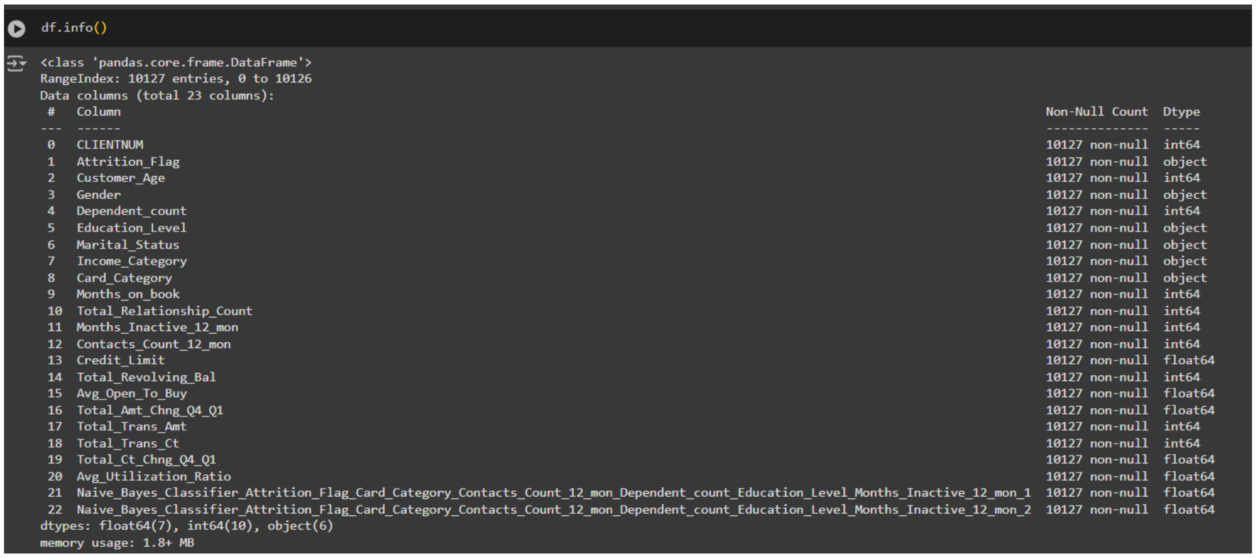
Question 1

Part1

This project uses the **Credit Card Customer Dataset** available on Kaggle (login required). The dataset includes details about credit card holders, such as demographics, account information, and card usage patterns. This dataset provides an excellent basis for understanding customer behavior and exploring factors that might influence their retention.

This dataset will help us investigate the factors associated with customer attrition in the credit card industry.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 23 columns):
 #   Column                                                                                                                               Non-Null Count  Dtype  
---  ------                                                                                                                               --------------  -----  
 0   CLIENTNUM                                                                                                                            10127 non-null  int64  
 1   Attrition_Flag                                                                                                                       10127 non-null  object 
 2   Customer_Age                                                                                                                         10127 non-null  int64  
 3   Gender                                                                                                                               10127 non-null  object 
 4   Dependent_count                                                                                                                      10127 non-null  int64  
 5   Education_Level                                                                                                                      10127 non-null  object 
 6   Marital_Status                                                                                                                       10127 non-null  object 
 7   Income_Category                                                                                                                      10127 non-null  object 
 8   Card_Category                                                                                                                        10127 non-null  object 
 9   Months_on_book                                                                                                                       10127 non-null  int64  
 10  Total_Relationship_Count                                                                                                             10127 non-null  int64  
 11  Months_Inactive_12_mon                                                                                                               10127 non-null  int64  
 12  Contacts_Count_12_mon                                                                                                                10127 non-null  int64  
 13  Credit_Limit                                                                                                                         10127 non-null  float64
 14  Total_Revolving_Bal                                                                                                                  10127 non-null  int64  
 15  Avg_Open_To_Buy                                                                                                                      10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1                                                                                                                 10127 non-null  float64
 17  Total_Trans_Amt                                                                                                                      10127 non-null  int64  
 18  Total_Trans_Ct                                                                                                                       10127 non-null  int64  
 19  Total_Ct_Chng_Q4_Q1                                                                                                                  10127 non-null  float64
 20  Avg_Utilization_Ratio                                                                                                                10127 non-null  float64
 21  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1  10127 non-null  float64
 22  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2  10127 non-null  float64
dtypes: float64(7), int64(10), object(6)
memory usage: 1.8+ MB
```

Figure 1: Features

The dimensions of this dataset is (10127,23) . This dataset has 1027 samples and 23 features.

Part2 - Data distribution

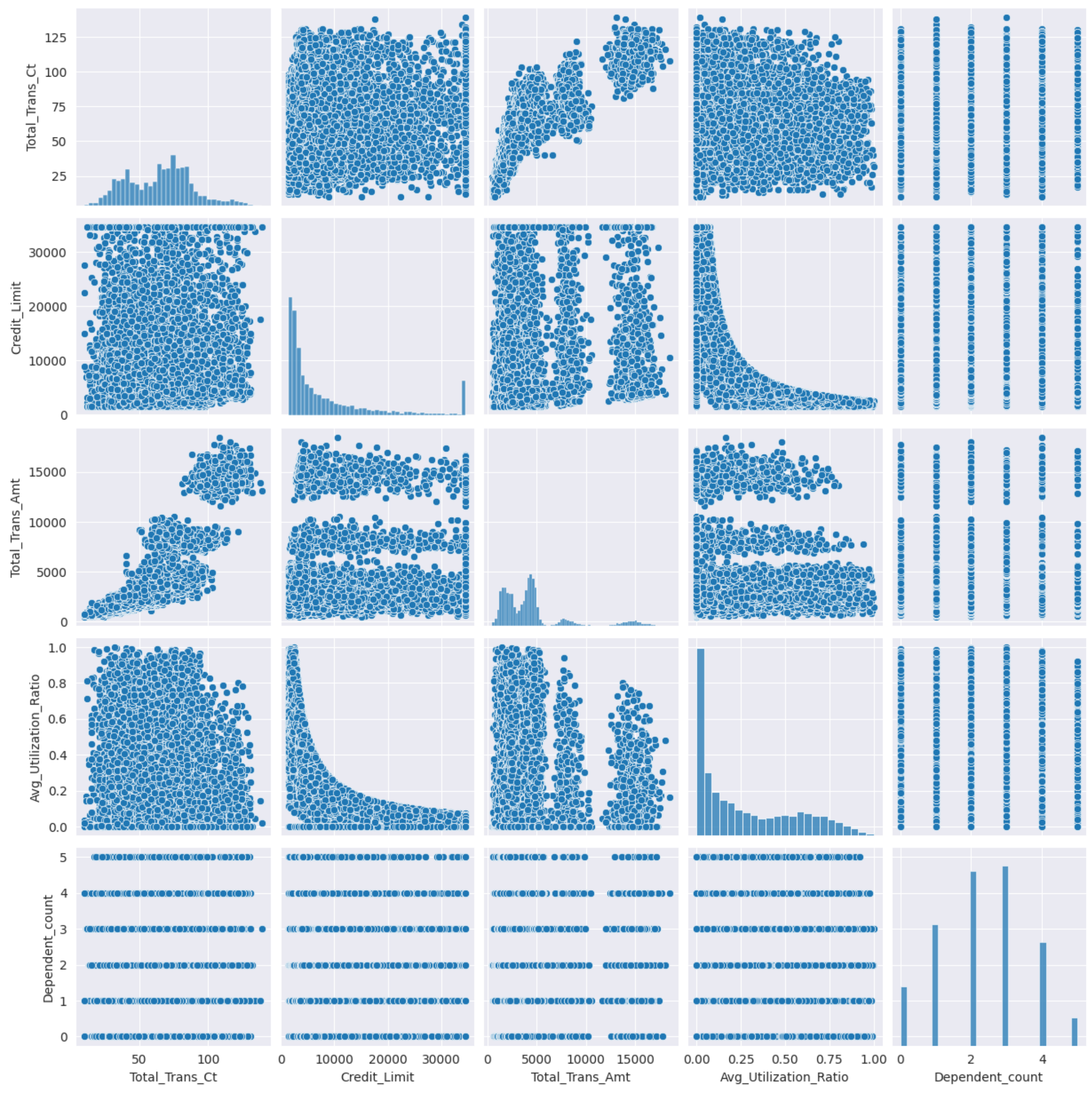
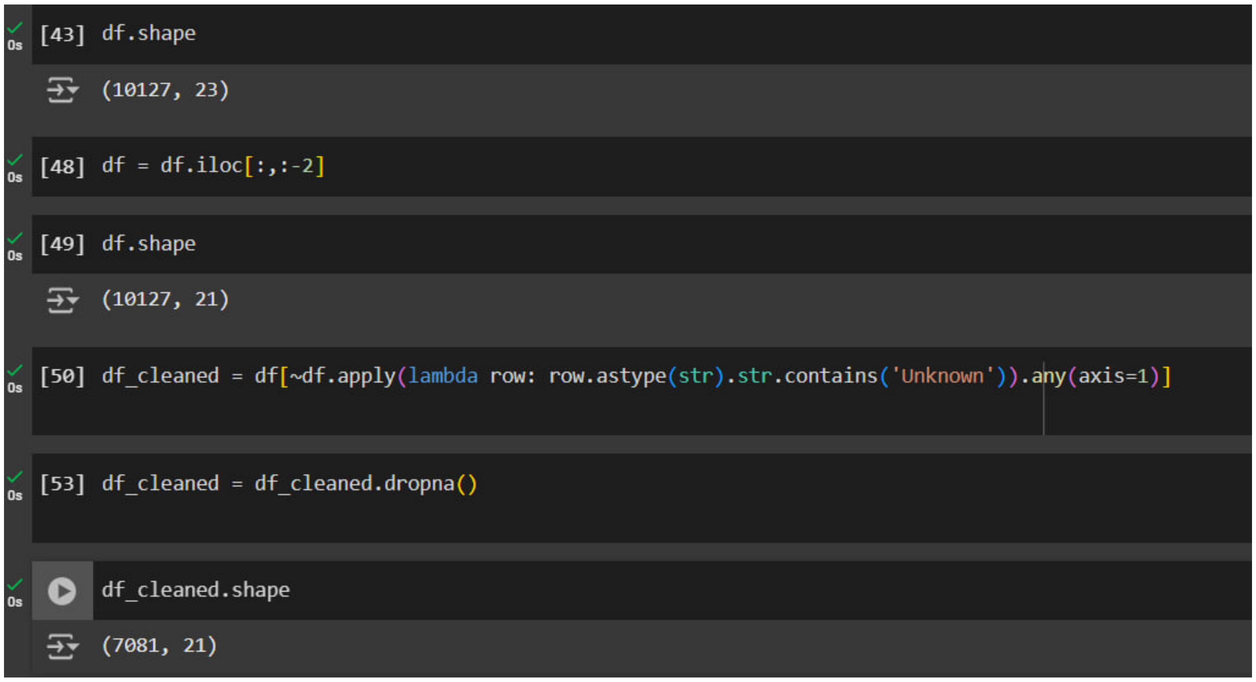


Figure 2: Data distribution

Unknown cells

I came across some unknown cells in this dataset. At first, I decided to remove them, but then I saw that the number of samples was greatly reduced. Then I decided to use them in my modeling because this data might have a meaningful pattern. There are other methods to replace these values (like KNN).



```
[43] df.shape
(10127, 23)

[48] df = df.iloc[:,:-2]

[49] df.shape
(10127, 21)

[50] df_cleaned = df[~df.apply(lambda row: row.astype(str).str.contains('Unknown')).any(axis=1)]

[53] df_cleaned = df_cleaned.dropna()

df_cleaned.shape
(7081, 21)
```

Figure 3: Unknown cells

Encoder

A `CustomEncoder` has been designed for this dataset, and we will examine its features below. The `CustomEncoder` class encodes any dataset by:

1. **Identifying Features:**
   - Detects **binary features** (with 2 unique values).
   - Detects **multi-class features** (with more than 2 unique values).
2. **Encoding:**
   - **Binary Features:** Encodes categories as `0` and `1`, with `'Unknown'` set to `-1`.
   - **Multi-Class Features:** Applies ordinal encoding (e.g., `0, 1, 2`), with `'Unknown'` encoded as `-1`.

3. **Transforming New Data:**

   - Ensures consistency by encoding unseen categories as `-1` for both binary and multi-class features.

This ensures all categorical data is converted into numerical format, suitable for machine learning models.

correlation

Correlation

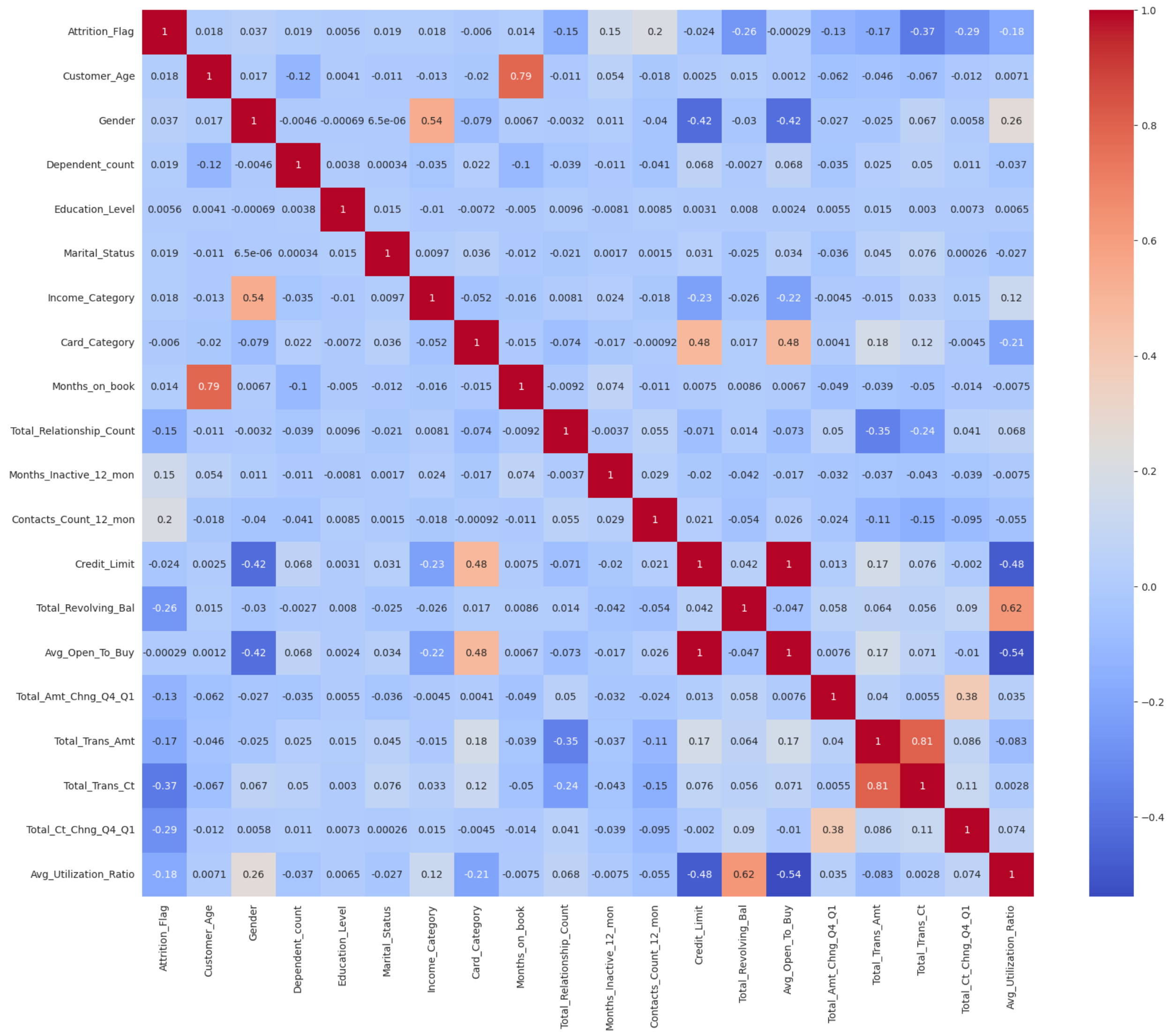
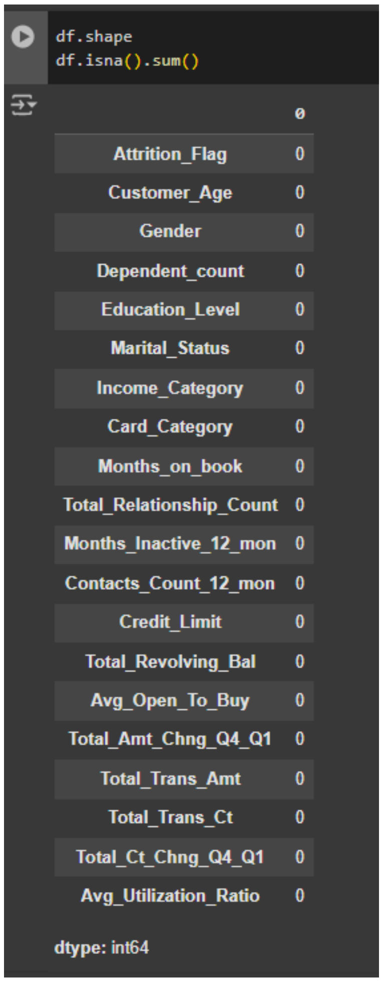


Figure 4: correlation

The figure above shows the correlation matrix of the dataset. As you can see, features like `Credit_Limit` and `Avg_Open_To_Buy` or `Total_Revolving_Bal` and `Avg_Utilization_Ratio` provide overlapping information, which can affect regression models.

We can use Principal Component Analysis (PCA) to reduce the dimensionality of the dataset. PCA helps in identifying the most significant components of the data, allowing us to retain the maximum variance while reducing redundancy caused by highly correlated features.

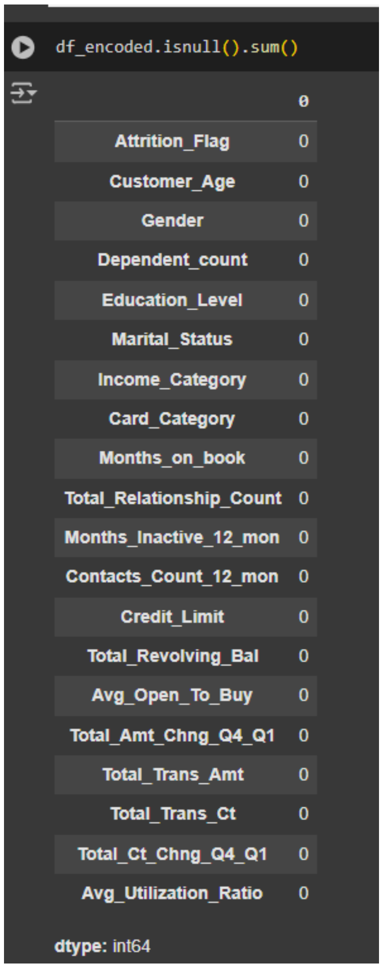
Part4 - NaN



```
df.shape
df.isna().sum()
```

```
                          0
Attrition_Flag            0
Customer_Age              0
Gender                    0
Dependent_count           0
Education_Level           0
Marital_Status            0
Income_Category           0
Card_Category             0
Months_on_book            0
Total_Relationship_Count  0
Months_Inactive_12_mon    0
Contacts_Count_12_mon     0
Credit_Limit              0
Total_Revolving_Bal       0
Avg_Open_To_Buy           0
Total_Amt_Chng_Q4_Q1      0
Total_Trans_Amt           0
Total_Trans_Ct            0
Total_Ct_Chng_Q4_Q1       0
Avg_Utilization_Ratio     0

dtype: int64
```

Figure 5



```
df_encoded.isnull().sum()
```

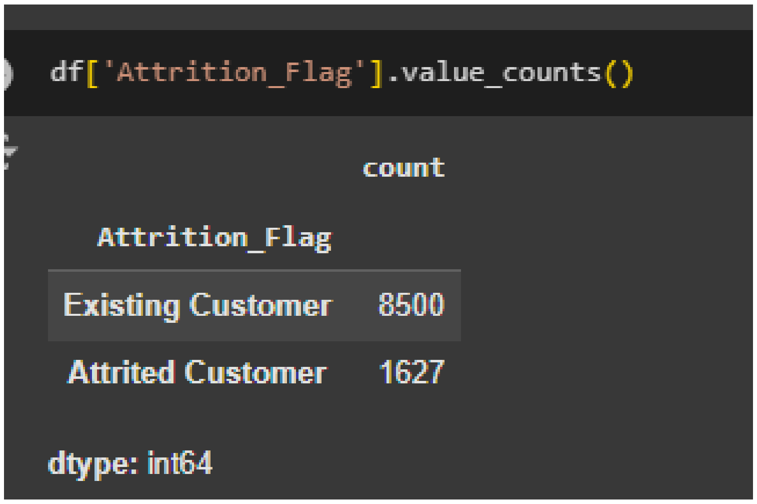
```
                          0
Attrition_Flag            0
Customer_Age              0
Gender                    0
Dependent_count           0
Education_Level           0
Marital_Status            0
Income_Category           0
Card_Category             0
Months_on_book            0
Total_Relationship_Count  0
Months_Inactive_12_mon    0
Contacts_Count_12_mon     0
Credit_Limit              0
Total_Revolving_Bal       0
Avg_Open_To_Buy           0
Total_Amt_Chng_Q4_Q1      0
Total_Trans_Amt           0
Total_Trans_Ct            0
Total_Ct_Chng_Q4_Q1       0
Avg_Utilization_Ratio     0

dtype: int64
```

Figure 6

As you can see, there is no NaN data either before or after encoding.

Part5 - Atrrition Flag

This feature indicates whether the customer has churned or not. The figure below shows that we have 8500 samples from class Existing Customer and 1627 samples from class Attrited Customer.



```
df['Attrition_Flag'].value_counts()

                     count
   Attrition_Flag
Existing Customer     8500
Attrited Customer     1627

dtype: int64
```

Figure 7: The names of the classes in this feature and their number

The chart below shows the distribution of data in this feature in the form of a pie plot.

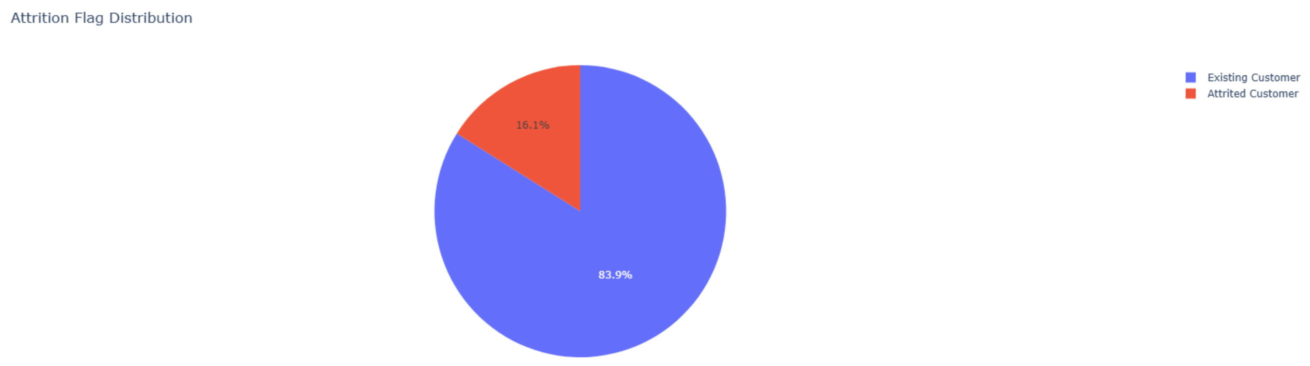


Figure 8: distribution of data in Atrrition Flag in the form of a pie plot

Part6 - The problem of imbalance in the data set

Considering the data distribution, we can say that this dataset is unbalanced. One of the common challenges in machine learning projects is dealing with imbalanced data. Data in which the distribution of samples across classes is not uniform and the number of samples in one or more classes is much lower than in the others. This can reduce the accuracy of machine learning models in predicting sparsely populated classes. In these cases, we should not expect classification algorithms to learn correctly, as these algorithms usually prefer classes with the majority label. In these cases, the classification evaluation criterion must also change and normal criteria such as accuracy cannot be used. If an algorithm classified all new samples as the densely populated class, the accuracy would be approximately 99.99, but in this type of problem, we cannot rely on this accuracy criterion. In these situations, different data balancing methods or specific algorithms must be used.

SMOTE

SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class, based on those that already exist. It works randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.

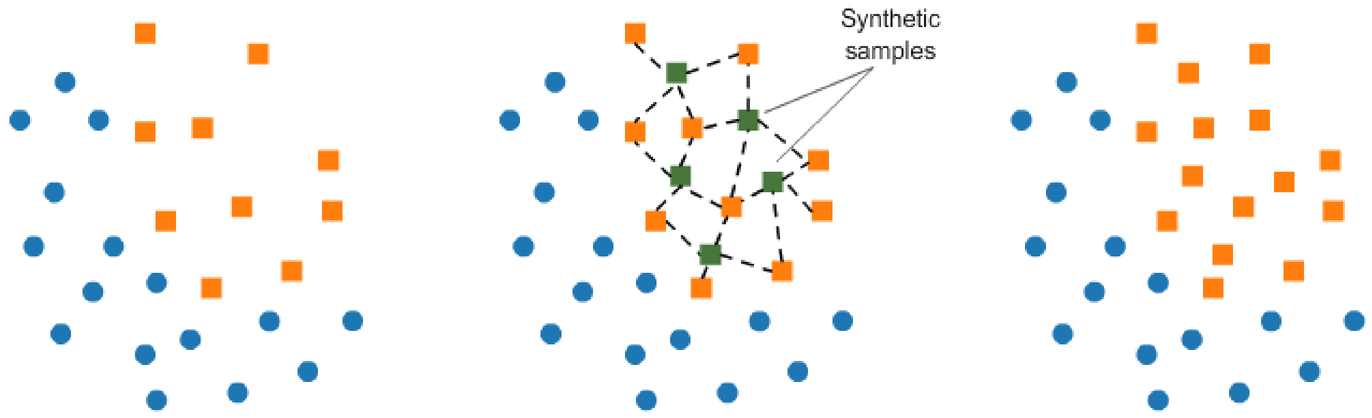


Figure 9: SMOTE

Let's examine this method using the image below.

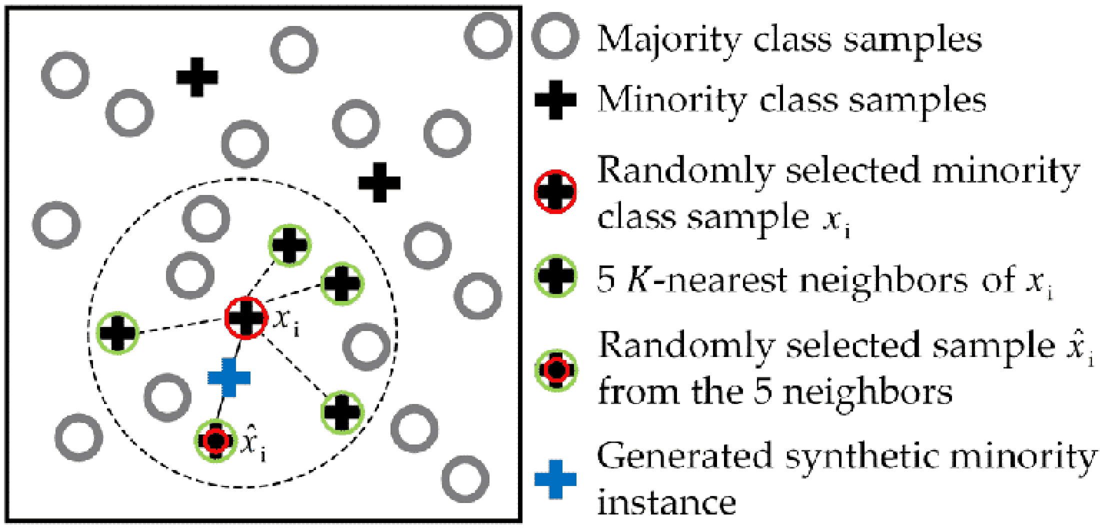

x_i
$\hat{x}_i$
Majority class samples
Minority class samples
Randomly selected minority class sample x_i
5 K-nearest neighbors of x_i
Randomly selected sample $\hat{x}_i$ from the 5 neighbors
Generated synthetic minority instance

Figure 10: SMOTE

1. **Select a random sample from the minority class:** A random sample x_i (highlighted in red in the figure) is selected from the minority class.
2. **Find the K-nearest neighbors:** The K-nearest neighbors of x_i (highlighted with green circles) are identified based on a distance metric, such as the Euclidean distance.
3. **Randomly choose one neighbor:** One of the K-nearest neighbors $\hat{x}_i$ (highlighted with a red circle and green border) is selected randomly.
4. **Generate a synthetic sample:** A new synthetic sample (represented by the blue cross in the figure) is created along the line segment joining x_i and $\hat{x}_i$. This is done using the following formula:

   $$x_{\text{synthetic}} = x_i + \lambda \cdot (\hat{x}_i - x_i)$$

   where λ is a random value in the range $[0, 1]$.

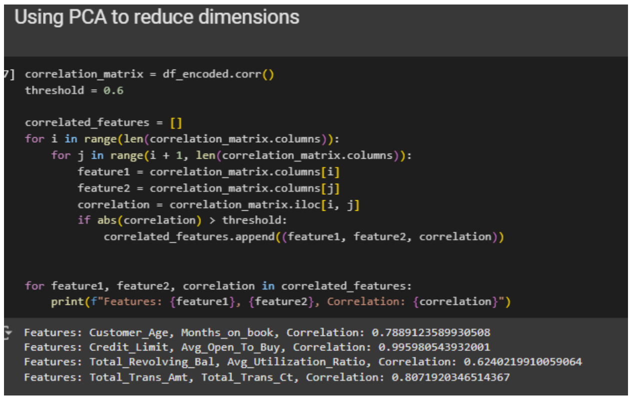
Balancing After Data Partitioning

This method is applied **after data partitioning**. By performing balancing only on the training set, information leakage to the test set is prevented. The test set remains a more realistic representation of the original data. This method ensures that the model's performance is evaluated on the real data (and not the balanced data).

The test set still represents the original distribution of the data, which is important for real-world evaluation. The test set should be reflective of the original data to test the model's performance in real-world conditions.

Dimensionality Reduction Using PCA

In this part, I used **Principal Component Analysis (PCA)** to reduce the dimensions of the problem. To achieve this, I examined the features two by two from the correlation matrix. If the dependence value between two features was greater than 0.6, PCA was applied to them.

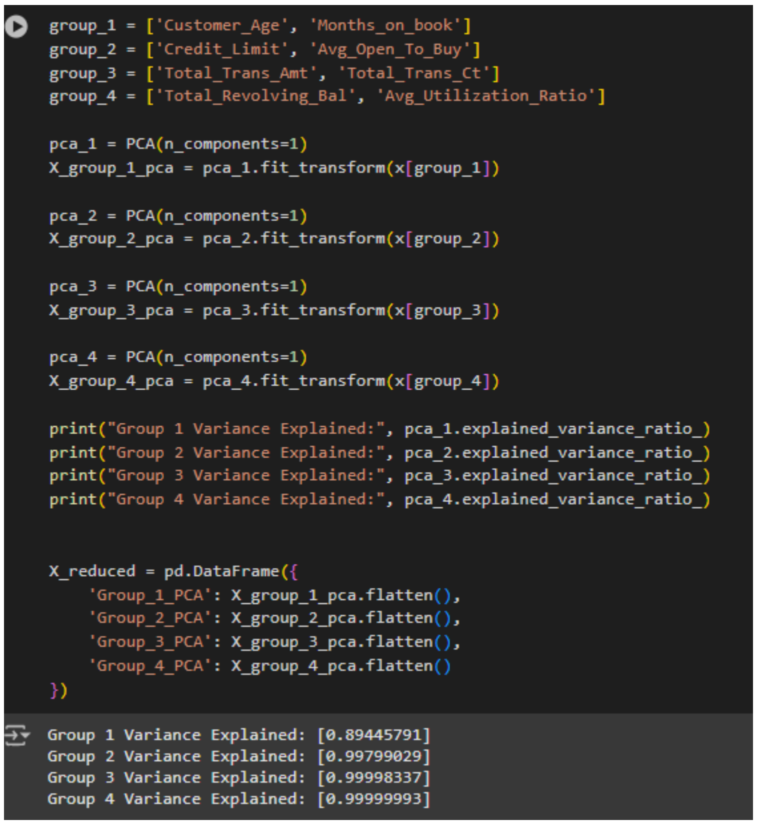


Using PCA to reduce dimensions

```
7] correlation_matrix = df_encoded.corr()
   threshold = 0.6

   correlated_features = []
   for i in range(len(correlation_matrix.columns)):
       for j in range(i + 1, len(correlation_matrix.columns)):
           feature1 = correlation_matrix.columns[i]
           feature2 = correlation_matrix.columns[j]
           correlation = correlation_matrix.iloc[i, j]
           if abs(correlation) > threshold:
               correlated_features.append((feature1, feature2, correlation))


   for feature1, feature2, correlation in correlated_features:
       print(f"Features: {feature1}, {feature2}, Correlation: {correlation}")
```

```
Features: Customer_Age, Months_on_book, Correlation: 0.7889123589930508
Features: Credit_Limit, Avg_Open_To_Buy, Correlation: 0.995980543932001
Features: Total_Revolving_Bal, Avg_Utilization_Ratio, Correlation: 0.6240219910059064
Features: Total_Trans_Amt, Total_Trans_Ct, Correlation: 0.8071920346514367
```

Figure 11: Applying PCA



```
group_1 = ['Customer_Age', 'Months_on_book']
group_2 = ['Credit_Limit', 'Avg_Open_To_Buy']
group_3 = ['Total_Trans_Amt', 'Total_Trans_Ct']
group_4 = ['Total_Revolving_Bal', 'Avg_Utilization_Ratio']

pca_1 = PCA(n_components=1)
X_group_1_pca = pca_1.fit_transform(x[group_1])

pca_2 = PCA(n_components=1)
X_group_2_pca = pca_2.fit_transform(x[group_2])

pca_3 = PCA(n_components=1)
X_group_3_pca = pca_3.fit_transform(x[group_3])

pca_4 = PCA(n_components=1)
X_group_4_pca = pca_4.fit_transform(x[group_4])

print("Group 1 Variance Explained:", pca_1.explained_variance_ratio_)
print("Group 2 Variance Explained:", pca_2.explained_variance_ratio_)
print("Group 3 Variance Explained:", pca_3.explained_variance_ratio_)
print("Group 4 Variance Explained:", pca_4.explained_variance_ratio_)


X_reduced = pd.DataFrame({
    'Group_1_PCA': X_group_1_pca.flatten(),
    'Group_2_PCA': X_group_2_pca.flatten(),
    'Group_3_PCA': X_group_3_pca.flatten(),
    'Group_4_PCA': X_group_4_pca.flatten()
})
```

```
Group 1 Variance Explained: [0.89445791]
Group 2 Variance Explained: [0.99799029]
Group 3 Variance Explained: [0.99998337]
Group 4 Variance Explained: [0.99999993]
```

Figure 12: Applying PCA

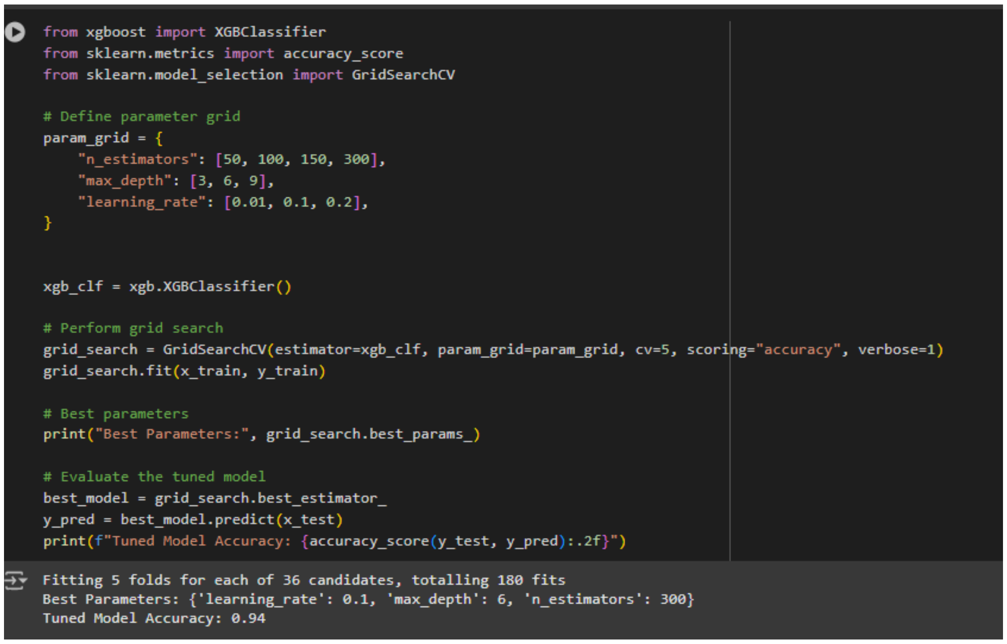
XGBoost Classifier

At this stage we start training the model. The model used is XG Boost. XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm based on the gradient boosting framework. It is designed to enhance model performance and efficiency, making it highly effective for classification tasks. The algorithm works by combining multiple weak learners, typically decision trees, into a strong predictive model.

Each tree is added iteratively, focusing on correcting the errors made by the previous trees. This process is guided by gradient descent, which minimizes a specified loss function and improves the model's predictions. XGBoost incorporates regularization techniques, such as L1 and L2 penalties, to prevent overfitting and enhance generalization.

A key feature of XGBoost is its ability to handle large and complex datasets efficiently. The algorithm supports parallel processing and can handle missing data inherently, making it both robust and versatile. Additionally, XGBoost allows customization through various hyperparameters, enabling practitioners to fine-tune the model for specific problems.

Due to its speed, flexibility, and superior performance, XGBoost has become a popular choice for machine learning tasks in both academic and industrial applications. It is particularly well-suited for structured/tabular data and has been widely adopted in machine learning competitions.



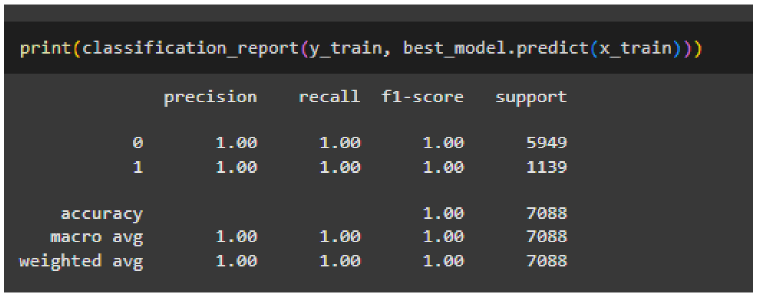
```python
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    "n_estimators": [50, 100, 150, 300],
    "max_depth": [3, 6, 9],
    "learning_rate": [0.01, 0.1, 0.2],
}


xgb_clf = xgb.XGBClassifier()

# Perform grid search
grid_search = GridSearchCV(estimator=xgb_clf, param_grid=param_grid, cv=5, scoring="accuracy", verbose=1)
grid_search.fit(x_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Evaluate the tuned model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
print(f"Tuned Model Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300}
Tuned Model Accuracy: 0.94
```

Figure 13: Training XGBoost model

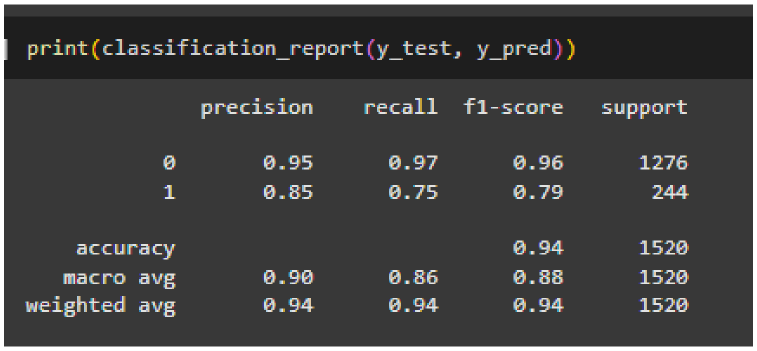
Hyperparameter Tuning with GridSearchCV

To optimize the performance of the XGBoost classifier, a grid search was conducted to identify the best combination of hyperparameters. This grid includes a range of values for each hyperparameter to comprehensively explore different configurations. It was implemented using the `GridSearchCV` method from `scikit-learn`. The search was conducted with a 5-fold cross-validation (`cv=5`) to ensure robust evaluation of each parameter combination. The scoring metric used for model evaluation was accuracy.



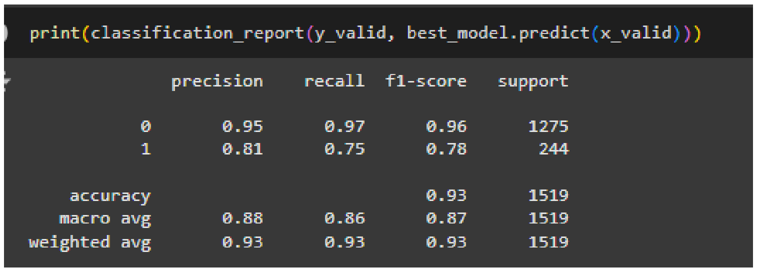
```
print(classification_report(y_train, best_model.predict(x_train)))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5949
           1       1.00      1.00      1.00      1139

    accuracy                           1.00      7088
   macro avg       1.00      1.00      1.00      7088
weighted avg       1.00      1.00      1.00      7088
```

Figure 14: Model classification report for training data



```
print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1276
           1       0.85      0.75      0.79       244

    accuracy                           0.94      1520
   macro avg       0.90      0.86      0.88      1520
weighted avg       0.94      0.94      0.94      1520
```

Figure 15: Model classification report for testing data



```
print(classification_report(y_valid, best_model.predict(x_valid)))

              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1275
           1       0.81      0.75      0.78       244

    accuracy                           0.93      1519
   macro avg       0.88      0.86      0.87      1519
weighted avg       0.93      0.93      0.93      1519
```

Figure 16: Model classification report for validation data

The classification report for the training data indicates that the model achieves near-perfect performance, as evidenced by high precision, recall, and F1-scores. However, for the testing data, a slight drop in performance metrics is observed, which is expected due to unseen data. The validation data report further confirms the consistency of the model across different datasets, demonstrating robust generalization capabilities.

Confusion Matrices and Model Performance Analysis

Confusion matrices for the training, testing, and validation datasets provide a comprehensive understanding of the model's performance. These matrices highlight the number of true positive, true negative, false positive, and false negative predictions, along with corresponding accuracy, precision, and recall metrics.

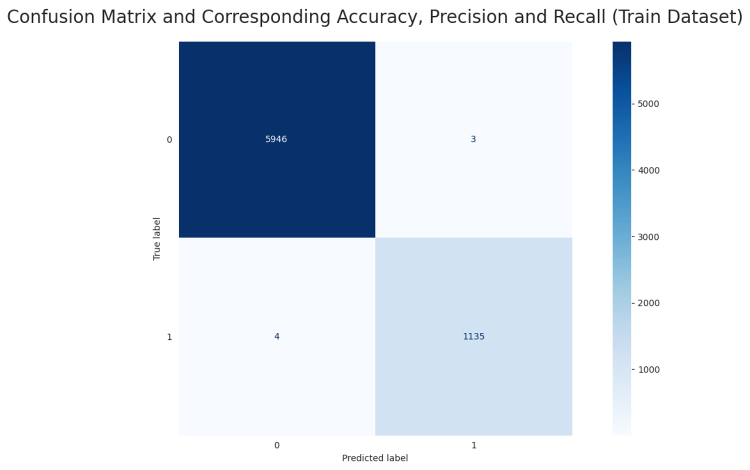


Figure 17: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Train Dataset)

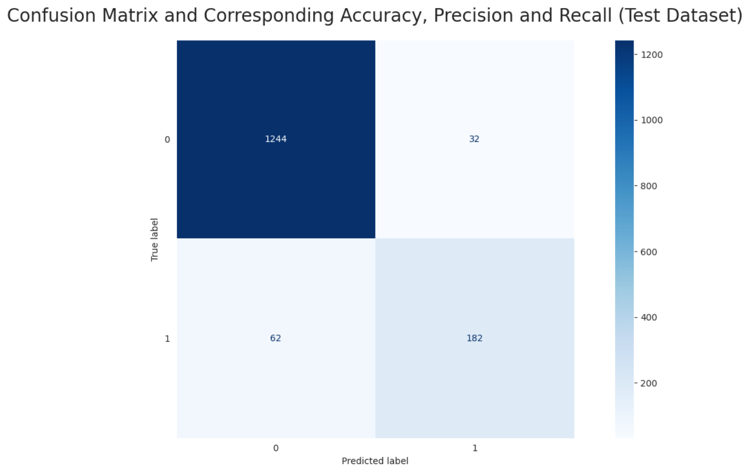


Figure 18: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Test Dataset)

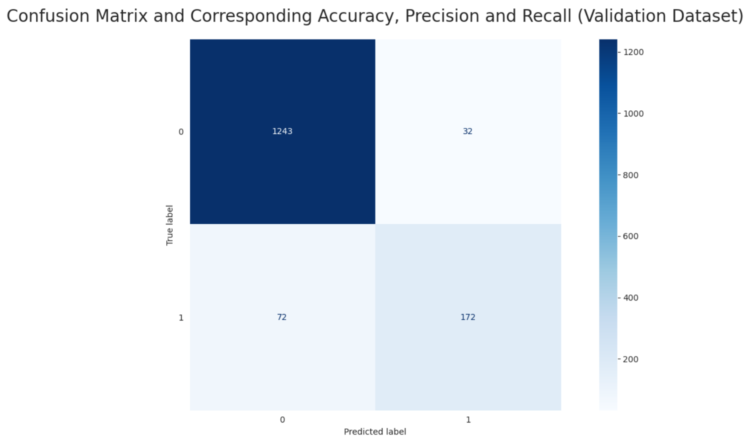


Figure 19: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Validation Dataset)

Training Dataset The confusion matrix for the training dataset indicates excellent performance, with nearly all predictions being correct. This suggests that the model has effectively learned the patterns in the training data, achieving high precision, recall, and F1-scores.

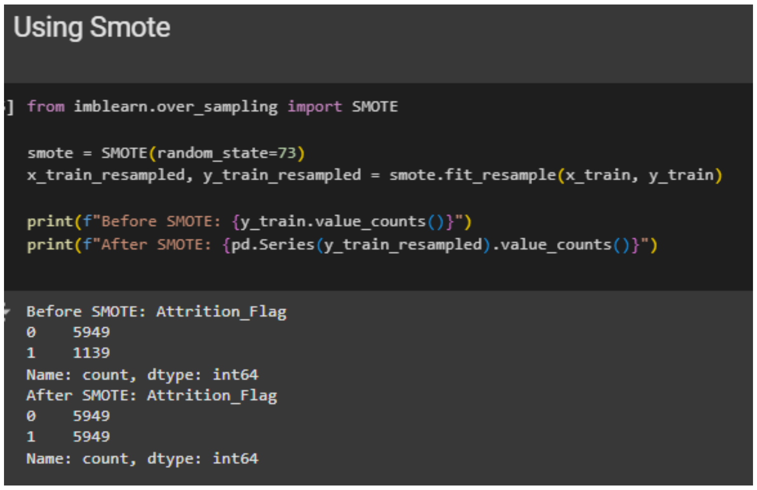
Testing Dataset For the testing dataset, a slight drop in performance metrics is observed compared to the training dataset. This is expected, as the test data represents unseen instances. Nevertheless, the overall accuracy and class-specific metrics remain strong, demonstrating the model's ability to generalize.

Validation Dataset The validation dataset results show consistent performance, with metrics comparable to those of the testing dataset. The model's precision, recall, and F1-scores confirm that the hyperparameter tuning and training process have successfully mitigated overfitting.

These results demonstrate the reliability and robustness of the model across different datasets, ensuring its suitability for deployment in real-world scenarios.

Model training after balancing

First, I balance the model using the method introduced.



Using Smote

```
] from imblearn.over_sampling import SMOTE

  smote = SMOTE(random_state=73)
  x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)

  print(f"Before SMOTE: {y_train.value_counts()}")
  print(f"After SMOTE: {pd.Series(y_train_resampled).value_counts()}")
```

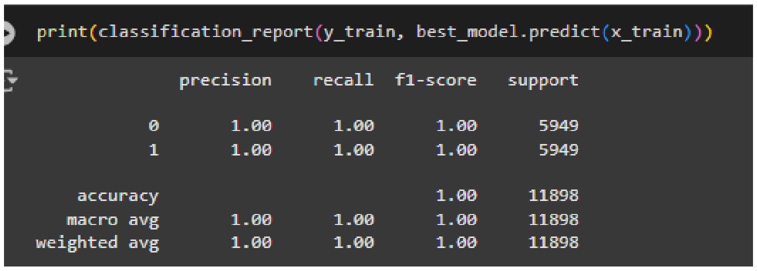
```
Before SMOTE: Attrition_Flag
0    5949
1    1139
Name: count, dtype: int64
After SMOTE: Attrition_Flag
0    5949
1    5949
Name: count, dtype: int64
```

Figure 20: Using Smote to Balance the Dataset

Performance Evaluation

The performance of the trained model was evaluated on the training, testing, and validation datasets using metrics such as precision, recall, F1-score, and overall accuracy. The classification reports for each dataset are presented below.

Training Dataset



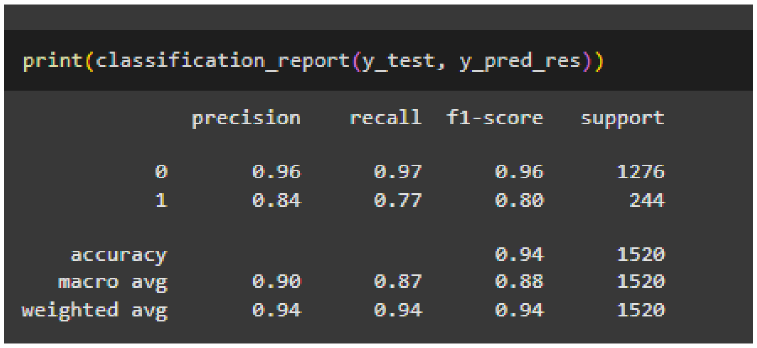
```
print(classification_report(y_train, best_model.predict(x_train)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5949
           1       1.00      1.00      1.00      5949

    accuracy                           1.00     11898
   macro avg       1.00      1.00      1.00     11898
weighted avg       1.00      1.00      1.00     11898
```

Figure 21: Model classification report for training data

The classification report for the training dataset is shown. The model achieves perfect precision, recall, and F1-scores, indicating that it has successfully captured the patterns in the training data.

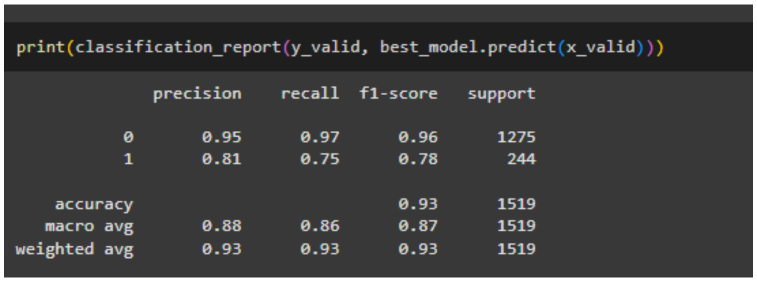
Testing Dataset



```
print(classification_report(y_test, y_pred_res))

              precision    recall  f1-score   support

           0       0.96      0.97      0.96      1276
           1       0.84      0.77      0.80       244

    accuracy                           0.94      1520
   macro avg       0.90      0.87      0.88      1520
weighted avg       0.94      0.94      0.94      1520
```

Figure 22: Model classification report for testing data

Validation Dataset



```
print(classification_report(y_valid, best_model.predict(x_valid)))

              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1275
           1       0.81      0.75      0.78       244

    accuracy                           0.93      1519
   macro avg       0.88      0.86      0.87      1519
weighted avg       0.93      0.93      0.93      1519
```

Figure 23: Model classification report for validation data

The training dataset results indicate perfect performance, which is expected as the model has been directly optimized on this data. The testing dataset results demonstrate strong generalization capabilities, with a slight drop in metrics compared to the training dataset. The validation dataset results are consistent with the testing dataset, further confirming that the model does not overfit and maintains robustness across different data splits.

These results validate the reliability of the model and its readiness for deployment in real-world scenarios.

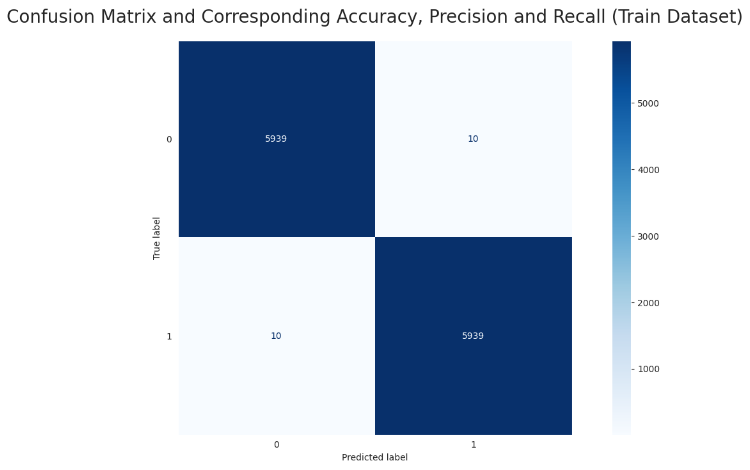


Figure 24: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Train Dataset)

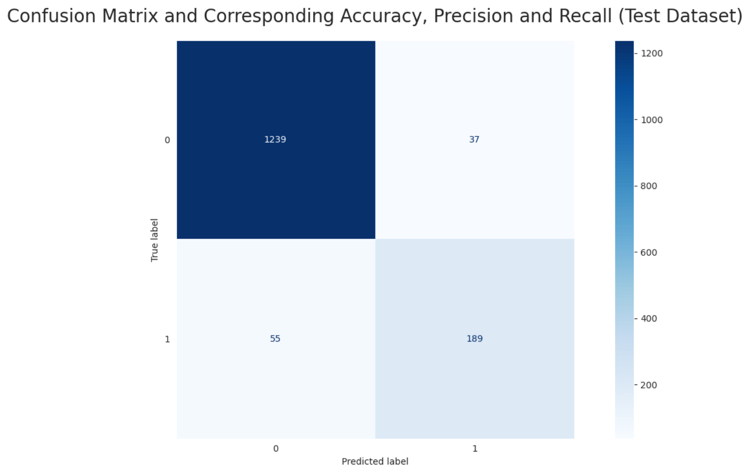


Figure 25: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Test Dataset)

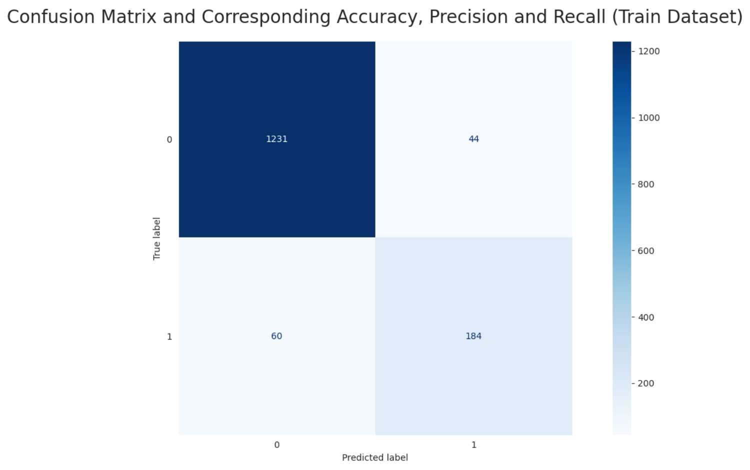


Figure 26: Confusion Matrix and Corresponding Accuracy, Precision and Recall (Test Dataset)

Comparing these matrices and the matrix confusions, we see that the previous model change was not created. This could be due to several reasons. For example, XGBoost is robust to balanced datasets. If we were using simpler models, we would expect the model accuracy to increase after balancing the data.

Data distribution

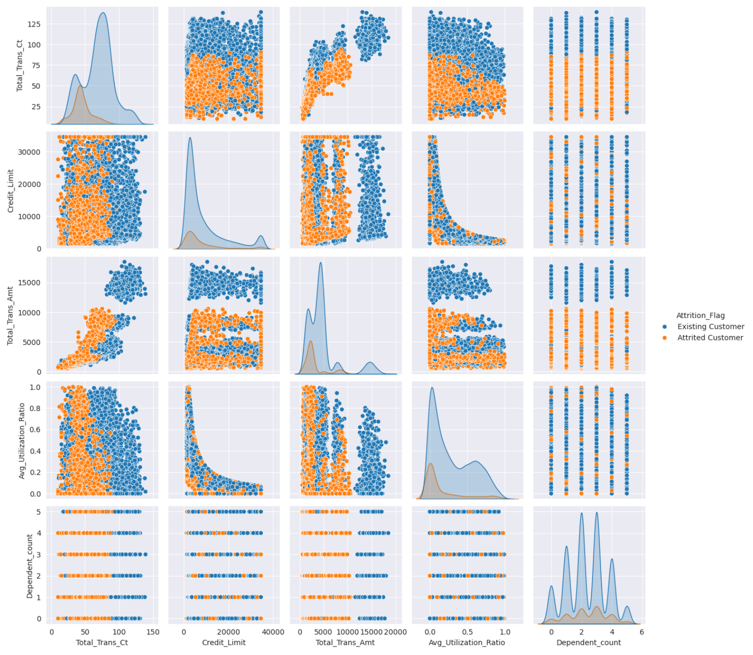


Figure 27: Data distribution