Retention Radar: Online Retail Churn Analysis

Ritam Choudhury

Techno Main Salt Lake

Abstract

This project focuses on analysing customer churn behaviour in an online retail dataset using machine learning and deep learning techniques. The primary objective is to identify customers who are likely to discontinue purchasing and to understand the key factors influencing churn. The overall workflow includes data preprocessing, exploratory data analysis, feature engineering, model training, and performance evaluation. Multiple traditional machine learning models are trained and compared using appropriate evaluation metrics to assess their predictive capability. In addition, an Artificial Neural Network (ANN) model is implemented to capture complex, non-linear relationships within the data. Experimental results demonstrate that ensemble-based machine learning models perform satisfactorily for churn prediction, while the ANN model achieves the highest accuracy indicating strong predictive performance. The findings provide valuable insights that can help online retail businesses design data-driven customer retention strategies and improve long-term customer engagement.

Introduction

Customer churn refers to the phenomenon where customers stop engaging with a business or discontinue purchasing its products or services over a given period of time. In the context of online retail, churn typically occurs when customers who were previously active no longer make repeat purchases. High customer churn rates directly impact business revenue, growth, and long-term sustainability, making churn analysis a critical area of study for data-driven decision-making.

Churn prediction plays a vital role in online retail because acquiring new customers is significantly more expensive than retaining existing ones. Online retail platforms operate in a highly competitive environment where customers have numerous alternatives available at minimal switching costs. As a result, understanding customer behaviour and identifying early signs of disengagement is essential. Accurate churn prediction enables businesses to proactively target at-risk customers through personalized offers, loyalty programs, and improved customer experiences, thereby reducing revenue loss and increasing customer lifetime value.

The objective of this project is to develop a predictive system that can accurately identify customers who are likely to churn based on their historical transaction and behavioral data. By applying machine learning and deep learning techniques, the project aims to uncover patterns and key factors that influence customer churn in an online retail setting. The study

evaluates multiple machine learning models and a deep learning–based Artificial Neural Network (ANN) to compare their effectiveness in churn prediction. The insights gained from this analysis can assist online retailers in making informed, data-driven decisions to improve customer retention and enhance overall business performance.

Environment Setup and Library Initialization

```
from google.colab import drive
drive.mount("/content/drive",force_remount=True)
```

```
Mounted at /content/drive
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf
```

This code cell prepares the notebook for the entire project workflow. It begins by mounting Google Drive, which makes it possible to access datasets stored in the Drive directly within the Colab environment. The core Python libraries are imported. Pandas is used for reading and handling structured data, NumPy supports efficient numerical computations, and Matplotlib along with Seaborn are used for visualizing data distributions and trends. Overall, this step ensures that all necessary tools are available to load the data, explore it, and apply machine and deep learning techniques effectively.

Data Loading

```
df=pd.read_csv("/content/drive/MyDrive/Datasets/online_retail_customer_churn.csv")
df
```

	Customer_ID	Age	Gender	Annual_Income	Total_Spend	Years_as_Customer	Num_of_Purchases	Average_Transaction_Amount	Num_of_Returns	Nu
0	1	62	Other	45.15	5892.58	5	22	453.80	2	
1	2	65	Male	79.51	9025.47	13	77	22.90	2	
2	3	18	Male	29.19	618.83	13	71	50.53	5	
3	4	21	Other	79.63	9110.30	3	33	411.83	5	
4	5	21	Other	77.66	5390.88	15	43	101.19	3	
...	...	...	...	...	...	...	...	...	...	
995	996	54	Male	143.72	1089.09	2	29	77.75	0	
996	997	19	Male	164.19	3700.24	9	90	34.45	6	
997	998	47	Female	113.31	705.85	17	69	187.37	7	
998	999	23	Male	72.98	3891.60	7	31	483.80	1	
999	1000	34	Other	134.86	3956.71	15	48	420.91	6	

Here, the online retail customer churn dataset is loaded from Google Drive using pandas.read_csv. Displaying the DataFrame helps confirm that the data has been read correctly and gives a quick overview of the features, values, and overall structure of the dataset.

Dataset Overview

```
1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, 1 to 1000
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Age                        1000 non-null   int64
 1   Gender                     1000 non-null   object
 2   Annual_Income              1000 non-null   float64
 3   Total_Spend                1000 non-null   float64
 4   Years_as_Customer          1000 non-null   int64
 5   Num_of_Purchases           1000 non-null   int64
 6   Average_Transaction_Amount 1000 non-null   float64
 7   Num_of_Returns             1000 non-null   int64
 8   Num_of_Support_Contacts    1000 non-null   int64
 9   Satisfaction_Score         1000 non-null   int64
 10  Last_Purchase_Days_Ago     1000 non-null   int64
 11  Email_Opt_In               1000 non-null   bool
 12  Promotion_Response         1000 non-null   object
 13  Target_Churn               1000 non-null   int64
dtypes: bool(1), float64(3), int64(8), object(2)
memory usage: 110.4+ KB
```

The df.info() function provides a concise summary of the dataset, including the total number of records, column names, data types, and non-null counts. This step helps identify which features are numerical or categorical and confirms that there are no missing values, guiding the choice of preprocessing and encoding methods later in the workflow.

Missing Value Check

```
1  df.isnull().sum()
```

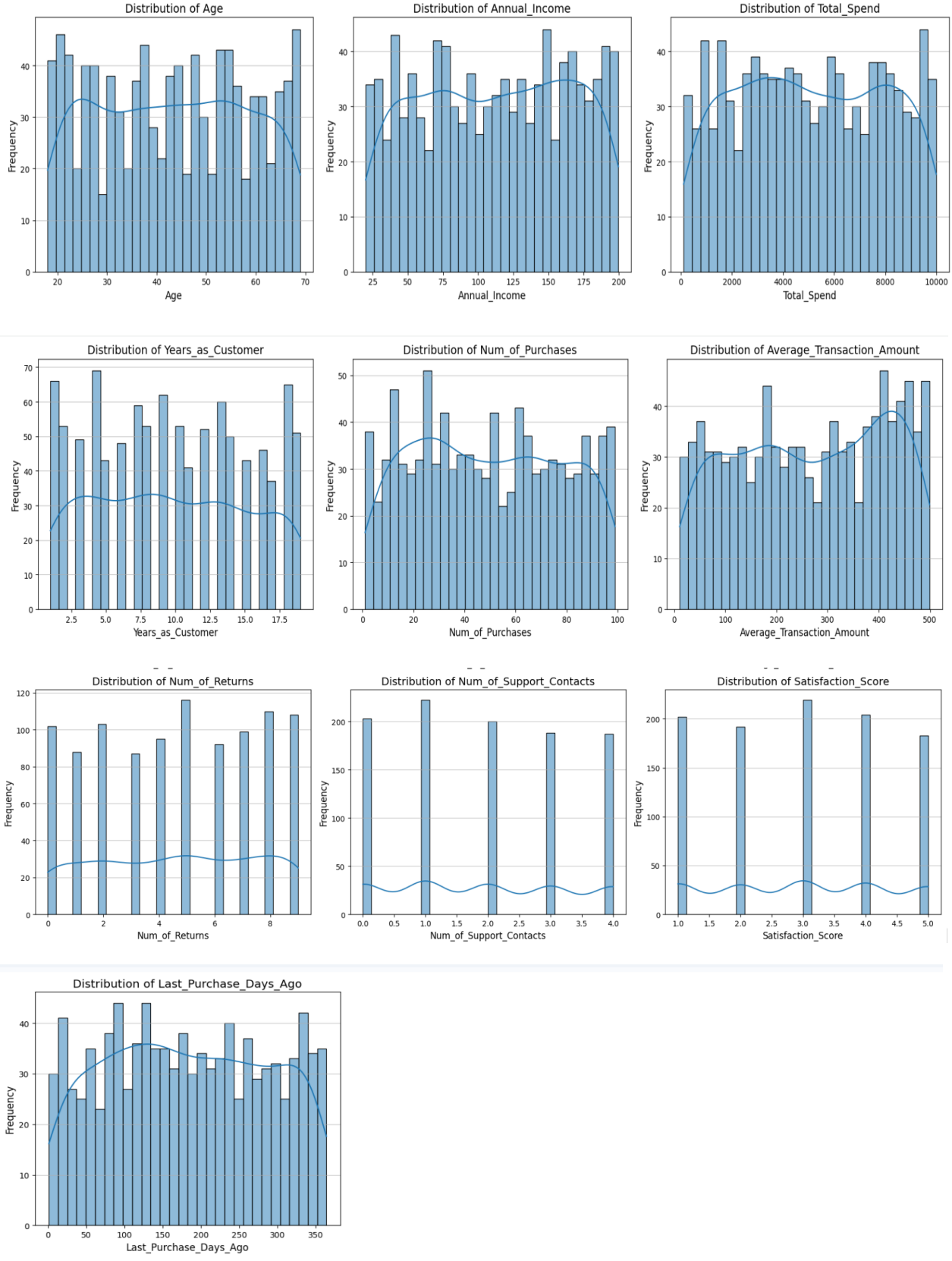
	0
Customer_ID	0
Age	0
Gender	0
Annual_Income	0
Total_Spend	0
Years_as_Customer	0
Num_of_Purchases	0
Average_Transaction_Amount	0
Num_of_Returns	0
Num_of_Support_Contacts	0
Satisfaction_Score	0
Last_Purchase_Days_Ago	0
Email_Opt_In	0
Promotion_Response	0
Target_Churn	0

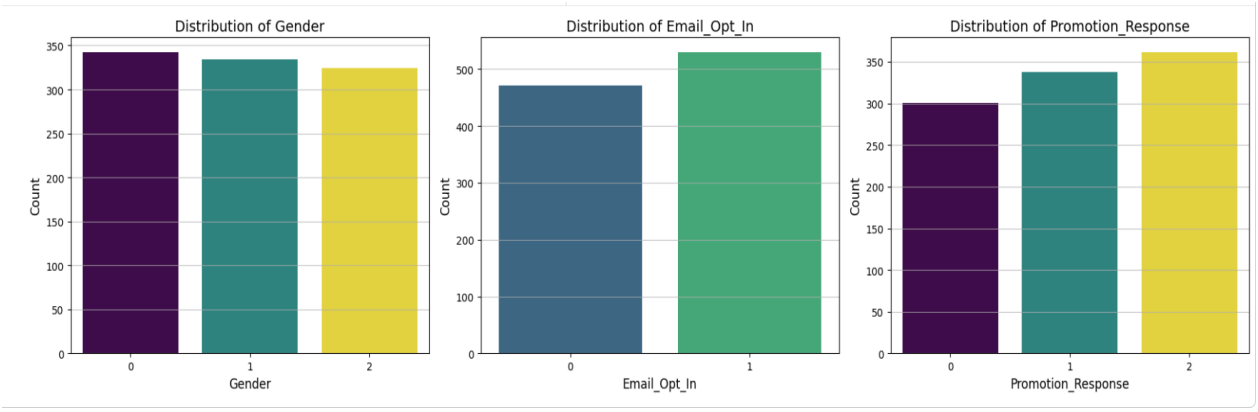
dtype: int64

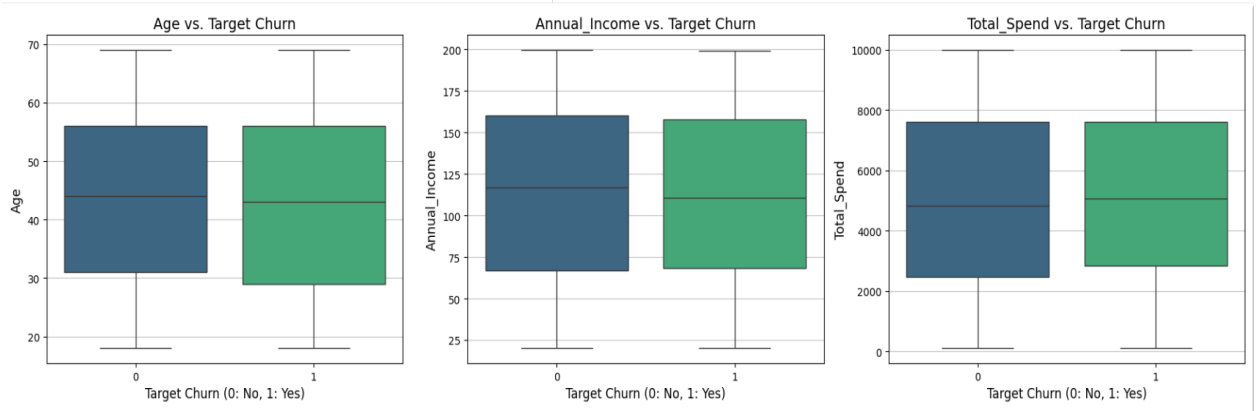
The df.isnull().sum() command checks each column for missing values by counting how many entries are null (if any) .This step confirms the data quality and verifies that no

additional imputation or cleaning is required before proceeding with preprocessing and model training.

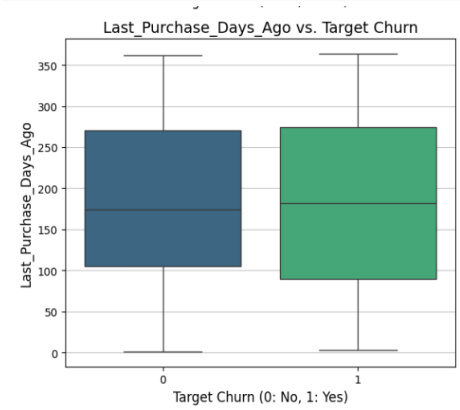
Exploratory Data Analysis

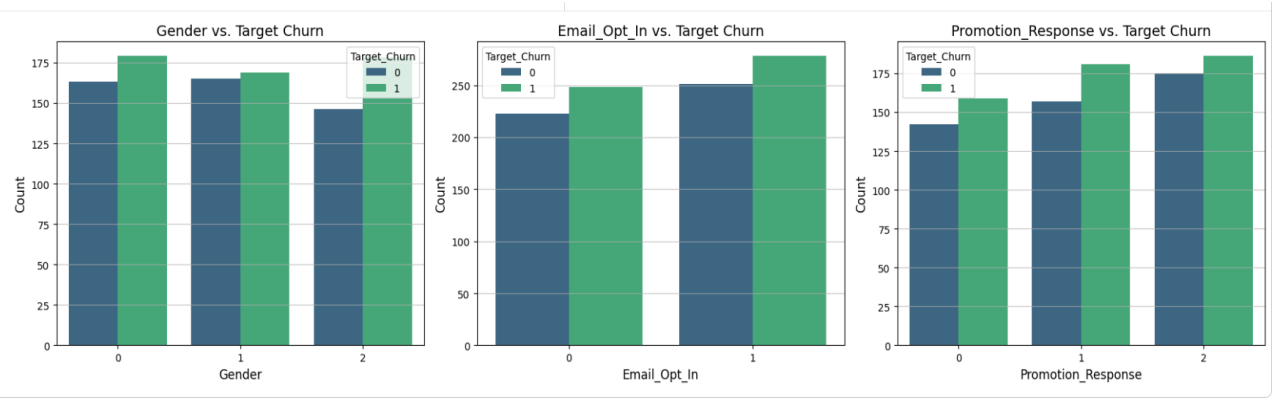



Distribution of Gender
Count
Gender
Distribution of Email_Opt_In
Count
Email_Opt_In
Distribution of Promotion_Response
Count
Promotion_Response


Age vs. Target Churn
Age
Target Churn (0: No, 1: Yes)
Annual_Income vs. Target Churn
Annual_Income
Target Churn (0: No, 1: Yes)
Total_Spend vs. Target Churn
Total_Spend
Target Churn (0: No, 1: Yes)


Years_as_Customer vs. Target Churn
Years_as_Customer
Target Churn (0: No, 1: Yes)
Num_of_Purchases vs. Target Churn
Num_of_Purchases
Target Churn (0: No, 1: Yes)
Average_Transaction_Amount vs. Target Churn
Average_Transaction_Amount
Target Churn (0: No, 1: Yes)


Last_Purchase_Days_Ago vs. Target Churn
Last_Purchase_Days_Ago
Target Churn (0: No, 1: Yes)



In this step, visualizations were used to explore how different features are related to each other and their distribution patterns. By comparing target churn against categories such as promotion response , email opt in , gender their distribution pattern can be identified. This analysis helps in understanding which features are more informative for prediction and guides decisions on feature selection, encoding methods, and model design later in the pipeline.

Key Insights from Exploratory Data Analysis

Based on the visualizations of numerical and categorical features, and their relationships with 'Target_Churn', the following key insights were observed:

Numerical Feature Distributions (Histograms):

- Most numerical features, such as 'Age', 'Annual_Income', 'Total_Spend', 'Years_as_Customer', 'Num_of_Purchases', 'Average_Transaction_Amount', 'Num_of_Returns', 'Num_of_Support_Contacts', 'Satisfaction_Score', and 'Last_Purchase_Days_Ago', show relatively uniform or somewhat normal distributions across their ranges. There are no highly skewed distributions or extreme outliers that would immediately suggest a need for aggressive transformation, although some features like 'Annual_Income' and 'Total_Spend' show multiple peaks, indicating possible sub-populations.

Categorical Feature Distributions (Count Plots):

- **Gender:** The distribution of 'Gender' (Female, Male, Other) is fairly balanced, with each category having a similar number of customers.
- **Email_Opt_In:** There's a slight imbalance, with more customers opting in for emails (represented as 1) than those who haven't (represented as 0).
- **Promotion_Response:** The distribution across 'Promotion_Response' categories (Ignored, Responded, Unsubscribed) is also relatively balanced, with 'Responded' being slightly more frequent.

Numerical Features vs. Target_Churn (Box Plots):

- **Annual_Income and Total_Spend:** There appears to be a slight difference in the median 'Annual_Income' and 'Total_Spend' between churned (1) and non-churned (0)

customers, but the overlap in distributions is significant, suggesting these features alone might not be strong predictors of churn.

- **Years_as_Customer:** The median 'Years_as_Customer' is quite similar for both churned and non-churned groups. However, the spread might differ slightly, implying that customer longevity has a nuanced relationship with churn.
- **Num_of_Purchases and Average_Transaction_Amount:** Similar to income and spend, these features show overlapping distributions, indicating no single clear threshold differentiates churned from non-churned customers.
- **Satisfaction_Score:** The box plot for 'Satisfaction_Score' against 'Target_Churn' shows a tendency for churned customers to have slightly lower satisfaction scores on average, though there's still a wide range for both groups.
- **Last_Purchase_Days_Ago:** Churned customers might have a slightly higher median for 'Last_Purchase_Days_Ago', suggesting that customers who haven't made a purchase recently are more prone to churn.
- **Other Numerical Features:** 'Age', 'Num_of_Returns', and 'Num_of_Support_Contacts' show very similar distributions between churned and non-churned groups, indicating a weaker direct relationship with churn in isolation.

Categorical Features vs. Target_Churn (Count Plots with Hue):

- **Gender:** The proportion of churned (1) and non-churned (0) customers is quite similar across all three gender categories (Female, Male, Other). This suggests that 'Gender' does not appear to be a strong predictor of churn.
- **Email_Opt_In:** Customers who have opted-in for emails (1) show a slightly higher count of churned customers compared to non-churned, but the overall proportion of churn remains relatively similar between opt-in and non-opt-in groups. This indicates a very weak relationship with churn.
- **Promotion_Response:** The distribution of churned vs. non-churned customers is fairly consistent across 'Ignored', 'Responded', and 'Unsubscribed' categories. No single response category shows a significantly higher propensity for churn.

- The initial EDA reveals that the dataset's features, when analysed in isolation, do not show extremely strong correlations or distinct patterns that clearly differentiate churned from non-churned customers.
- While some subtle tendencies were observed (e.g., slightly lower satisfaction scores and higher last purchase days for churned customers), no single feature stands out as an overwhelming predictor.
- The dataset appears balanced in terms of feature distributions and churn target, which is good for model training but implies that a more complex model might be needed to capture the nuances of churn behaviour.
- Further analysis involving feature engineering, interaction terms, and advanced machine learning models will be necessary to build an effective churn prediction model.

Data Cleaning

```
df['Target_Churn']=df['Target_Churn'].map({True:1,False:0})
df['Target_Churn']
```

	Target_Churn
0	1
1	0
2	1
3	1
4	0
...	...
995	0
996	1
997	0
998	1
999	1

1000 rows × 1 columns

dtype: int64

```
df.set_index('Customer_ID',inplace=True)
```

```
df.columns
```

```
Index(['Age', 'Gender', 'Annual_Income', 'Total_Spend', 'Years_as_Customer',
       'Num_of_Purchases', 'Average_Transaction_Amount', 'Num_of_Returns',
       'Num_of_Support_Contacts', 'Satisfaction_Score',
       'Last_Purchase_Days_Ago', 'Email_Opt_In', 'Promotion_Response',
       'Target_Churn'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder
le1=LabelEncoder()
le2=LabelEncoder()

```

```
df['Email_Opt_In'].value_counts()
df['Email_Opt_In']=le1.fit_transform(df['Email_Opt_In'])
df['Email_Opt_In'].value_counts()
```

Email_Opt_In	count
1	529
0	471

dtype: int64

```
df['Promotion_Response'].value_counts()
df['Promotion_Response']=le2.fit_transform(df['Promotion_Response'])
df['Promotion_Response'].value_counts()
```

Promotion_Response	count
2	361
1	338
0	301

dtype: int64

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

Gender	count
Female	342
Male	334
Other	324

dtype: int64

```
df['Gender']=le1.fit_transform(df['Gender'])
df['Gender'].value_counts()
```

Gender	count
0	342
1	334
2	324

dtype: int64

The Target_Churn column is converted from boolean to binary numerical values (1 for churn, 0 for non-churn) to make it suitable for machine learning models.

Customer_ID is set as the dataset index since it is a unique identifier and should not be used as a predictive feature.

The dataset columns are checked to confirm the final structure after preprocessing.

Categorical variables are converted into numerical form using LabelEncoder to ensure model compatibility.

The Email_Opt_In feature is label encoded into binary values representing customer email subscription status.

The Promotion_Response feature is encoded into numerical labels to represent different customer responses to promotions.

The Gender column is label encoded to transform categorical gender values into numerical format while preserving data distribution.

Correlation Analysis of Features in the Dataset

```
df.corr()
```

	Age	Gender	Annual_Income	Total_Spend	Years_as_Customer	Num_of_Purchases	Average_Transaction_Amount	Num_of_Returns	Num_of_Support_Contacts	Satisfaction_Score	Last_Purchase_Days_Ago	Email_Opt_In	Promotion_Response	Target_Churn
Age	1.000000	-0.067836	0.040689	-0.033285	-0.024781	0.031459	0.007184	0.000331	-0.021226	0.017408	0.009790	0.001283	-0.003399	-0.009260
Gender	-0.067836	1.000000	-0.061088	-0.028145	-0.018586	-0.024866	-0.011475	-0.013539	-0.007158	-0.019795	-0.007964	0.035658	0.025800	0.020786
Annual_Income	0.040689	-0.061088	1.000000	0.017615	-0.028853	0.025921	0.006284	0.039609	0.022472	0.016590	0.036988	0.007855	-0.006622	-0.036322
Total_Spend	-0.033285	-0.028145	0.017615	1.000000	0.037125	0.045609	-0.086144	0.010103	-0.036756	-0.004413	-0.038241	0.033981	-0.043458	0.028659
Years_as_Customer	-0.024781	-0.018586	-0.028853	0.037125	1.000000	0.017676	-0.021645	0.042259	-0.008638	-0.027812	-0.011783	-0.040759	0.012557	-0.029823
Num_of_Purchases	0.031459	-0.024866	0.025921	0.045609	0.017676	1.000000	-0.018408	-0.004420	0.009478	0.006799	-0.044072	0.010798	0.032858	-0.032772
Average_Transaction_Amount	0.007184	-0.011475	0.006284	-0.086144	-0.021645	-0.018408	1.000000	0.013257	0.042396	0.017337	0.034158	-0.025714	0.006122	-0.024723
Num_of_Returns	0.000331	-0.013539	0.039609	0.010103	0.042259	-0.004420	0.013257	1.000000	0.080650	-0.003001	0.036108	-0.008129	0.008206	0.000061
Num_of_Support_Contacts	-0.021226	-0.007158	0.022472	-0.036756	-0.008638	0.009478	0.042396	0.080650	1.000000	0.033984	-0.023189	0.024169	-0.006188	0.009593
Satisfaction_Score	0.017408	-0.019795	0.016590	-0.004413	-0.027812	0.006799	0.017337	-0.003001	0.033984	1.000000	0.018692	0.047168	-0.023423	0.022567
Last_Purchase_Days_Ago	0.009790	-0.007964	0.036988	-0.038241	-0.011783	-0.044072	0.034158	0.036108	-0.023189	0.018692	1.000000	0.023140	0.010815	-0.013823
Email_Opt_In	0.001283	0.035658	0.007855	0.033981	-0.040759	0.010798	-0.025714	-0.008129	0.024169	0.047168	0.023140	1.000000	0.000642	-0.001019
Promotion_Response	-0.003399	0.025800	-0.006622	-0.043458	0.012557	0.032858	0.006122	0.008206	-0.006188	-0.023423	0.010815	0.000642	1.000000	-0.011255
Target_Churn	-0.009260	0.020786	-0.036322	0.028659	-0.029823	-0.032772	-0.024723	0.000061	0.009593	0.022567	-0.013823	-0.001019	-0.011255	1.000000

The correlation matrix shows the pairwise linear relationships between all numerical features and the target variable (Target_Churn). Overall, most feature pairs exhibit low correlation values, indicating minimal multicollinearity in the dataset. This is beneficial for machine learning models, as it reduces redundancy and improves model stability.

Features such as Total_Spend, Num_of_Purchases, Annual_Income, and Satisfaction_Score show small but meaningful correlations with customer behaviour variables, suggesting they may contribute to churn prediction when combined with other features. The weak correlations imply that churn is influenced by multiple interacting factors rather than a single dominant feature.

The target variable (Target_Churn) has very low linear correlation with individual features, indicating that churn behaviour is non-linear and complex. This justifies the use of advanced models such as Random Forest, XGBoost, LightGBM, and ANN, which are capable of capturing non-linear relationships and feature interactions beyond simple linear dependencies.

Overall, the dataset is well-balanced in terms of feature independence, making it suitable for ensemble and tree-based models without major concerns of multicollinearity.

Feature–Target Separation and Numerical Feature Identification

```
X=df.drop('Target_Churn',axis=1)
y=df['Target_Churn']
numerical_features=X.select_dtypes(include=['int64','float64']).columns.to_list
()
```

In this step, the dataset is prepared for machine learning by separating the input features and the target variable.

The variable Target_Churn represents the output label, indicating whether a customer has churned or not. This column is removed from the feature set and stored separately as the target variable **y**. The remaining columns form the feature matrix **X**, which contains all the independent variables used for prediction.

Additionally, all **numerical features** are identified from the feature set by selecting columns with integer (int64) and floating-point (float64) data types. These numerical features are stored separately, as they typically require preprocessing steps such as scaling or normalization before model training.

This separation ensures a structured and efficient workflow for further preprocessing, feature engineering, and model development.

Importing Machine Learning and Deep Learning Models with Evaluation Metrics

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
```

This cell imports libraries for data preprocessing, model training, and evaluation. It includes classical machine learning models (KNN, Random Forest), ensemble and boosting models (XGBoost, LightGBM), and tree-based model(Decision Tree). Evaluation metrics such as accuracy, classification report, and ROC-AUC are also imported to assess and compare model performance.

Feature Scaling and Train–Test Split

```
sc=StandardScaler()
X[numerical_features]=sc.fit_transform(X[numerical_features])
```

```
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X,y,
test_size=0.2,random_state=42,stratify=y)
```

Feature Scaling**:**
Numerical features are standardized using StandardScaler, which transforms them to have a mean of 0 and a standard deviation of 1. This ensures that all numerical variables are on the same scale and prevents features with larger values from dominating the model.

Train–Test Split**:**
The dataset is split into training (80%) and testing (20%) sets. Stratification is applied to preserve the original class distribution of the target variable, and a fixed random state ensures reproducibility..

K-Nearest Neighbors (KNN) Model Training and Evaluation

Mathematical Intuition :

KNN is a distance-based, non-parametric learning algorithm, meaning it does not assume any underlying data distribution and does not learn explicit model parameters during training. Instead, it stores the entire training dataset.

For a given test point x, the algorithm computes the distance (most commonly Euclidean distance) between $x$and every training data point x_i, defined as:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

After computing these distances, the K closest data points are selected to form the neighborhood $N_K(x)$. The predicted class label is then assigned using majority voting, where the class that appears most frequently among the K neighbors becomes the final prediction:

$$\hat{y} = \text{mode}\{y_i \mid x_i \in N_K(x)\}$$

The value of K controls model behaviour:

- Small Kmakes the model sensitive to noise (high variance).

- Large K smooths predictions but may underfit (high bias).

Feature scaling is critical because KNN relies entirely on distance calculations. Without scaling, features with larger numerical ranges dominate the distance metric, leading to biased neighbour selection and poor model performance.

```
model1=KNeighborsClassifier()
model1.fit(X_train_scaled,y_train)
y_pred=model1.predict(X_test_scaled)
print('---Model 1 Performance ---')
print(classification_report(y_test,y_pred))
auc_knn = roc_auc_score(y_test, y_pred)
print(f'AUC Score for KNeighbors Classifier: {auc_knn:.4f}')
```

```
---Model 1 Performance ---
              precision    recall  f1-score   support

           0       0.55      0.55      0.55        95
           1       0.59      0.59      0.59       105

    accuracy                           0.57       200
   macro avg       0.57      0.57      0.57       200
weighted avg       0.57      0.57      0.57       200

AUC Score for KNeighbors Classifier: 0.5689
```

This cell initializes a KNN classifier, trains it on the scaled training data, and predicts churn on the test set. The model's performance is evaluated using a classification report (precision, recall, F1-score) and the ROC-AUC score to measure its ability to distinguish between churn and non-churn classes.

Random Forest Model Training and Evaluation

Mathematical Intuition :

Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees to improve overall performance and robustness.

Instead of training a single tree on the entire dataset, Random Forest uses bootstrap sampling, where each tree is trained on a random subset of the data selected with replacement.

During tree construction, a random subset of features is considered at each split rather than all features. This feature randomness ensures that the trees are decorrelated, meaning they learn different patterns from the data.

Each individual decision tree $T_i(x)$ produces a class prediction for an input x. The final prediction is obtained through majority voting across all trees:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), \ldots, T_n(x)\}$$

This ensemble approach significantly reduces variance compared to a single decision tree, making the model more stable and less prone to overfitting.

While individual trees may overfit locally, their combined decision generalizes better to unseen data.

```
rf_model=RandomForestClassifier(n_estimators=150)
rf_model.fit(X_train_scaled,y_train)
y_pred=rf_model.predict(X_test_scaled)
print("--- Model 2 Performance ---",classification_report(y_test,y_pred))
auc_rf=roc_auc_score(y_test,y_pred)
print(f'AUC Score for Random Forest Classifier: {auc_rf:.4f}')
```

```
--- Model 2 Performance ---              precision    recall  f1-score   support

           0       0.49      0.43      0.46        95
           1       0.53      0.59      0.56       105

    accuracy                           0.52       200
   macro avg       0.51      0.51      0.51       200
weighted avg       0.51      0.52      0.51       200

AUC Score for Random Forest Classifier: 0.5110
```

This cell trains a Random Forest classifier with 150 decision trees on the scaled training data. It predicts churn on the test set and evaluates performance using a classification report and ROC-AUC score to assess overall classification quality.

Extreme Gradient Boosting (XGBoost) Model Training and Evaluation

Mathematical Intuition :
XGBoost is a gradient boosting ensemble algorithm that builds models sequentially, where each new decision tree corrects the errors made by the previous ensemble. XGBoost minimizes a regularized objective function using gradient-based optimization.

The objective function at iteration $t$is:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

where:

- $l(\cdot)$is the loss function (log loss for binary classification),
- $f_t(x)$is the new tree added at iteration t,
- $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum w^2$is a regularization term that penalizes model complexity.
- XGBoost uses first- and second-order derivatives (gradient and Hessian) of the loss function to find optimal splits, making training faster and more accurate. Parameters such as max_depth control tree complexity, learning rate scales the contribution of each tree, while subsample and colsample_bytree introduce randomness to reduce overfitting.

For binary classification, XGBoost outputs probabilities using the logistic (sigmoid) function:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-\hat{y}}}$$

```python
xgb_model=XGBClassifier(n_estimators=200,
    max_depth=10,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="binary:logistic",
    eval_metric="logloss",
    random_state=42)
xgb_model.fit(X_train_scaled,y_train)
y_pred=xgb_model.predict(X_test_scaled)
print("--- Model 3 Performance ---",classification_report(y_test,y_pred))
auc_rf=roc_auc_score(y_test,y_pred)
print(f'AUC Score for XGBoostClassifier: {auc_rf:.4f}')
```

```
--- Model 3 Performance ---              precision    recall  f1-score   support

           0       0.51      0.45      0.48        95
           1       0.55      0.61      0.58       105

    accuracy                           0.54       200
   macro avg       0.53      0.53      0.53       200
weighted avg       0.53      0.54      0.53       200

AUC Score for XGBoostClassifier: 0.5311
```

This cell initializes an XGBoost classifier with tuned hyperparameters and trains it on the scaled training data. The trained model predicts churn on the test set, and its performance is evaluated using a classification report and the ROC-AUC score to measure its ability to distinguish between churned and non-churned customers.

Light Gradient Boosting Machine (LightGBM) Model Training and Evaluation

Mathematical Intuition :
LightGBM is a gradient boosting framework that builds decision trees sequentially to minimize a loss function, similar to XGBoost, but with key optimizations for speed and efficiency. It grows trees leaf-wise instead of level-wise, meaning it expands the leaf that results in the maximum reduction in loss, leading to faster convergence and better accuracy.

At each iteration t, LightGBM adds a new tree $f_t(x)$to minimize the regularized objective:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

LightGBM uses histogram-based binning to discretize continuous features, which reduces memory usage and speeds up split finding. The num_leaves parameter controls model complexity, while max_depth limits tree growth to prevent overfitting. Randomness introduced through subsample and colsample_bytree improves generalization.For binary classification, LightGBM outputs probabilities using the logistic loss (log loss) function:

```python
from lightgbm import LGBMClassifier
lgbm_model=LGBMClassifier(n_estimators=300,
    learning_rate=0.1,
    max_depth=-5,
    num_leaves=31,
    subsample=0.8,
    colsample_bytree=0.8,
    force_col_wise=True,
    random_state=42)
lgbm_model.fit(X_train_scaled,y_train)
y_pred=lgbm_model.predict(X_test_scaled)
print("--- Model 4 Performance ---",classification_report(y_test,y_pred))
auc_rf=roc_auc_score(y_test,y_pred)
print(f'AUC Score for LGBMClassifier: {auc_rf:.4f}')
```

```
    --- Model 4 Performance ---              precision    recall  f1-score   support
...
                0       0.54      0.48      0.51        95
                1       0.57      0.63      0.60       105

         accuracy                           0.56       200
        macro avg       0.56      0.56      0.56       200
     weighted avg       0.56      0.56      0.56       200

    AUC Score for LGBMClassifier: 0.5564
```

This cell initializes a LightGBM classifier with tuned hyperparameters and trains it on the scaled training data. The model predicts churn on the test set, and performance is evaluated using a classification report and ROC-AUC score to measure classification effectiveness.

Decision Tree Classifier Model Training and Evaluation

Mathematical Intuition :

A Decision Tree is a rule-based, non-linear model that partitions the feature space into homogeneous regions using a sequence of decision rules. At each internal node, the algorithm selects the feature and split value that best separates the data based on a purity criterion.

For classification, impurity is commonly measured using the Gini Index:

$$\text{Gini}(S) = 1 - \sum_{k=1}^{K} p_k^2$$

or Entropy:

$$\text{Entropy}(S) = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

where $p_k$is the proportion of samples belonging to class $k$in subset S.

For a candidate split, the algorithm computes the Information Gain, which measures the reduction in impurity:

$$\text{Information Gain} = I(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} I(S_i)$$

The split that maximizes this gain is selected .This recursive splitting continues until a stopping condition is met (e.g., maximum depth, minimum samples per leaf).

Each leaf node stores the majority class of the samples it contains, and predictions are made by traversing the tree according to feature thresholds .

Decision Trees naturally capture feature interactions and non-linear decision boundaries, but without constraints, they tend to overfit. Hence, depth and sample limits are often applied to improve generalization.

```python
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled,y_train)
y_pred=dt_model.predict(X_test_scaled)
print("--- Model 5 Performance ---",classification_report(y_test,y_pred))
auc_rf=roc_auc_score(y_test,y_pred)
print(f'AUC Score for DecisionTreeClassifier: {auc_rf:.4f}')
```

```
--- Model 5 Performance ---              precision    recall  f1-score   support

           0       0.52      0.46      0.49        95
           1       0.56      0.61      0.58       105

    accuracy                           0.54       200
   macro avg       0.54      0.54      0.54       200
weighted avg       0.54      0.54      0.54       200

AUC Score for DecisionTreeClassifier: 0.5363
```

This cell initializes a Decision Tree classifier, trains it on the scaled training data, and predicts churn on the test set. Model performance is evaluated using a classification report and the ROC-AUC score to assess classification effectiveness.

Comparative Performance Evaluation of Classification Models

From the comparative evaluation, XGBoost and LightGBM perform better than the other models. Their relatively higher F1-score and ROC-AUC values indicate a stronger ability to capture non-linear relationships and complex decision boundaries compared to KNN and a single Decision Tree. Boosting-based models improve performance by sequentially correcting the errors of previous models, leading to better generalization.

Although the dataset is class-balanced, the overall accuracy lies between 50–60%. This is because the correlation analysis shows that individual features have very weak linear relationships with the target variable. Customer churn is therefore influenced by subtle and non-linear patterns rather than a single dominant feature. As a result, even advanced models achieve moderate accuracy.

In this context, accuracy alone is not sufficient to judge model performance. Metrics such as ROC-AUC, Precision, Recall, and F1-score provide a more reliable assessment of how well the models distinguish between churned and non-churned customers.

Conclusion

In this study, multiple machine learning models were implemented and evaluated for customer churn prediction, including K-Nearest Neighbors, Decision Tree, Random Forest, XGBoost, and LightGBM. A comprehensive preprocessing pipeline was applied to ensure data consistency and suitability for modelling. Model performance was assessed using accuracy, precision, recall, F1-score, and ROC-AUC metrics.

The results indicate that ensemble-based boosting models, particularly XGBoost and LightGBM, outperform other approaches in terms of overall predictive capability. Their superior performance can be attributed to their ability to model complex, non-linear

relationships and reduce prediction errors through iterative learning. Tree-based ensemble methods also demonstrated greater stability compared to distance-based and single-tree models.

Despite the dataset being class-balanced, the achieved accuracy remained moderate. This reflects the inherent complexity of customer churn behaviour, where individual features exhibit weak direct relationships with the target variable. Consequently, churn prediction requires models capable of capturing subtle and non-linear patterns rather than relying on simple linear associations.

Overall, the study highlights that ROC-AUC, F1-score, and recall are more informative metrics than accuracy alone for evaluating churn prediction models. Among all the models tested, XGBoost and LightGBM are identified as the most suitable choices for this dataset due to their consistent and reliable performance.