand_dims(X_test, axis=-1)  # Ensure the test data
is reshaped correctly

# Predict using the model
y_pred_probs = tfmodel.predict(X_test_reshaped)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()  # Convert
probabilities to binary class labels

# Compute and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Compute and print other evaluation metrics
f1 = f1_score(y_test, y_pred)
print(f1)
recall = recall_score(y_test, y_pred)
print(recall)
precision = precision_score(y_test, y_pred)
print(precision)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)

# Store the metrics
reg_scores_tf = [cm, f1, recall, precision, accuracy]

titles = ['cm', 'f1', 'recall', 'precision','accuracy' ]
# Create a DataFrame
cnn = pd.DataFrame(index=titles)
cnn.insert(0, 'regular', reg_scores_tf)
cnn.insert(1, 'scaled', scaled_scores_tf)
cnn.insert(2, 'weighted', weg_scores_tf)

# Apply table styles
styles = [{'selector': 'th', 'props': [('text-align', 'center')]}]
cnn= cnn.style.set_table_styles(styles)

# Display the styled DataFrame
cnn
```