Bank Customer Churn Analysis

Ruchel Weissman

July 2024

Business Problem:

- High customer churn impacting revenue and growth.
- The dataset used in this study, referred to as the Bank Churn Dataset, was sourced from a bank's customer database. For each customer it contains both personal details and account-related information.
- The objective of this project is to understand what factors are causing customer churn.

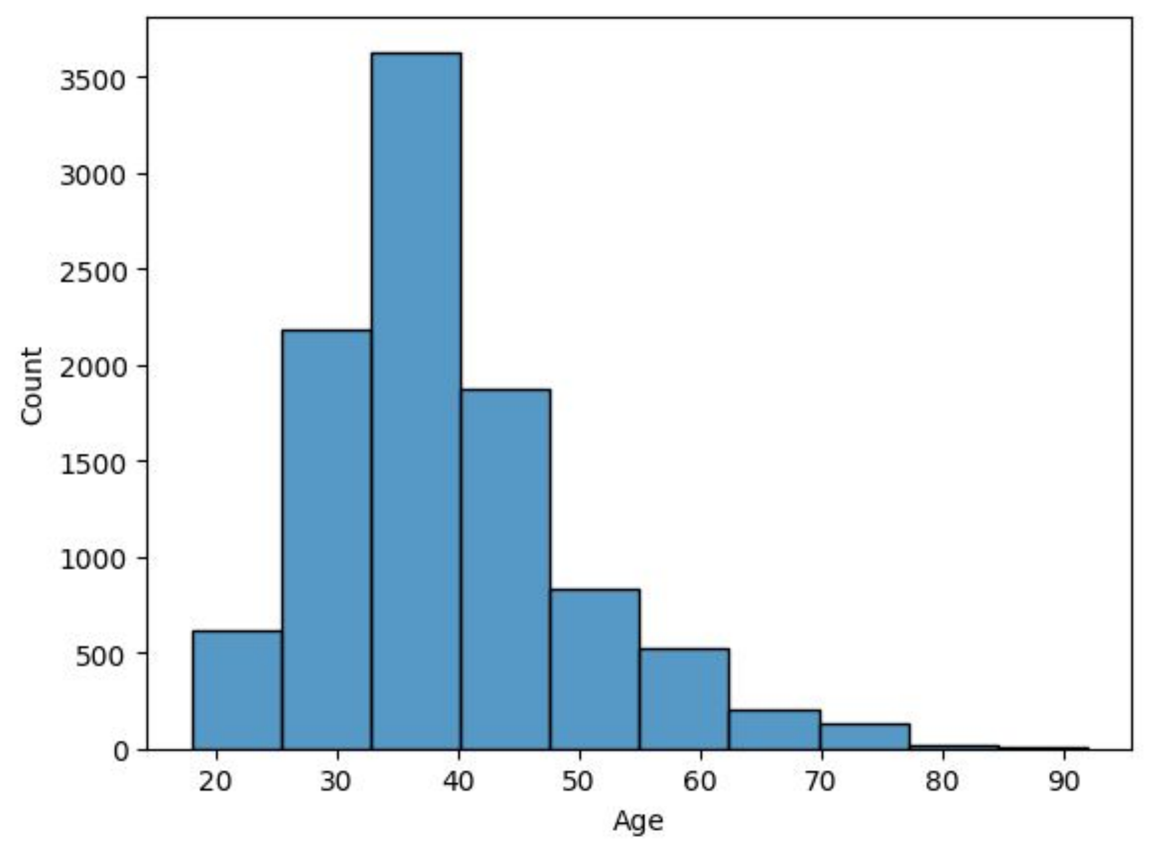
High level approach to solving the problem

The goal of this project is to develop a churn prediction model for the banking sector to better understand the factors leading to customer churn. By identifying these factors and predicting potential churners, the project aims to help the bank devise effective customer retention strategies.

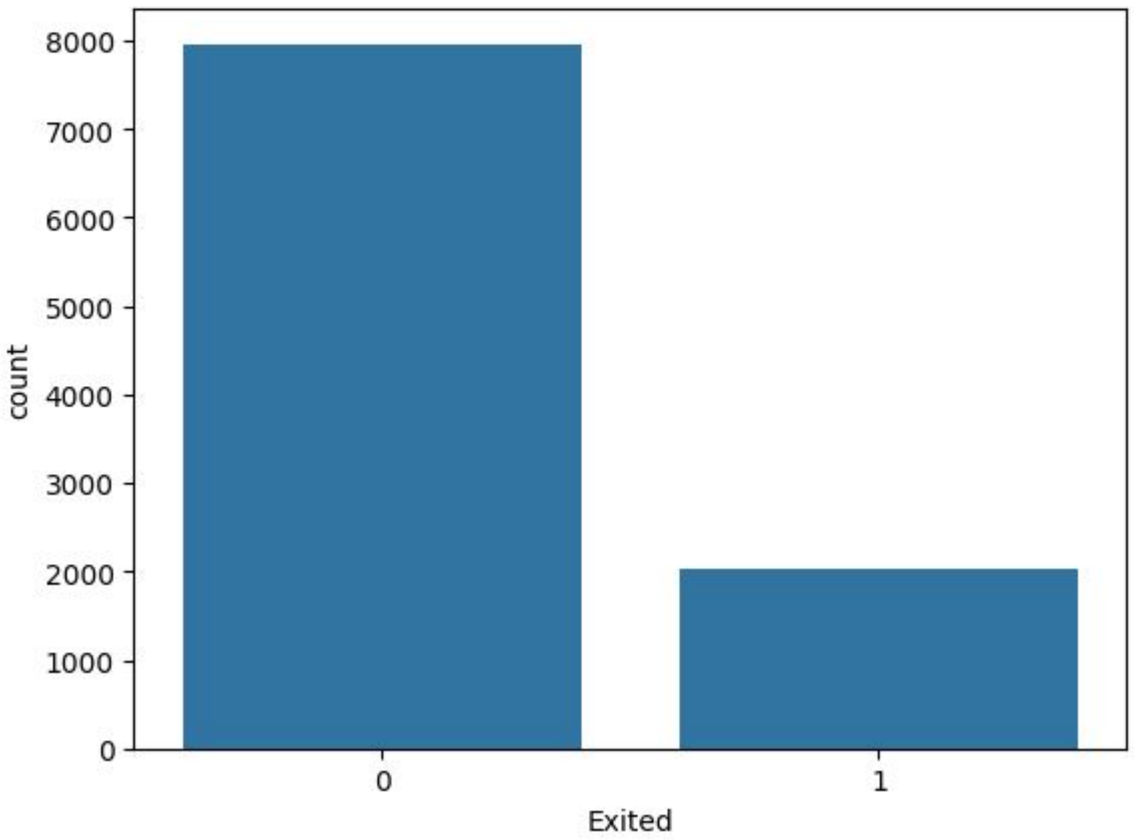
Data

- The dataset contains 18 features for each of the 1,000 customers tracked. The features include age, location, salary, credit card ownership, the number of bank products held, card type, and whether the customer had lodged any complaints.
- The target variable for this study is the "Exited" column, indicating whether a customer has exited the bank.
- The data is from 3 countries: Spain, France and germany.
- Average customer age is 39 with a standard deviation of 10. The min age is 18 max is 92.
- The average tenure is 5 years, the longest is 10.
- The data set was very clean and no missing values were detected.

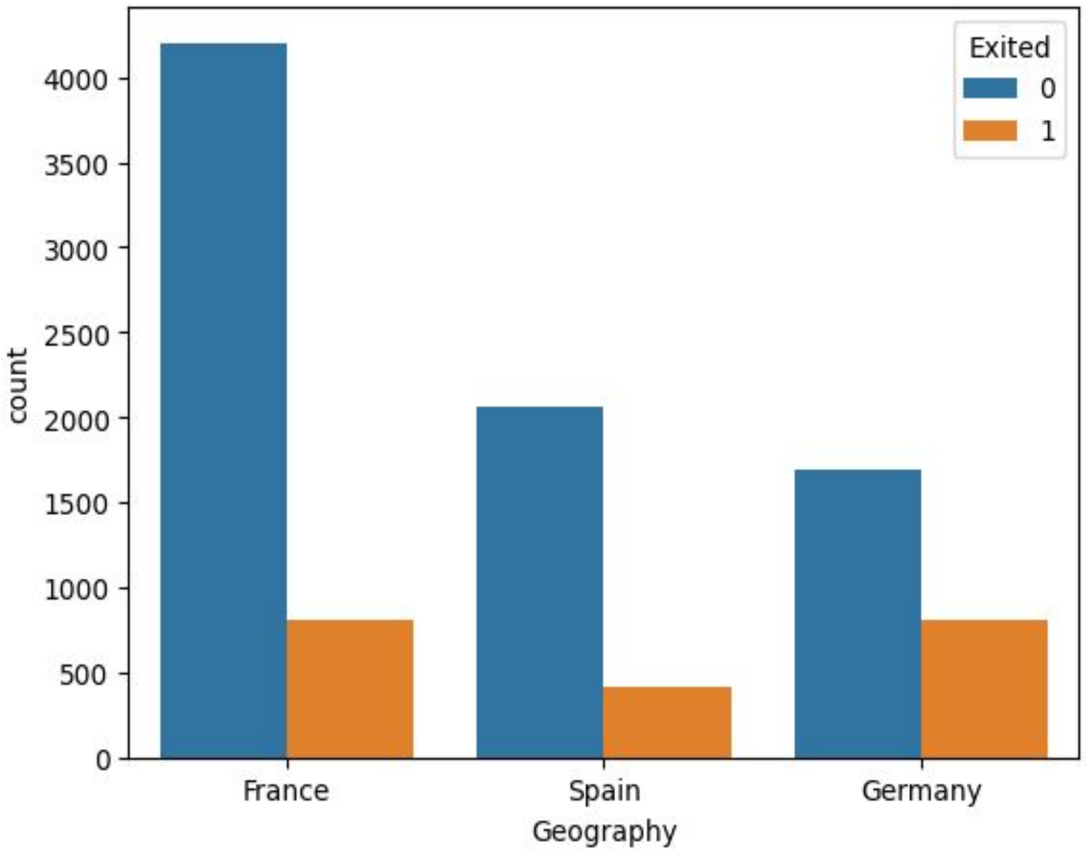
Age Distribution



Exploratory Data Analysis

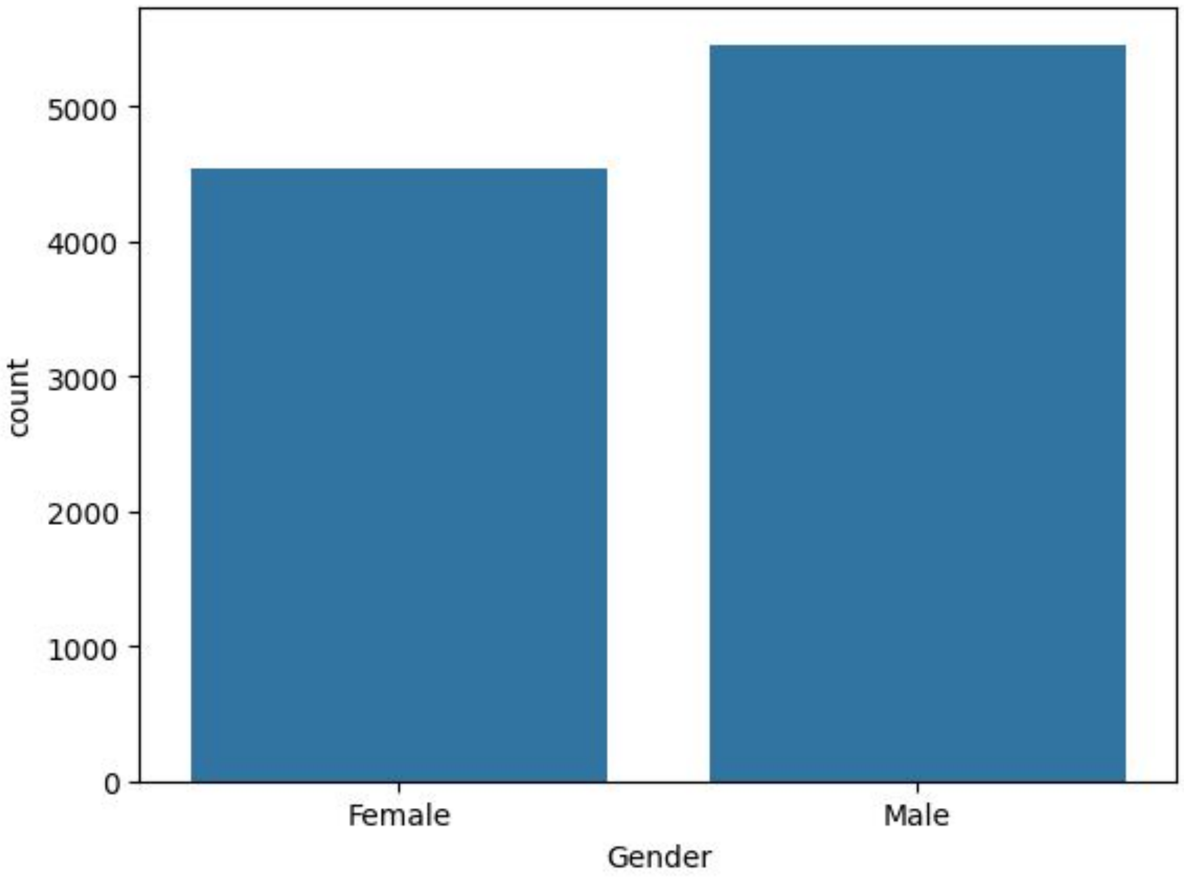


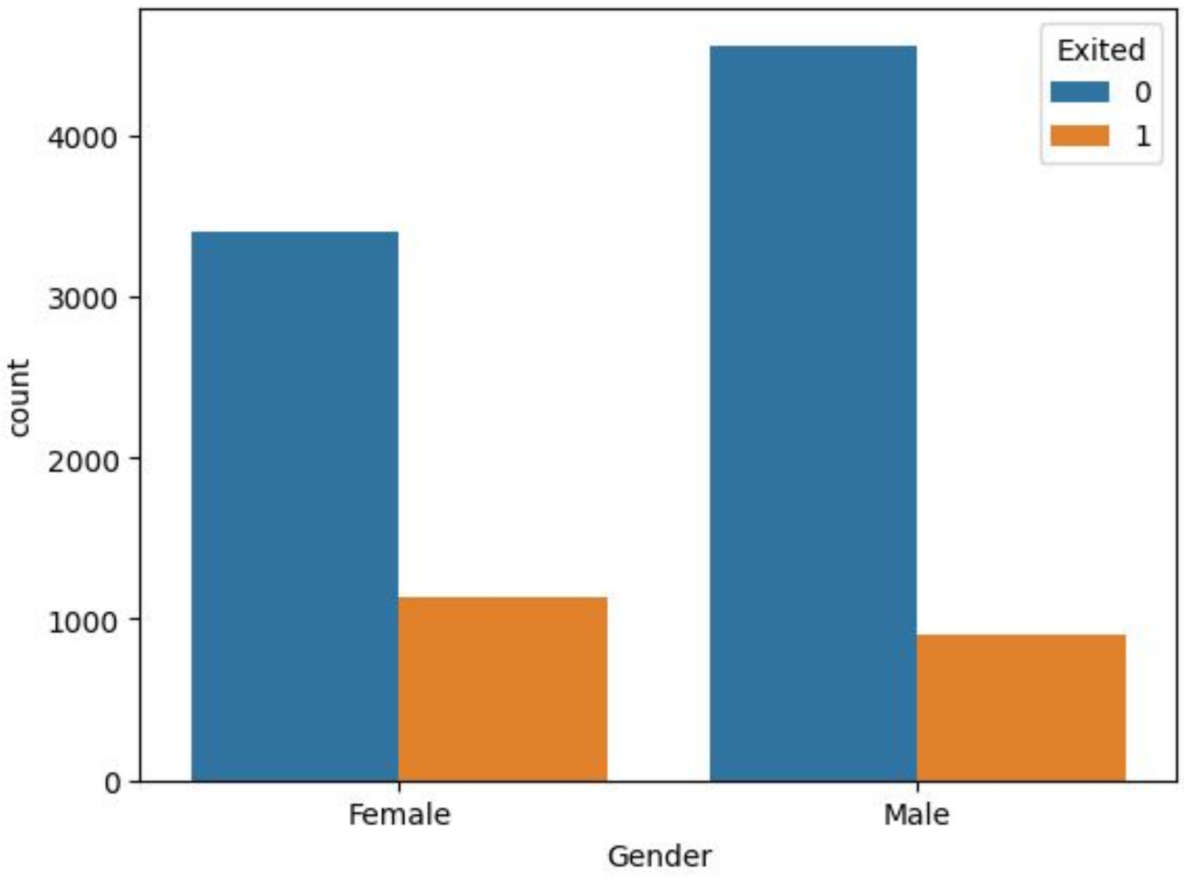
Only about a quarter of all the people churned so the data is imbalanced.

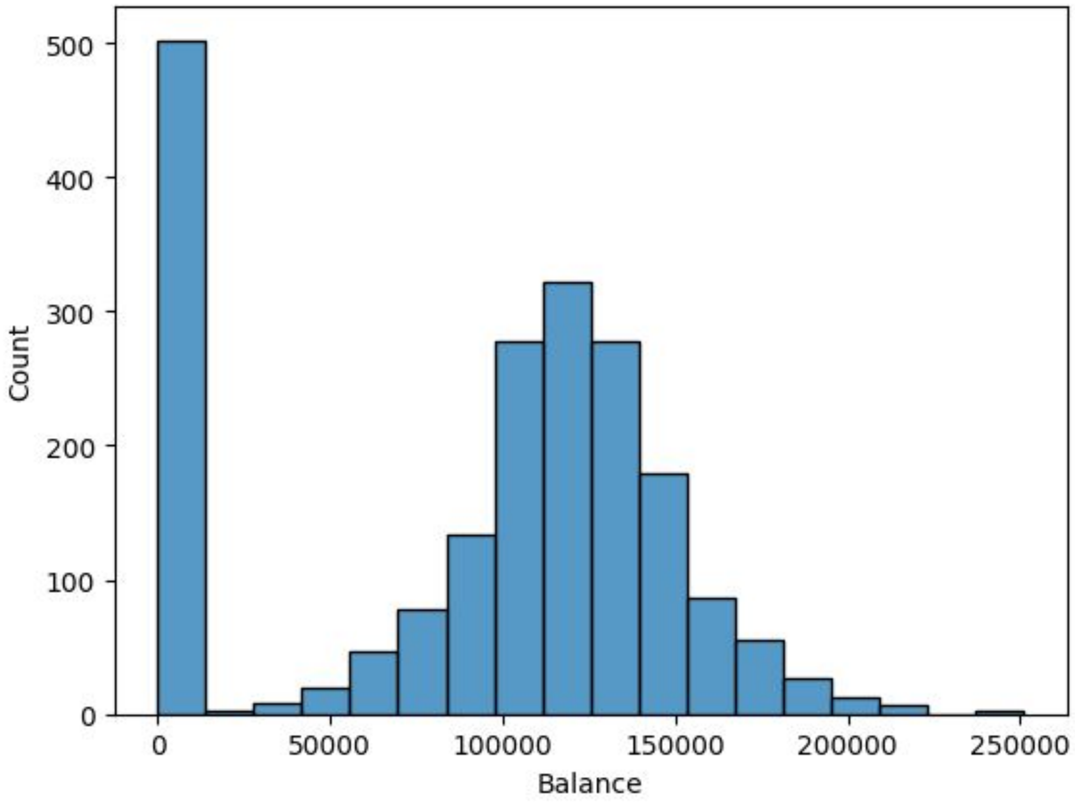


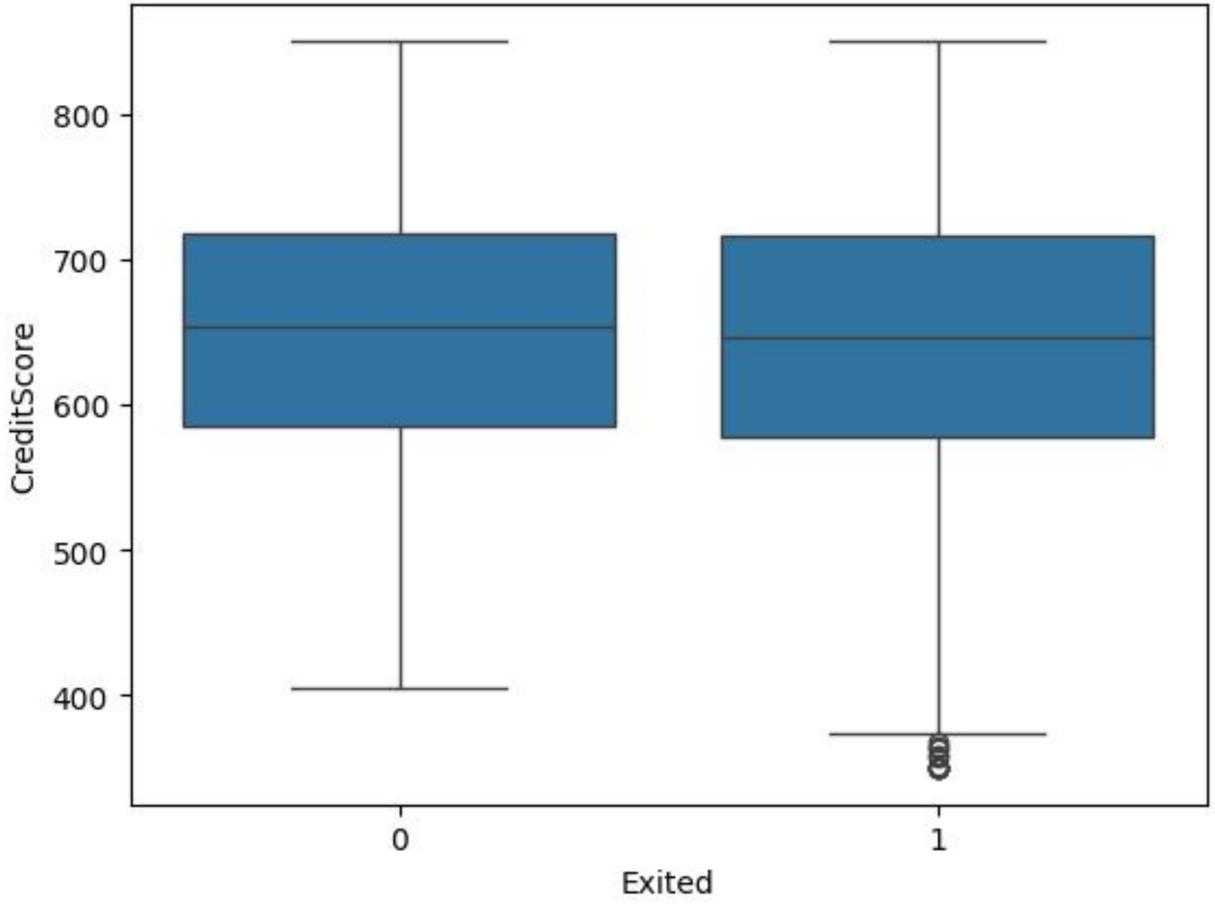
Most customers are from France but Germany has the highest churn percentage.

When analyzing the gender we see that there are a bit more males than females, but females tend to churn more.









Largest number of people who churn have a 0 balance

Credit scores are very similar but the ones that don't churn but have a slightly lower median and lower outliers as well.

Methodology

Model Selection

- Basic Models: Logistic Regression, Kneighbors, SVC, Gaussian NB
- Advanced Models: Random Forest, XGBoost, Gaussian Mixture
- Deep Learning Models: CNN

Evaluation Metrics:

- Accuracy, Precision, Recall, F1-Score, AUC.

Predictors and Target Variables

Target Variable:

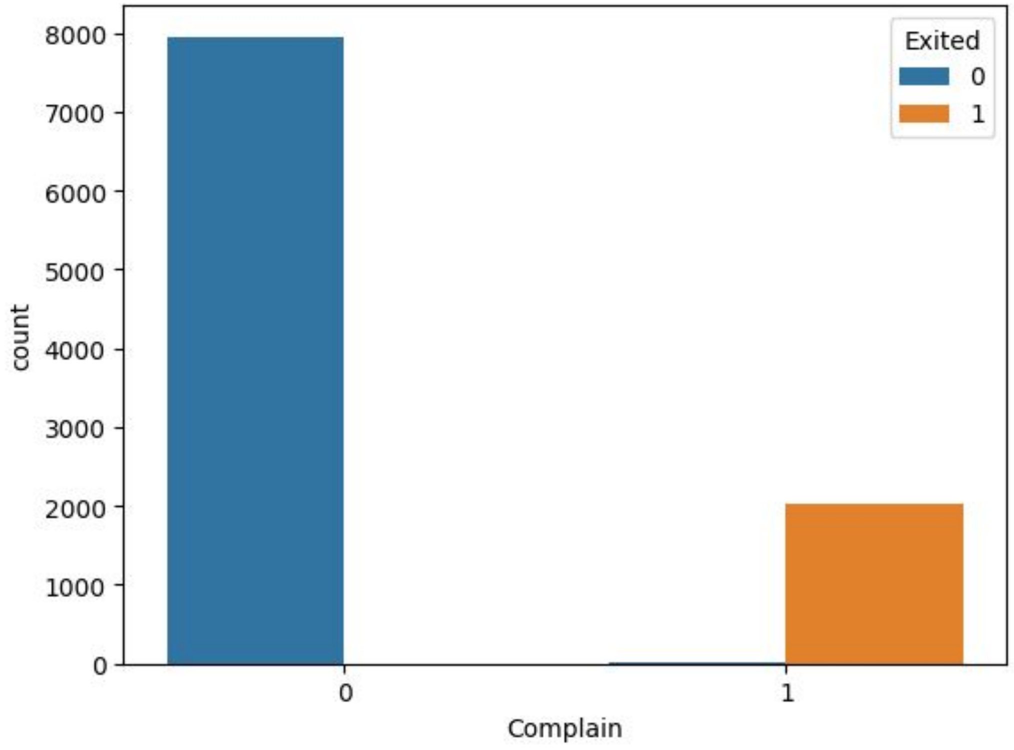
- **Exited:** Binary variable indicating if a customer has churned (1) or not (0).

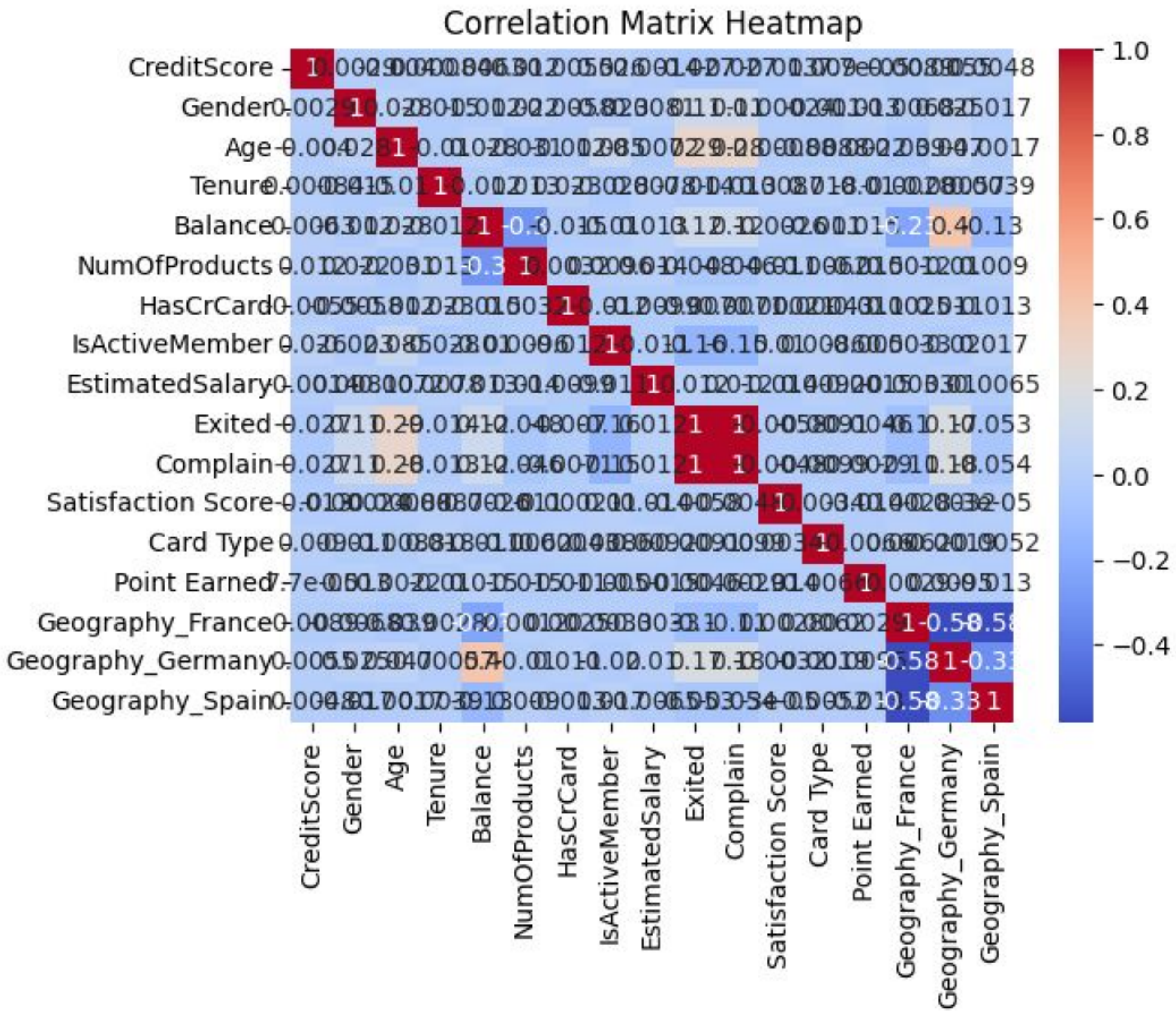
Predictor Variables:

- **Demographics:** Age, gender, geography.
- **Account Information:** Balance, number of products, tenure.
- **Customer Engagement:** Participation in loyalty programs
- **Historical Data:** Previous complaints

Correlations

Complain is correlated to exited.





- Removed columns not using including complaint since is too highly correlated for any of the other variables to matter.
- Converted categorical variables to numeric. Used one hot encoding for geography since its not ordinal.
- Split into testing and training

```
#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)
```

Prepared 3 version of the data set to test which one performs best.

Regular version, scaled version, balanced version.

```
#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)
```

Created a function that tries it on a list of models.

Initial results of model on all 3 versions of the data.

Balanced data had highest scores on f1 score and recall which is what matters most for churn prediction.

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

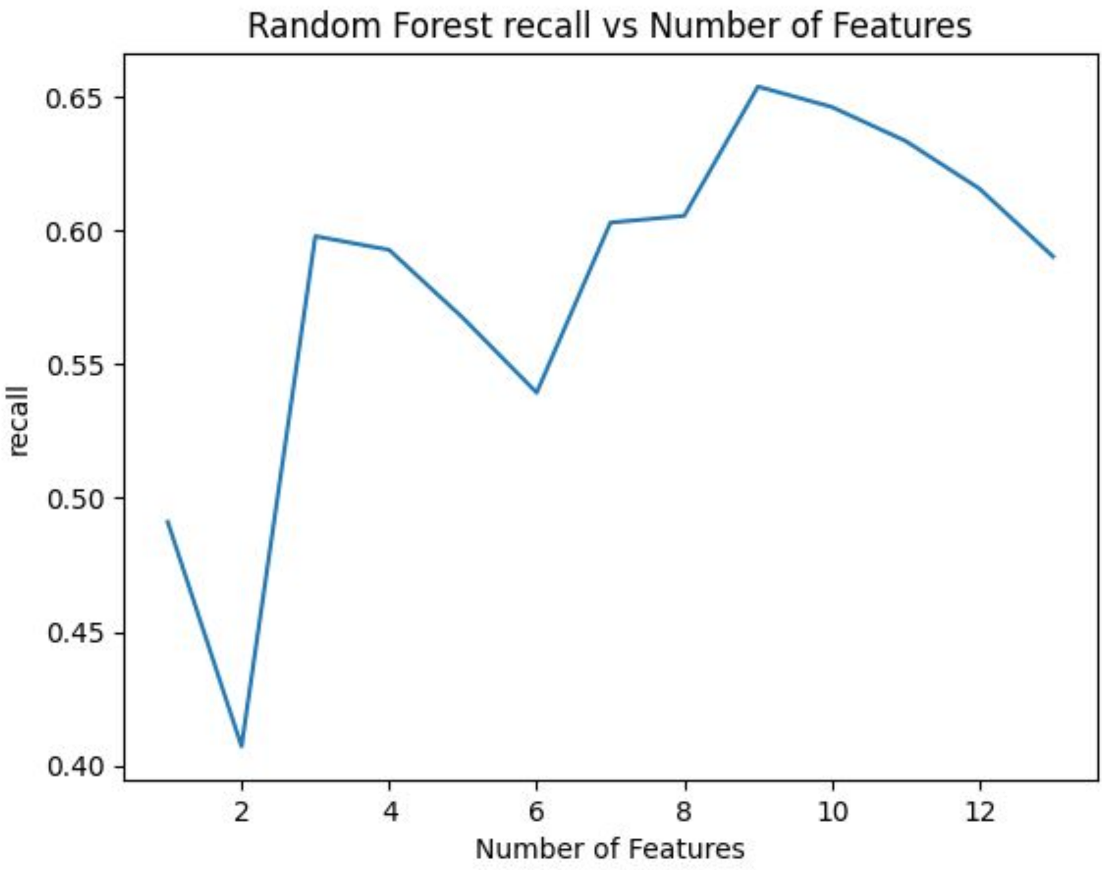
Weighted Data

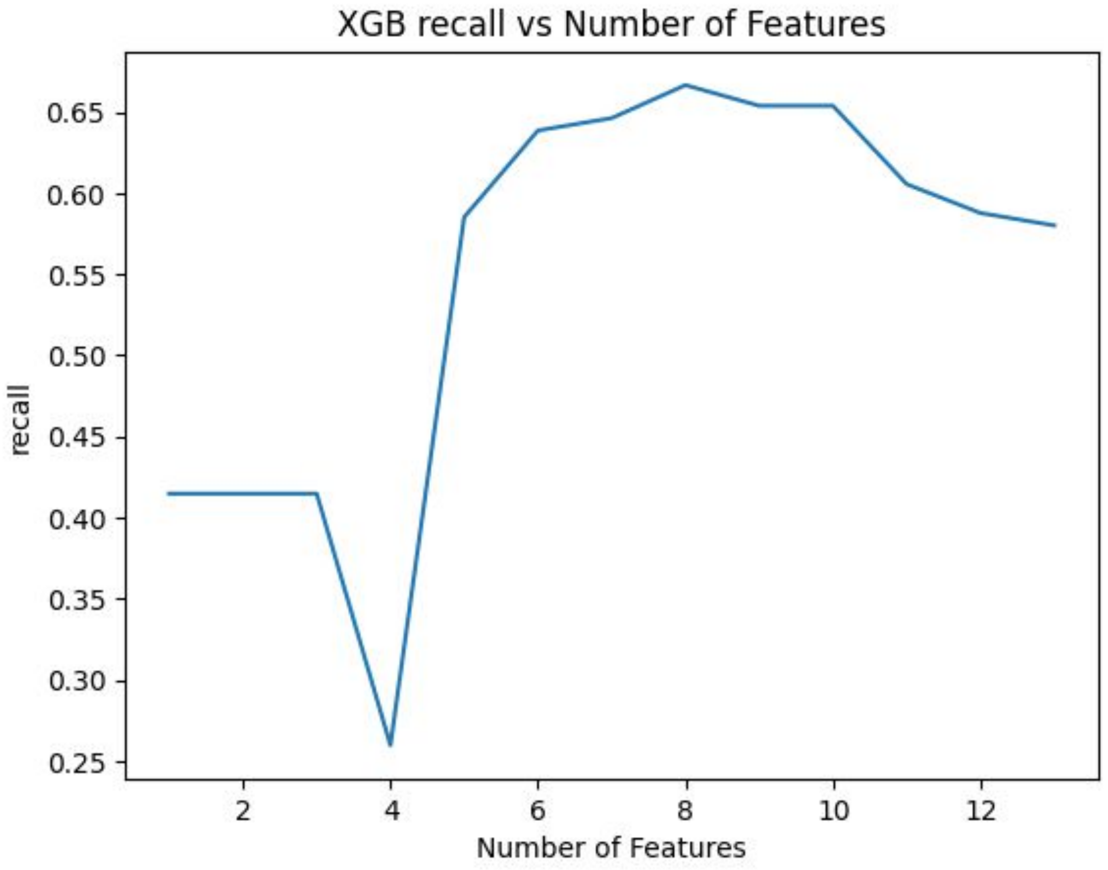
model	cm	f1	recall	precision	accuracy	auc
GaussianNB	[[1073 534] [117 276]]	0.458853	0.702290	0.340741	0.674500	0.684997
SVC	[[606 1001] [96 297]]	0.351271	0.755725	0.228814	0.451500	0.566413
KNeighborsClassifier	[[983 624] [230 163]]	0.276271	0.414758	0.207116	0.573000	0.513229
LogisticRegression	[[1013 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

Additional data engineering for the 3 highest performing models.

Tried doing pca and rfe to select best features. Used line plot to visualize best number of features for each model.

Used 8 for xgb model and 9 for the random forest.





Looking at recall the svc model on the regular dataset still performs the best. With the optimized xgb second.

If we look at f1 score the optimized xgb performs best with svc on pca data second.
Will go with the optimized xgb model as that one is one of top two on both merics looking at and performs much better than svc on other metrics as accuracy and precision

	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

Additional models and fine tuning

- Tried an advanced cnn model but the results weren't close to the top models here

- Used Grid search cv to get best parameters but it did not increase the scores.

```
# 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)
```

Top Features

Looking at the top features that indicate 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.

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

```
Selected Features: Index(['Age', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember',
       'Geography_France', 'Geography_Germany', 'Geography_Spain'],
      dtype='object')
```

Next Steps

- Future work could focus on enhancing model transparency, perhaps through the integration of explainable AI techniques, to facilitate better understanding
- Might try to incorporate findings 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.

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.

Appendix

Python Code

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

```
print(churn.shape)
```

```
(10000, 18)
```

```
churn =churn.rename(columns={'Satisfaction Score':'SatisfactionScore','Card Type':'CardType','Point Earned':'PointEarned'})
```

```
print(churn.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   RowNumber          10000 non-null  int64  
 1   CustomerId         10000 non-null  int64  
 2   Surname            10000 non-null  object 
 3   CreditScore        10000 non-null  int64  
 4   Geography          10000 non-null  object 
 5   Gender             10000 non-null  object 
 6   Age                10000 non-null  int64  
 7   Tenure             10000 non-null  int64  
 8   Balance            10000 non-null  float64
 9   NumOfProducts      10000 non-null  int64  
 10  HasCrCard          10000 non-null  int64  
 11  IsActiveMember     10000 non-null  int64  
 12  EstimatedSalary    10000 non-null  float64
 13  Exited             10000 non-null  int64  
 14  Complain           10000 non-null  int64  
 15  SatisfactionScore  10000 non-null  int64  
 16  CardType           10000 non-null  object 
 17  PointEarned        10000 non-null  int64  
dtypes: float64(2), int64(12), object(4)
memory usage: 1.4+ MB
None
```

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

```
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
Complain              0
Satisfaction Score    0
Card Type             0
Point Earned          0
dtype: int64
```

```python
#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
```

```python
#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)
```