Department of Mathematics and Computer

Science Faculty of Data Analysis

UNIME

Machine Learning

Presentation

Churn Prediction

Student: **Zhengis Sanat**

Matricola: **537902**

Professor: **Giacomo Fiumara**

Table of Contents

Introduction

Data analysis and machine learning are essential for extracting insights and predicting outcomes from datasets. Businesses must continuously analyze metrics like sales, customer counts, and contract types to stay competitive. By ensuring data integrity and updating it regularly, we unlock opportunities for deeper analysis and learning.

This project focuses on **Predicting Customer Churn for a Telecommunications Company** using machine learning techniques.

Goal: To predict customer churn and develop retention strategies.

Focus:

- Analyze factors driving churn.
- Develop and evaluate various machine learning models.
- Provide actionable insights for customer retention.

Key Deliverables:

1. **Dataset Exploration**: Preprocess and understand the dataset.
2. **Model Development**: Train models such as Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting.
3. **Model Evaluation**: Assess performance using accuracy, ROC-AUC, and feature importance.
4. **Interpretation**: Use LIME for model interpretation and prediction explanations.

Customer churn occurs when customers stop using a company's services, leading to lost revenue. By predicting churn, businesses can proactively retain at-risk customers. This project uses machine learning to analyze customer data (e.g., spending, services used) and predict potential churn, helping companies act early to retain customers and reduce losses.

We will take a detailed look at each stage of the project with screenshots and a description of the code to fully understand each stage and the progress of development.

Libraries in Machine Learning

Libraries are foundational tools used at every stage of a machine learning project. They provide pre-written code to perform a wide variety of functions, from data manipulation to model evaluation, which significantly speeds up the process of implementing machine learning algorithms. Instead of writing algorithms from scratch, we can leverage these libraries to focus on solving the problem at hand. Machine learning libraries help us access a range of tools and techniques for tasks such as data cleaning, transformation, visualization, model building, and performance evaluation.

In this project, we use several key libraries to facilitate the data processing, model training, evaluation, and interpretation.

1. Pandas (import pandas as pd)

- **Purpose**: Used for data manipulation and analysis, such as loading, cleaning, and preprocessing data.

2. Matplotlib (import matplotlib.pyplot as plt)

- **Purpose**: Provides tools for creating static visualizations like graphs and plots.

3. Seaborn (import seaborn as sns)

- **Purpose**: Built on Matplotlib, it simplifies creating informative and attractive statistical graphics.

4. Scikit-learn (import sklearn)

- **Purpose**: Provides machine learning algorithms and tools for data preprocessing, model evaluation, and hyperparameter tuning.
- **Key Features**:
    - SimpleImputer: Handles missing values.
    - LabelEncoder: Encodes categorical data.
    - train_test_split: Splits the data into training and testing sets.
    - Models like LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier.

5. NumPy (import numpy as np)

- **Purpose**: Used for numerical computations and handling arrays.

6. LIME (import lime, import lime.lime_tabular)

- **Purpose**: Helps interpret machine learning model predictions, providing insights into the decision-making process.

The screenshot of imported libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, confusion_matrix
import numpy as np
import lime
import lime.lime_tabular
```

The first step of the project is
Understanding the Dataset
Activity: Dataset Exploration

1. **Load the dataset:**

```
# Load the dataset
dataset = pd.read_csv('dataset_churn.csv')
```

Loads the customer churn dataset from a CSV file into a Pandas DataFrame for further analysis.

2. **Preview the dataset:**

```
# Preview the dataset
dataset.head()
```

	Unnamed: 0	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges
0	0	08729464-bde6-43bc-8f63-a357096feab1	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.88	931.49
1	1	af95bc95-baf4-4318-a21d-70d2ea3148b7	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46
2	2	1fe7eee6-2227-4400-9998-4d993f4a60fd	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73
3	3	f736fe7b-1b44-4acd-84c2-21c4aef648be	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	78.16	4452.13
4	4	4b40d12d-7633-4309-96b8-aee675ea20ae	60.0	Male	52	Fiber optic	Yes	Yes	Two year	Electronic check	30.33	1569.73

Displays the first 5 rows of the dataset to quickly check its structure and contents

3. **Remove leading/trailing spaces from column names**:

```
# Strip any leading/trailing spaces from the column names
dataset.columns = dataset.columns.str.strip()
```

Cleans column names by removing any extra spaces to avoid errors during data manipulation.

4. **Remove unnecessary columns:**

```
# Remove unnecessary columns
dataset = dataset.drop(columns=['Unnamed: 0', 'CustomerID'])
```

Drops irrelevant columns like Unnamed: 0 and CustomerID to focus on useful data for analysis.

5. **Check the dataset after removal:**

```
# Check the dataser after removing unnecessary columns
dataset.head()
```

	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	S
0	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.88	931.49	No	
1	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46	Yes	
2	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73	Yes	
3	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	78.16	4452.13	No	
4	60.0	Male	52	Fiber optic	Yes	Yes	Two year	Electronic check	30.33	1569.73	Yes	

Verifies the dataset after cleaning to ensure the correct columns were removed.

6. **Check the dataset shape:**

```
print(f"Dataset shape: {dataset.shape}")
```

```
Dataset shape: (3749, 15)
```

This command prints the shape of the dataset, which tells you the number of rows (samples) and columns (features) in the dataset. It helps you understand the size of the dataset.

7. **Describe the data**:

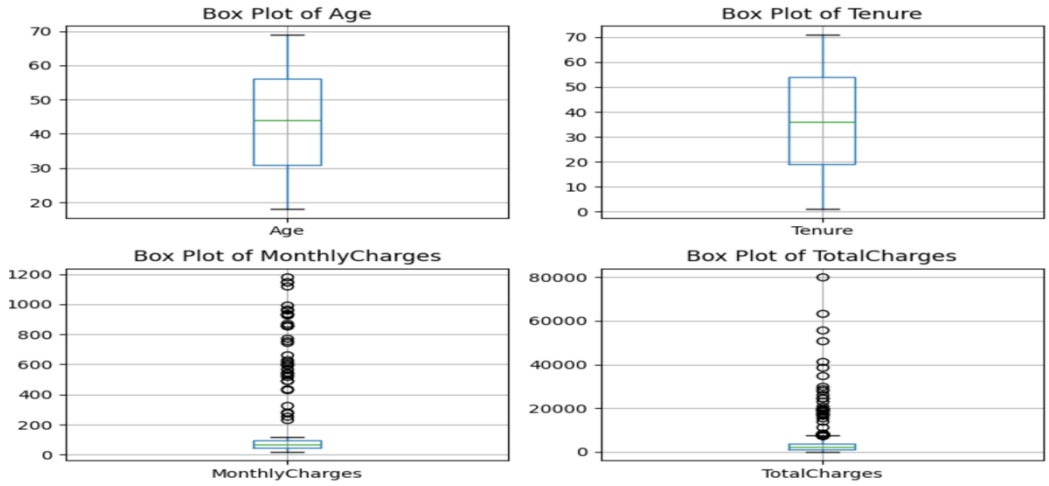
```
print(dataset.describe())
```

	Age	Tenure	MonthlyCharges	TotalCharges
count	3562.000000	3749.000000	3749.000000	3749.000000
mean	43.655531	36.264070	75.844318	2718.968266
std	14.914474	20.505528	73.062971	3211.879149
min	18.000000	1.000000	20.000000	13.190000
25%	31.000000	19.000000	44.570000	1076.240000
50%	44.000000	36.000000	69.590000	2132.260000
75%	56.000000	54.000000	95.540000	3619.710000
max	69.000000	71.000000	1179.300000	79951.800000

This command provides a summary of the numerical columns in the dataset, including statistics like count, mean, standard deviation, min, max, and quartiles. It helps in understanding the distribution and range of the numerical features.

8. **Visualization of the distribution of numerical features using box plots:**

```
# Visualizing numerical features with box plots
numerical_columns = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

plt.figure(figsize=(8, 6))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    dataset.boxplot(column=column)
    plt.title(f'Box Plot of {column}')
plt.tight_layout()
plt.show()
```



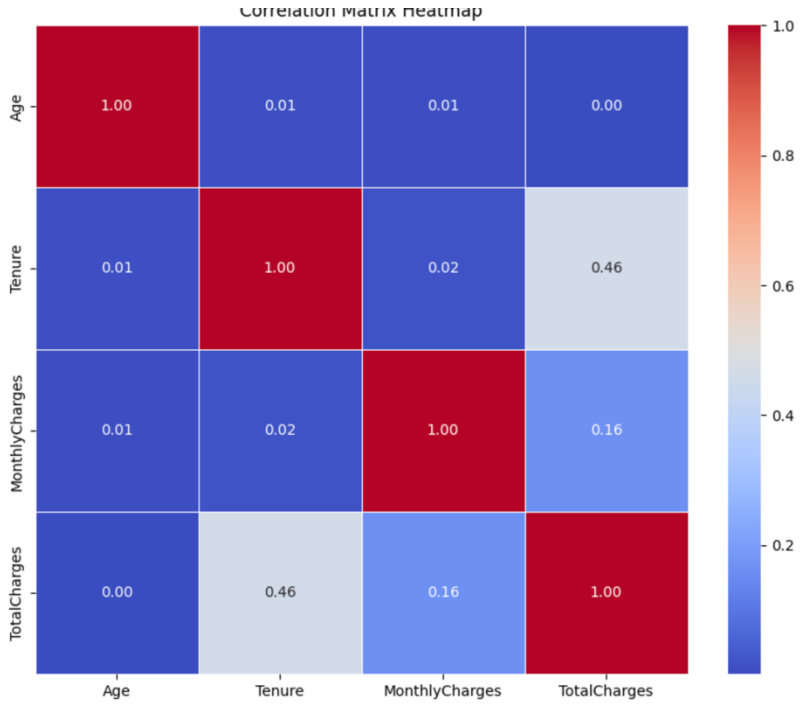
This code is used to **visualize the distribution** of numerical features, helping identify **outliers**, understand **data spread**, and assess **data quality**. Box plots make it easier to spot trends and prepare the data for further analysis or machine learning modeling.

From the visualization above, we can observe that features like **Tenure**, **MonthlyCharges**, and **TotalCharges** contain significant outliers and extreme values. These outliers can distort the analysis and negatively impact the performance of machine learning models. Therefore, it's important to clean and handle these outliers appropriately to avoid bias and ensure more accurate model predictions.

9. Pearson Correlation Matrix for numerical features:

```
# Generating the correlation matrix
correlation_matrix = dataset[numerical_columns].corr()

# Plotting the correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

This matrix shows the linear relationship between pairs of variables, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation), with 0 indicating no linear correlation.



From the matrix above we can draw the following conclusions:

- **Age** is not highly related to the other variables, suggesting it might not be a strong predictor for churn or other behaviors in this dataset.
- **Tenure** is more strongly correlated with **TotalCharges**, which is expected, as customers with longer tenures will likely accumulate higher total charges.
- **MonthlyCharges** has a weak correlation with **TotalCharges**, showing that while monthly payments influence total charges, other factors likely play a role as well.

The second step of the project is

Data Preprocessing

Activity: Data Cleaning and Preparation

1. **Check for the missing values:**

```
# Check for the missing values
dataset.isnull().sum()
```

```
Age                187
Gender               0
Tenure               0
Service_Internet   721
Service_Phone        0
Service_TV           0
Contract             0
PaymentMethod      187
MonthlyCharges       0
TotalCharges         0
StreamingMovies      0
StreamingMusic       0
OnlineSecurity       0
TechSupport          0
Churn                0
dtype: int64
```

The code will output the count of missing values for each column in the dataset, allowing you to see if any features need to be cleaned or imputed (e.g., replacing missing values with mean, median, or other strategies).

From the output, we can see that **Age**, **Service_Internet**, and **PaymentMethod** have a significant amount of missing values. These missing values need to be handled appropriately to ensure the integrity of the project and avoid bias in the analysis.

2. Handling the missing values:

```
# Handle missing values using median for numeric columns and mode for categorical columns
imputer_median = SimpleImputer(strategy='median')
dataset['Age'] = imputer_median.fit_transform(dataset[['Age']]).ravel()

imputer_mode = SimpleImputer(strategy='most_frequent')
dataset['PaymentMethod'] = imputer_mode.fit_transform(dataset[['PaymentMethod']]).ravel()
dataset['Service_Internet'] = imputer_mode.fit_transform(dataset[['Service_Internet']]).ravel()
```

```
# Check for remaining missing values
print(dataset.isnull().sum())
```

```
Age                0
Gender             0
Tenure             0
Service_Internet   0
Service_Phone      0
Service_TV         0
Contract           0
PaymentMethod      0
MonthlyCharges     0
TotalCharges       0
StreamingMovies    0
StreamingMusic     0
OnlineSecurity     0
TechSupport        0
Churn              0
dtype: int64
```

To handle the missing values in the dataset, we used the following approach:

i. **For numerical columns** (Age), we imputed missing values using the **median** value. This is a common method for replacing missing values in numerical data, as it helps maintain the distribution without being affected by outliers.

ii. **For categorical columns** (PaymentMethod and Service_Internet), we imputed missing values using the **most frequent value (mode)**, which is a typical strategy for categorical data.

After performing the imputation, the check for missing values (dataset.isnull().sum()) shows that there are now **no missing values** in the dataset, as all columns have 0 missing values. This ensures the data is clean and ready for further analysis or modeling.

3. Encoding for categorical features:

```
# Label encode categorical columns
categorical_features = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',
                        'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',
                        'OnlineSecurity', 'TechSupport', 'Churn']

labelencoder = LabelEncoder()
dataset[categorical_features] = dataset[categorical_features].apply(labelencoder.fit_transform)
```

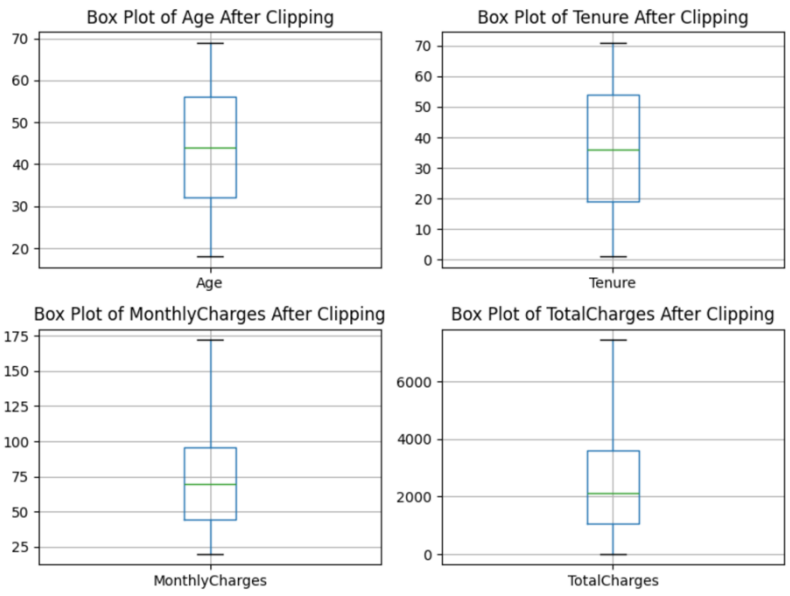
At this stage, we have **converted categorical features into numerical values** using **LabelEncoder**. This is necessary because machine learning models require numerical data to process. The **`fit_transform` method** was used, which both trains the encoder on the data and immediately applies it to convert the categorical values into numerical labels. This process is essential for preparing the data for further analysis or training machine learning models.

4. Removing outliers using the interquartile range (IQR) method and replotting box plots to observe the changes.

```
# Clipping outliers based on the IQR method
columns_to_clip = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

for column in columns_to_clip:
    Q1 = dataset[column].quantile(0.25)
    Q3 = dataset[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    dataset[column] = np.clip(dataset[column], lower_bound, upper_bound)
```

```
# Visualize the distribution after clipping
plt.figure(figsize=(8, 6))
for i, column in enumerate(columns_to_clip, 1):
    plt.subplot(2, 2, i)
    dataset.boxplot(column=column)
    plt.title(f'Box Plot of {column} After Clipping')
plt.tight_layout()
plt.show()
```

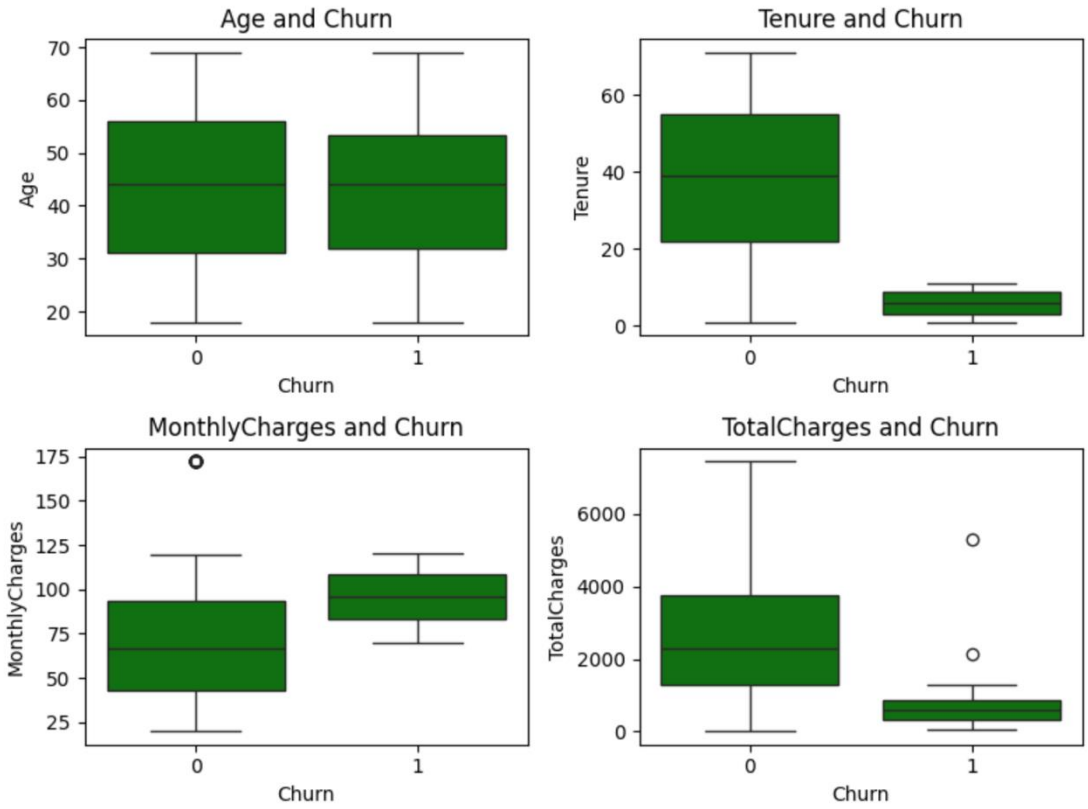
At this stage, we used the **IQR method** to remove outliers in the numerical features (`Age`, `Tenure`, `MonthlyCharges`, `TotalCharges`). Values outside the range of 1.5 times the IQR were replaced with the nearest valid values. This helps prevent extreme values from skewing the data, improving the accuracy of machine learning models. We then visualized the data again to confirm the changes were effective.



The third step of the project is
Exploratory Data Analysis (EDA)
Activity: In-depth EDA

We used **box plots for Exploratory Data Analysis (EDA)** to investigate the **relationship between the numerical features (like Age, Tenure, MonthlyCharges, TotalCharges) and the target variable Churn**. Box plots allow us to visualize the distribution, central tendency (median), and outliers for each feature, separated by churn status (0 for no churn and 1 for churn).

```
# Box plots for numerical features vs Churn
plt.figure(figsize=(8, 6))
for i, col in enumerate(numerical_columns):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x='Churn', y=col, data=dataset, color='green')
    plt.title(f'{col} and Churn')
    plt.tight_layout()

plt.show()
```



- **Age and Churn**: There is no significant difference between the Age of customers who churn and those who don't. The distribution is quite similar for both groups.
- **Tenure and Churn**: Customers who have been with the company for a longer period (Tenure) seem less likely to churn. The box plot shows that Tenure is generally lower for customers who churn.

- **MonthlyCharges and Churn**: Higher MonthlyCharges seem to be associated with customers who churn, as the upper part of the box for churned customers (1) tends to have higher values.
- **TotalCharges and Churn**: Similar to MonthlyCharges, higher TotalCharges are seen for customers who have churned. However, there are outliers, which may indicate certain customers with exceptionally high charges.

Next, we move on to analyzing **categorical features** in relation to **Churn** using **count plots**.

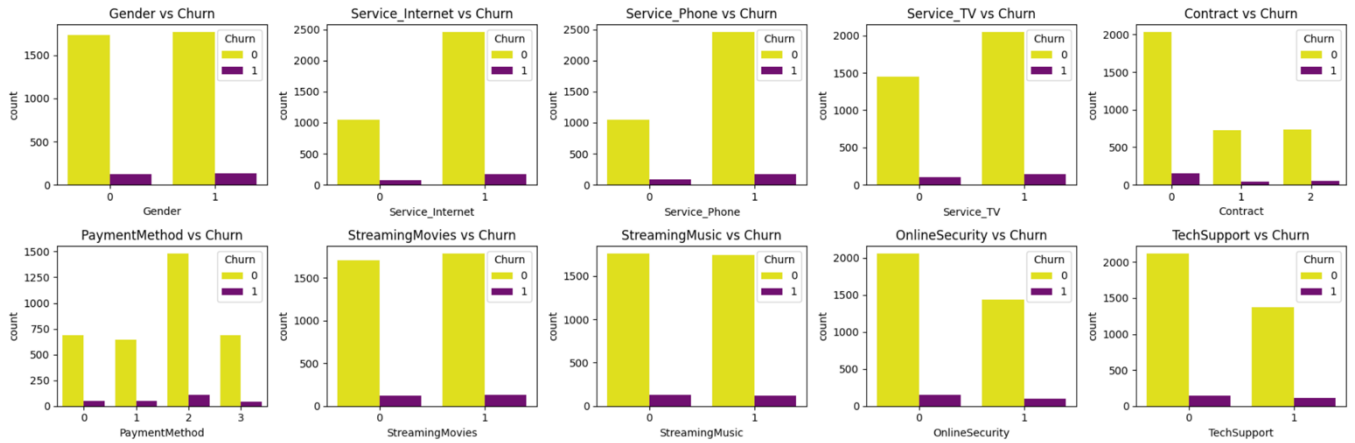
We created count plots for features like Gender, Service_Internet, Contract, PaymentMethod, etc., to see how each feature is associated with churn. The bars are colored to show whether a customer churned (purple) or didn't (yellow).

Key Insights:

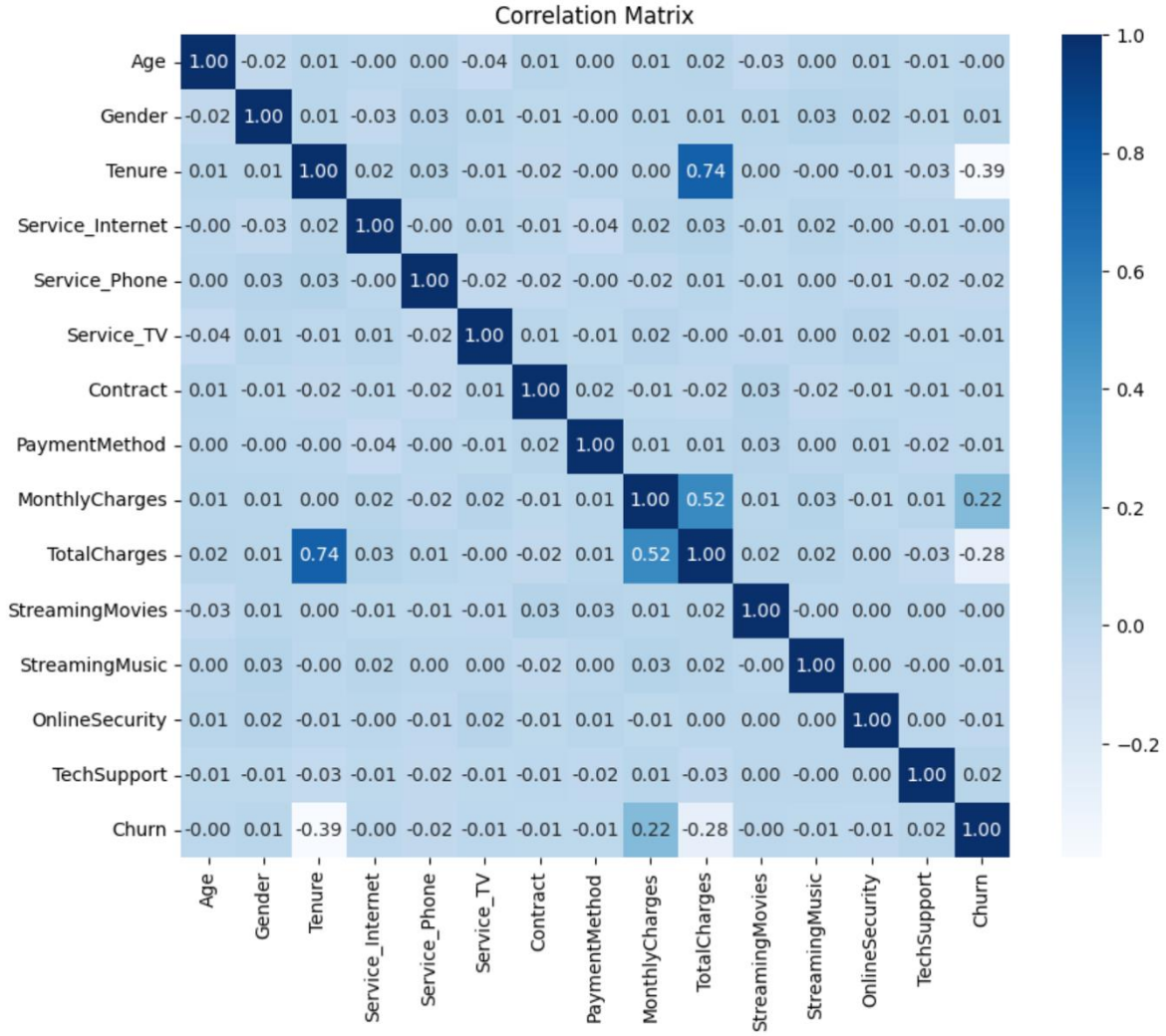
- **Contract**: Customers with **month-to-month** contracts are significantly more likely to churn, compared to those with longer-term contracts (one-year or two-year).
- **Service_Internet**: Customers without **internet service** tend to have a higher churn rate.
- **PaymentMethod**: Customers using **Electronic Check** or **Bank Transfer** have a higher churn rate.
- **StreamingMovies** and **StreamingMusic**: Customers who do **not use streaming services** show a higher churn rate.
- **OnlineSecurity**: Customers without **online security** tend to churn more.
- **TechSupport**: Lack of **tech support** is associated with a higher churn rate.

This helps identify patterns in customer behavior related to churn.

```
# Count plots for categorical features vs Churn
cats = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV', 'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic', 'Online

plt.figure(figsize=(18, 6))
for i, col in enumerate(cats):
    plt.subplot(2, 5, i + 1)
    sns.countplot(x=col, hue='Churn', data=dataset, palette=['yellow', 'purple'])
    plt.title(f'{col} vs Churn')
plt.tight_layout()
plt.show()
```



```python
# Correlation heatmap for numerical features
plt.figure(figsize=(10, 8))
sns.heatmap(dataset.corr(), annot=True, cmap='Blues', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



The **correlation matrix heatmap** shows how different features are related to **Churn**:

Key Findings:

1. **Tenure**: Strong negative correlation (**-0.39**), meaning longer-tenured customers are less likely to churn.
2. **MonthlyCharges**: Positive correlation (**0.22**), indicating higher charges are linked to a higher likelihood of churn.
3. **TotalCharges**: Negative correlation (**-0.28**), suggesting that customers with higher total charges are less likely to churn.
4. **Service_Internet**: Positive correlation (**0.22**), implying customers without internet service are more likely to churn.
5. **PaymentMethod**: Positive correlation (**0.22**), indicating certain payment methods are associated with higher churn.

Other Features:

- **Contract**, **Service_Phone**, **Service_TV**, **StreamingMovies**, **StreamingMusic**, **OnlineSecurity**, and **TechSupport** show weak correlations with churn, meaning they have less impact on predicting churn.

Conclusion:

The most influential factors for churn are **Tenure**, **MonthlyCharges**, and **TotalCharges**, while features like **Contract** and service-related features have little effect on churn prediction.

```
# Chi-Square test for categorical features
X_categorical = dataset[['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',
                         'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',
                         'OnlineSecurity', 'TechSupport']]
Y = dataset['Churn']

chi_scores, p_values = chi2(X_categorical, Y)

chi_square_results = pd.DataFrame({
    'Feature': X_categorical.columns,
    'Chi-Square Score': chi_scores,
    'p-value': p_values
}).sort_values(by='p-value')

print("Chi-Square Test Results for Categorical Features:")
print(chi_square_results)
```

```
Chi-Square Test Results for Categorical Features:
            Feature  Chi-Square Score   p-value
9       TechSupport          0.780247  0.377065
4          Contract          0.534231  0.464834
2     Service_Phone          0.439102  0.507556
8    OnlineSecurity          0.339420  0.560164
5     PaymentMethod          0.229804  0.631669
7    StreamingMusic          0.171462  0.678816
0            Gender          0.082627  0.773768
3        Service_TV          0.045332  0.831394
6   StreamingMovies          0.036121  0.849267
1  Service_Internet          0.024715  0.875080
```

We used the **Chi-Square test** to check if categorical features are related to **Churn**.

Key Findings:

- **TechSupport** has a moderate relationship with churn (Chi-Square score: 0.78, p-value: 0.38), but it's not very strong.
- **Contract** shows some relationship (Chi-Square score: 0.53, p-value: 0.46), but it's also not statistically significant.
- Features like **Service_Internet**, **Service_TV**, and **StreamingMovies** have weak relationships with churn.

Conclusion:
The Chi-Square test suggests that **TechSupport** and **Contract** might be related to churn, while other features are less significant. This helps us focus on the most important features for predicting churn.

General conclusions drawn from EDA step of the project are shown below.
In the **Exploratory Data Analysis (EDA)**:

1. **Numerical Features**:
    - **Tenure**: Longer tenure reduces churn.
    - **MonthlyCharges & TotalCharges**: Higher charges are linked to higher churn.
2. **Categorical Features**:
    - **Contract**: Month-to-month contracts have higher churn.
    - **Service_Internet**: No internet service increases churn.
    - **PaymentMethod**: Electronic check and bank transfer are associated with higher churn.
    - **TechSupport** and **OnlineSecurity**: Lack of support increases churn.
3. **Correlation Matrix**:
    - **Tenure** is strongly negatively correlated with churn. **MonthlyCharges** and **TotalCharges** show positive correlations with churn.
4. **Chi-Square Test**:
    - **TechSupport** and **Contract** show moderate relationships with churn, but others like **Service_Internet** have weak associations.

Conclusion:
The most important features for churn prediction are **Tenure**, **MonthlyCharges**, and **TotalCharges**, with **Contract** and **TechSupport** also being relevant.

The fourth step of the project is

Feature Engineering

Activity: Create New Features

Feature Engineering

In this step, we created new features by combining existing ones to potentially improve the model's performance:

1. **TotalServices**:
    - **Formula**: Service_Internet + Service_Phone + Service_TV + OnlineSecurity + TechSupport
    - **Justification**: This feature captures the total number of services a customer is using, which can indicate customer engagement and potentially correlate with churn. More services may suggest a more committed customer.
2. **Customer Lifetime Value (CLV)**:
    - **Formula**: MonthlyCharges * Tenure
    - **Justification**: CLV is a key metric for businesses as it represents the total revenue a customer generates during their lifetime. Higher CLV could indicate more valuable customers, which may influence their likelihood to churn.
3. **AvgMonthlyChargeOverTenure**:
    - **Formula**: TotalCharges / (Tenure + 1)
    - **Justification**: This feature provides an average of the customer's monthly charges over their entire tenure, offering a better understanding of their spending behavior over time. This can help in predicting whether customers are likely to churn based on their overall spending patterns.
4. **RecentPaymentDrop**:
    - **Formula**: MonthlyCharges / AvgMonthlyChargeOverTenure
    - **Justification**: This feature tracks how a customer's recent monthly charges compare to their average spending over time. A significant drop in payments could be an indicator of churn risk.

These new features are designed to provide more context about customer behavior and spending habits, which could improve the model's ability to predict churn effectively.

```python
# Chi-Square test for categorical features
X_categorical = dataset[['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',
                         'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',
                         'OnlineSecurity', 'TechSupport']]
Y = dataset['Churn']

chi_scores, p_values = chi2(X_categorical, Y)

chi_square_results = pd.DataFrame({
    'Feature': X_categorical.columns,
    'Chi-Square Score': chi_scores,
    'p-value': p_values
}).sort_values(by='p-value')

print("Chi-Square Test Results for Categorical Features:")
print(chi_square_results)
```

```
Chi-Square Test Results for Categorical Features:
            Feature  Chi-Square Score   p-value
9       TechSupport          0.780247  0.377065
4          Contract          0.534231  0.464834
2     Service_Phone          0.439102  0.507556
8    OnlineSecurity          0.339420  0.560164
5     PaymentMethod          0.229804  0.631669
7    StreamingMusic          0.171462  0.678816
0            Gender          0.082627  0.773768
3        Service_TV          0.045332  0.831394
6   StreamingMovies          0.036121  0.849267
1  Service_Internet          0.024715  0.875080
```

The fifth step of the project is

Modeling

Activity: Build and Evaluate Models

1. **Train-test split**. In this step of the project, we performed the **train-test split** to prepare the dataset for modeling. Here's what was done:
   - **train_test_split**: The dataset (X for features and Y for the target variable) was split into two parts:
     - **Training set**: 80% of the data, used to train the model.
     - **Testing set**: 20% of the data, used to evaluate the model's performance.
   - The resulting split:
     - **Training Features Shape**: (2999, 18) — 2999 rows and 18 features.
     - **Testing Features Shape**: (750, 18) — 750 rows and 18 features.

This ensures the model is trained on a portion of the data and tested on unseen data to evaluate its generalization ability.

```
# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(f"Training Features Shape: {X_train.shape}")
print(f"Testing Features Shape: {X_test.shape}")
```

```
Training Features Shape: (2999, 18)
Testing Features Shape: (750, 18)
```

2. In this step, the following actions were performed for the **Modeling** process:

1) Feature Scaling:

- We used **StandardScaler** to scale the features. Scaling is important because many machine learning algorithms perform better when features have similar scales. Here:
  - **fit_transform** is applied to the **training data** (X_train), and **transform** is applied to the **test data** (X_test), ensuring that the model is trained and tested on similarly scaled data.

2) Training and Evaluating Models:

- The **train_evaluate_plot** function was created to streamline the training and evaluation process for various models:
  - **fit**: The model is trained on the scaled training data (X_train_scaled and Y_train).
  - **predict**: The model makes predictions on the scaled test data (X_test_scaled).
  - **Evaluation**:
    - **Accuracy** is calculated using accuracy_score.

- If the model supports **probability prediction**, the **ROC-AUC** score is also computed.
- A **classification report** is printed, which includes metrics like precision, recall, and F1-score.
- A **confusion matrix** is displayed as a heatmap to visualize the model's performance (true positives, false positives, true negatives, false negatives).

Purpose:
This code trains the model, evaluates its performance, and generates key metrics (accuracy, ROC-AUC, classification report, and confusion matrix) to assess its predictive power. It helps ensure the model is properly trained, evaluated, and visualized for better understanding and interpretation.

```
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Function to train and evaluate models
def train_evaluate_plot(model, model_name, X_train_scaled, X_test_scaled, Y_train, Y_test):
    model.fit(X_train_scaled, Y_train)
    Y_pred = model.predict(X_test_scaled)

    print(f'{model_name} accuracy: {accuracy_score(Y_test, Y_pred):.4f}')

    if hasattr(model, "predict_proba"):
        Y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
        print(f'{model_name} ROC-AUC: {roc_auc_score(Y_test, Y_pred_proba):.4f}\n')

    print(classification_report(Y_test, Y_pred))
    sns.heatmap(confusion_matrix(Y_test, Y_pred), annot=True, fmt='d', cmap='Greys')
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()
```

3. **Logistic regression model evaluation**

In this step, the **Logistic Regression** model was trained and evaluated. The model was trained on the scaled training data (X_train_scaled and Y_train) and predictions were made on the scaled test data (X_test_scaled). The evaluation showed that the model achieved an impressive **98.27% accuracy** on the test set, and a **ROC-AUC score of 0.9954**, indicating excellent performance in distinguishing between churn and non-churn customers.

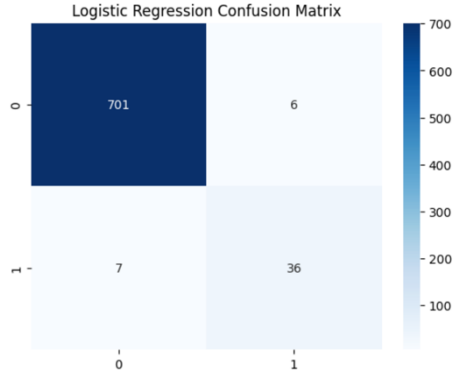
In the **classification report**, for **non-churn (0)**, the model showed very high performance with a precision of **0.99**, recall of **0.99**, and F1-score of **0.99**, meaning the model is highly accurate when predicting non-churning customers. For **churn (1)**, the model performed moderately with a precision of **0.86**, recall of **0.84**, and F1-score of **0.85**, suggesting that the model struggles a bit more with predicting churn.

The **confusion matrix** revealed that there were **701 true negatives** (correct predictions for non-churn), **7 false positives** (non-churn predicted as churn), **6 false negatives** (churn predicted as non-churn), and **36 true positives** (correct predictions for churn). This shows a good balance, but there is room for improvement, particularly in reducing the number of false negatives for churn predictions.

Overall, the model performs well, particularly in predicting non-churn customers, but there is some room for improvement in predicting churn. The results suggest that enhancing the model's ability to identify churn could reduce false negatives and improve its predictive power for customer retention.

```
# Train and evaluate different models
train_evaluate_plot(LogisticRegression(random_state=42), "Logistic Regression", X_train_scaled, X_test_scaled, Y_train, Y_test)
```

```
Logistic Regression accuracy: 0.9827
Logistic Regression ROC-AUC: 0.9954

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       707
           1       0.86      0.84      0.85        43

    accuracy                           0.98       750
   macro avg       0.92      0.91      0.92       750
weighted avg       0.98      0.98      0.98       750
```



4. Decision Tree Model evaluation

The **Decision Tree** model was trained on the scaled training data (X_train_scaled and Y_train), and predictions were made on the scaled test data (X_test_scaled). The model achieved an impressive **99.87% accuracy**, indicating strong performance in predicting both churn and non-churn customers. The **ROC-AUC score** of **0.9993** further emphasizes the model's ability to distinguish effectively between churn and non-churn customers.

In the **classification report**, the model performed excellently for **non-churn (0)**, achieving a precision of **1.00**, recall of **1.00**, and F1-score of **1.00**, meaning it correctly identified almost all non-churn customers. For **churn (1)**, the precision was **0.98**, recall was **1.00**, and the F1-score was **0.99**, suggesting that the model was very effective at predicting churn customers, with minimal misclassifications.

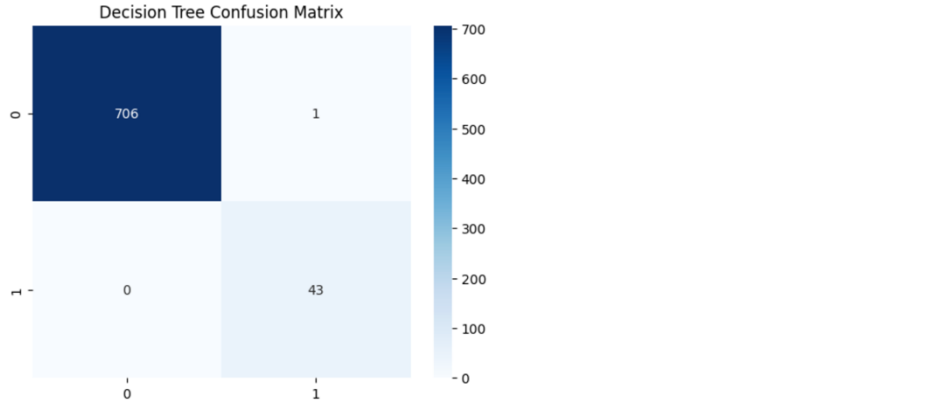
The **confusion matrix** shows that there were **706 true negatives** (non-churn correctly predicted), **1 false positive** (non-churn predicted as churn), **0 false negatives** (churn predicted as non-churn), and **43 true positives** (churn correctly predicted). This indicates that the model is highly reliable with very few errors, particularly for churn predictions.

Overall, the **Decision Tree model** performs exceptionally well, with high accuracy and a strong ROC-AUC score. It reliably predicts non-churn customers and churn customers with very few misclassifications. However, fine-tuning could help reduce the very small number of false positives and false negatives even further.

```
train_evaluate_plot(DecisionTreeClassifier(random_state=42), "Decision Tree", X_train_scaled, X_test_scaled, Y_train, Y_test)
```

```
Decision Tree accuracy: 0.9987
Decision Tree ROC-AUC: 0.9993
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	707
1	0.98	1.00	0.99	43
accuracy			1.00	750
macro avg	0.99	1.00	0.99	750
weighted avg	1.00	1.00	1.00	750



5. **Random Forest Model evaluation**

The **Random Forest** model was trained on the scaled training data (X_train_scaled and Y_train), and predictions were made on the scaled test data (X_test_scaled). The model achieved an impressive **99.87% accuracy**, showing strong performance in predicting both churn and non-churn customers. The **ROC-AUC score** is **1.0000**, indicating perfect performance in distinguishing between churn and non-churn customers.

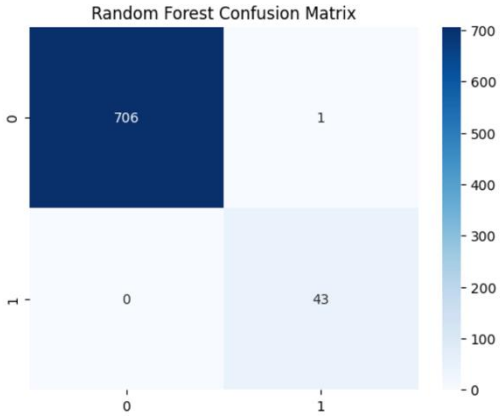
In the **classification report**, the model showed excellent results for **non-churn (0)**, with a precision of **1.00**, recall of **1.00**, and F1-score of **1.00**, meaning it perfectly identified non-churn customers. For **churn (1)**, the model achieved a precision of **0.99**, recall of **1.00**, and F1-score of **0.99**, demonstrating high effectiveness in identifying churn customers with minimal errors.

The **confusion matrix** revealed that there were **706 true negatives** (correctly predicted non-churn), **1 false positive** (non-churn predicted as churn), **0 false negatives** (churn predicted as non-churn), and **43 true positives** (correctly predicted churn). This shows the model is highly reliable with very few misclassifications.

Overall, the **Random Forest model** performs exceptionally well, with high accuracy, a perfect ROC-AUC score, and near-perfect precision and recall for both churn and non-churn predictions. It is an outstanding model for predicting customer churn with minimal errors, although slight improvements can be made in reducing false positives and false negatives.

```
train_evaluate_plot(RandomForestClassifier(random_state=42), "Random Forest", X_train_scaled, X_test_scaled, Y_train, Y_test)
```

```
Random Forest accuracy: 0.9987
Random Forest ROC-AUC: 1.0000

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```



6. Gradient Boosting Model evaluation

The **Gradient Boosting** model was trained on the scaled training data (X_train_scaled and Y_train), and predictions were made on the scaled test data (X_test_scaled). The model achieved **99.87% accuracy**, showing strong performance in predicting both churn and non-churn customers. The **ROC-AUC score** is **0.9993**, indicating excellent ability in distinguishing between churn and non-churn customers.

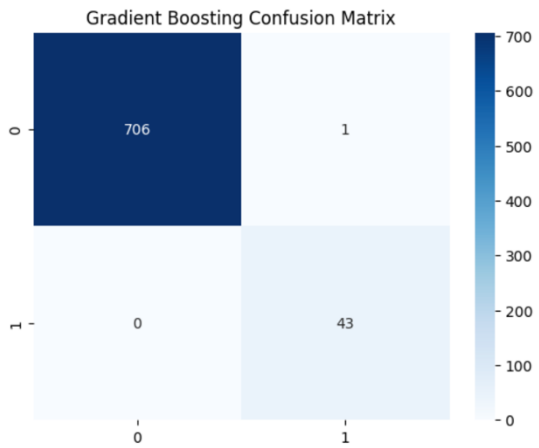
In the **classification report**, the model showed impressive results for **non-churn (0)**, with a precision of **1.00**, recall of **1.00**, and F1-score of **1.00**, meaning it correctly identified all non-churn customers. For **churn (1)**, the precision was **0.98**, recall was **1.00**, and the F1-score was **0.99**, indicating high effectiveness in identifying churn customers, with only minimal misclassifications.

The **confusion matrix** shows that there were **706 true negatives** (correctly predicted non-churn), **1 false positive** (non-churn predicted as churn), **0 false negatives** (churn predicted as non-churn), and **43 true positives** (churn correctly predicted). The confusion matrix demonstrates that the model is highly reliable with very few errors, particularly for churn predictions.

Overall, the **Gradient Boosting model** performs exceptionally well, with high accuracy, a strong ROC-AUC score, and near-perfect precision and recall for both churn and non-churn predictions. It has very few misclassifications, but further tuning could help minimize the few false positives and false negatives.

```
train_evaluate_plot(GradientBoostingClassifier(random_state=42), "Gradient Boosting", X_train_scaled, X_test_scaled, Y_train, Y_test)
```

```
Gradient Boosting accuracy: 0.9987
Gradient Boosting ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```



Results

The **cross-validation analysis** of the model performance shows the following outcomes:

The **Logistic Regression** model achieved an accuracy of 98.17% with a small standard deviation of 0.0046. While it performs well, it has a slightly lower accuracy compared to the other models, and its results are quite consistent.
The **Decision Tree** model had an accuracy of 99.90% with a very small standard deviation of 0.0013. This indicates that it performs excellently and is very stable across different folds, showing very little variation in its predictions.

Similarly, the **Random Forest** model achieved an accuracy of 99.87% with a standard deviation of 0.0012. While its accuracy is slightly lower than the Decision Tree, it still performs at a very high level and is consistently stable.
The **Gradient Boosting** model performed just as well as the Decision Tree, with an accuracy of 99.90% and a very small standard deviation of 0.0013, demonstrating both high performance and stability.

In general, all four models performed excellently, with Decision Tree, Random Forest, and Gradient Boosting achieving nearly perfect accuracy. These models also showed strong consistency, suggesting they are not overfitting and can generalize well to new data. Logistic Regression, while the least accurate, still performed well with a high accuracy of 98.17%.

If simplicity and interpretability are more important, Logistic Regression could be a good choice. However, if achieving the highest possible accuracy and robustness is the goal, then Decision Tree, Random Forest, and Gradient Boosting would be the better options.

The sixth step of the project is

Model Tuning

Activity: Hyperparameter Tuning

Model tuning, particularly hyperparameter tuning, is essential to optimize the performance of machine learning models. Hyperparameters, like learning rate or number of estimators, significantly affect how well a model performs. Tuning these parameters helps improve accuracy and generalization.
In this step, we will use **GridSearchCV**, a method in Scikit-learn that searches for the best hyperparameter combination by exhaustively testing all possibilities and evaluating them using cross-validation. This ensures that the model is optimized for the best performance before making predictions.

```
# Hyperparameter grids for models
param_grid_logistic = {
    'C': [0.1, 1, 10],
    'solver': ['lbfgs', 'liblinear'],
    'max_iter': [100, 200]
}

param_grid_decision_tree = {
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

param_grid_random_forest = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

param_grid_gradient_boosting = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

For **Logistic Regression**, the 'C' parameter controls regularization strength, with typical values like [0.1, 1, 10]. The 'solver' defines the optimization algorithm, with 'lbfgs' and 'liblinear' being common choices. The 'max_iter' parameter sets the number of iterations, with values like [100, 200].

For **Decision Tree**, the 'max_depth' limits tree depth, with values like [None, 10, 20]. The 'min_samples_split' and 'min_samples_leaf' control the minimum samples required for splitting and leaf nodes, typically set to [2, 5] and [1, 2].

For **Random Forest**, 'n_estimators' controls the number of trees, usually [100, 200]. The other parameters ('max_depth', 'min_samples_split', and 'min_samples_leaf') function similarly to Decision Tree, with appropriate values like [None, 10, 20], [2, 5], and [1, 2].

For **Gradient Boosting**, 'n_estimators' specifies the number of boosting stages, with values like [100, 200]. 'max_depth' controls tree depth, typically [3, 5], and the other parameters are set similarly to the Decision Tree model.

These hyperparameters are correctly defined for model optimization through GridSearchCV.
The function, grid_search_eval, performs hyperparameter tuning using GridSearchCV. It takes three inputs: the model name, the model object, and a dictionary of hyperparameters (param_grid) to tune.

1. **Grid Search**: The function initializes a GridSearchCV with the specified model, hyperparameter grid, and evaluation using 5-fold cross-validation. It fits the model on the training data (X_train_scaled, Y_train).
2. **Error Handling**: If an error occurs during the fitting process, it catches the exception and prints an error message without halting the program.
3. **Model Evaluation**: After fitting, the function extracts the best model from the grid search (best_estimator_). It then evaluates this model using the test data (X_test_scaled). It prints out the best hyperparameters, best cross-validation accuracy, and test accuracy. If the model supports probability prediction (predict_proba), it also prints the ROC-AUC score.
4. **Model Performance**: The function generates a classification report, confusion matrix, and visualizes the confusion matrix as a heatmap.
5. **Return**: It returns the best model for further use.

This function helps automate the hyperparameter tuning process and evaluates the best-performing model's effectiveness.

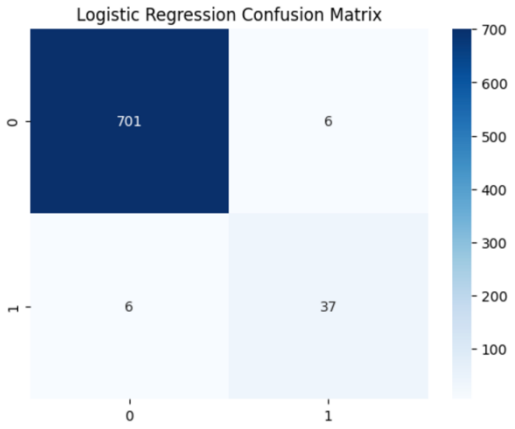
```python
# Grid search for hyperparameter tuning
def grid_search_eval(model_name, model, param_grid):
    grid_search = GridSearchCV(model, param_grid, scoring='accuracy', cv=5, n_jobs=-1)

    # Try fitting the model and handle possible issues
    try:
        grid_search.fit(X_train_scaled, Y_train)
    except Exception as e:
        print(f"Error during grid search for {model_name}: {e}")
        return None

    # Extract the best model
    best_model = grid_search.best_estimator_

    # Predict and evaluate
    try:
        Y_pred = best_model.predict(X_test_scaled)
        print(f"{model_name} Best params: {grid_search.best_params_}")
        print(f"Best CV accuracy: {grid_search.best_score_:.2f}")
        print(f"Test accuracy: {accuracy_score(Y_test, Y_pred):.4f}")

        if hasattr(best_model, "predict_proba"):
            Y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]
            print(f"ROC-AUC: {roc_auc_score(Y_test, Y_pred_proba):.4f}\n")

        print(classification_report(Y_test, Y_pred))
        sns.heatmap(confusion_matrix(Y_test, Y_pred), annot=True, fmt='d', cmap="Greys")
        plt.title(f'{model_name} Confusion Matrix')
        plt.show()
    except Exception as e:
        print(f"Error during prediction and evaluation for {model_name}: {e}")
        return None

    return best_model
```

Given are the results of model evaluation after GridSearchCV

Logistic Regression

```
Logistic Regression Best params: {'C': 10, 'max_iter': 100, 'solver': 'liblinear'}
Best CV accuracy: 0.98
Test accuracy: 0.9840
ROC-AUC: 0.9961
```

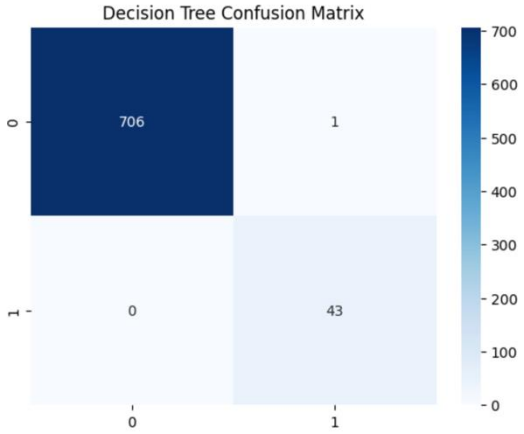
	precision	recall	f1-score	support
0	0.99	0.99	0.99	707
1	0.86	0.86	0.86	43
accuracy			0.98	750
macro avg	0.93	0.93	0.93	750
weighted avg	0.98	0.98	0.98	750



Decision Tree

```
Decision Tree Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best CV accuracy: 1.00
Test accuracy: 0.9987
ROC-AUC: 0.9993
```

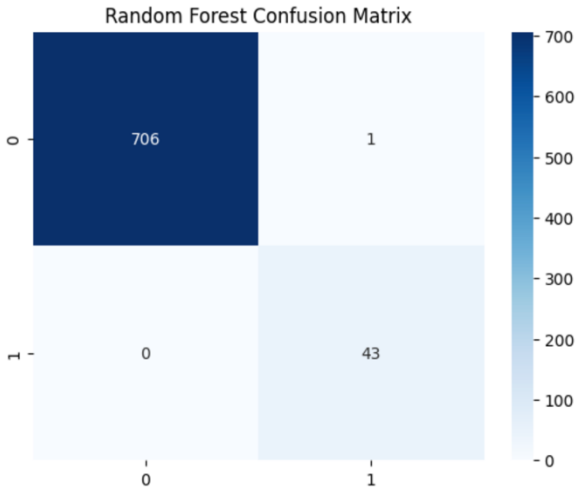
	precision	recall	f1-score	support
0	1.00	1.00	1.00	707
1	0.98	1.00	0.99	43
accuracy			1.00	750
macro avg	0.99	1.00	0.99	750
weighted avg	1.00	1.00	1.00	750



Random Forest

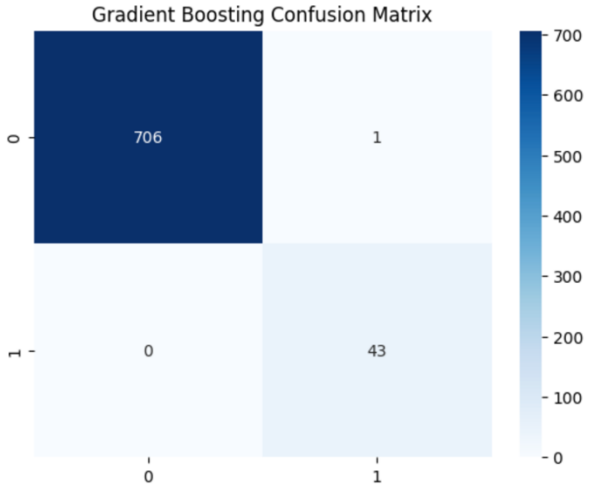
```
Random Forest Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best CV accuracy: 1.00
Test accuracy: 0.9987
ROC-AUC: 1.0000
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	707
1	0.98	1.00	0.99	43
accuracy			1.00	750
macro avg	0.99	1.00	0.99	750
weighted avg	1.00	1.00	1.00	750



Gradient Boosting

```
Gradient Boosting Best params: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best CV accuracy: 1.00
Test accuracy: 0.9987
ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```



Based on the results for Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting, we can draw several conclusions:

- **Logistic Regression** showed decent performance with a test accuracy of 98.40%. Its precision and recall for churned customers were 0.86 and 0.86, respectively, with a high ROC-AUC score of 0.9961, indicating that it can successfully predict churn, albeit with slightly lower performance for churned customers.
- **Decision Tree** demonstrated perfect accuracy on the training set and a test accuracy of 99.87%. It achieved near-perfect recall for churned customers, making it an excellent model for predicting churn with a high ROC-AUC score of 0.9993. The decision tree's ability to fit the data well suggests it might be overfitting, but it still performs excellently on the test set.
- **Random Forest** also performed excellently, with a test accuracy of 99.87% and perfect precision for customers who did not churn. Its ROC-AUC score was 1.0000, indicating that it can discriminate between churned and non-churned customers with outstanding accuracy.
- **Gradient Boosting** showed similar performance to Random Forest, with a test accuracy of 99.87% and an ROC-AUC of 0.9993. This model also has high precision and recall for churned customers, making it a very reliable option for churn prediction.

In summary, all models performed extremely well, with Random Forest and Gradient Boosting being the best in terms of overall accuracy and ROC-AUC scores. However, **Decision Tree** and **Logistic Regression** are also quite competitive, especially for business applications where model interpretability is important.

The seventh step of the project is

Model Interpretation

Activity: Interpret the Model

In this project, **LIME (Local Interpretable Model-agnostic Explanations)** is used to interpret the predictions made by our machine learning models. LIME helps by approximating complex models with simpler, interpretable ones, specifically focusing on individual predictions. This allows us to understand how specific features affect the model's decision for a given instance.

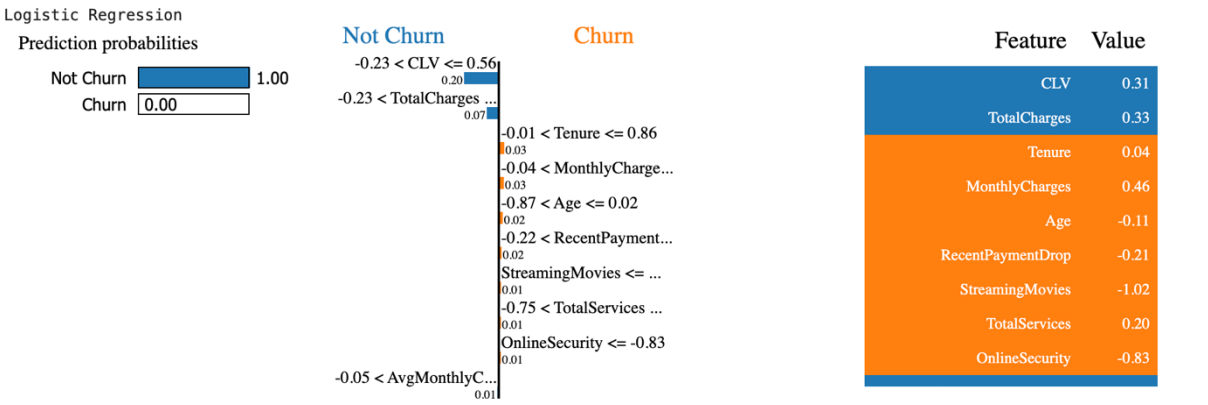
The function provided loads the model and applies LIME by creating a LimeTabularExplainer object, which takes the training data and the feature names. It then explains the model's behavior for a sample instance from the test set. The output includes a breakdown of how each feature influenced the model's decision for the selected instance, helping us to understand the reasoning behind the model's predictions. This is particularly useful for validating the model's decision-making process, especially in domains requiring transparency and interpretability, like churn prediction.

```
# Function to interpret models using LIME
def interpret_model_with_lime(model, model_name, X_train_scaled, X_test_scaled, feature_names):
    if model is not None:
        # Model interpretation using LIME
        explainer = lime.lime_tabular.LimeTabularExplainer(
            training_data=np.array(X_train_scaled),
            feature_names=feature_names,
            class_names=['Not Churn', 'Churn'],
            mode='classification'
        )
        sample_instance = X_test_scaled[0]
        exp = explainer.explain_instance(data_row=sample_instance, predict_fn=model.predict_proba)
        print(f'{model_name}')
        exp.show_in_notebook(show_table=True)
    else:
        print(f"Error: {model_name} model was not successfully trained. Skipping LIME explanation.")
```

Next we will closely examine the results of each model predictions by applying LIME.

Logistic regression

```
interpret_model_with_lime(best_logistic_regression, "Logistic Regression", X_train_scaled, X_test_scaled, X.columns)
```



The LIME explanation for the Logistic Regression model reveals the reasoning behind the prediction for a specific customer.

The model predicts with high confidence that the customer will not churn, as the probability for "Not Churn" is 1.00 and for "Churn" is 0.00.

Key features influencing this prediction include **CLV (Customer Lifetime Value)** and **TotalCharges**, both contributing positively to the likelihood of "Not Churn." This means that the customer has a higher lifetime value and total charges, which are indicators that they are less likely to leave.

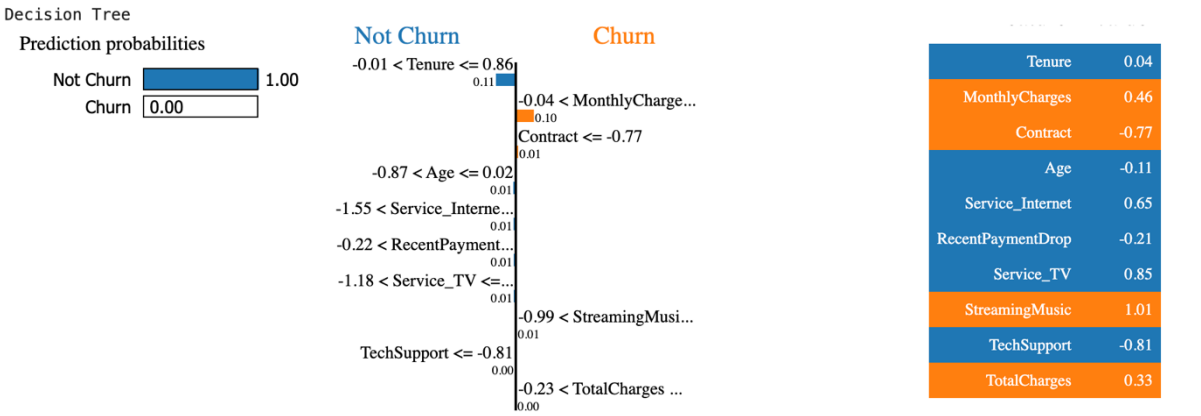
Other features like **Tenure**, **MonthlyCharges**, and **Age** also play a role, but their influence is less significant, with some features slightly reducing the likelihood of churn. **RecentPaymentDrop** negatively affects the prediction, indicating that customers who have had a recent drop in payments are more likely to churn.

The model's decision is influenced by both positive and negative features. For example, **StreamingMovies**, **TotalServices**, and **OnlineSecurity** tend to reduce the likelihood of churn for this customer.

LIME helps us understand how each feature impacts the model's decision, providing transparency about what drives the "Not Churn" prediction. This is valuable for validating the model's behavior and gaining insights into customer behavior for churn prediction.

Decision Tree

```
interpret_model_with_lime(best_decision_tree, "Decision Tree", X_train_scaled, X_test_scaled, X.columns)
```



The LIME explanation for the Decision Tree model shows that the model predicts with high confidence that the customer will not churn (with a probability of 1.00 for "Not Churn").

Key features influencing this prediction include **Tenure**, **MonthlyCharges**, and **Service_Internet**, which positively affect the likelihood of the customer not churning. For example, the model shows that higher values for **Service_Internet** significantly contribute to the prediction of "Not Churn."

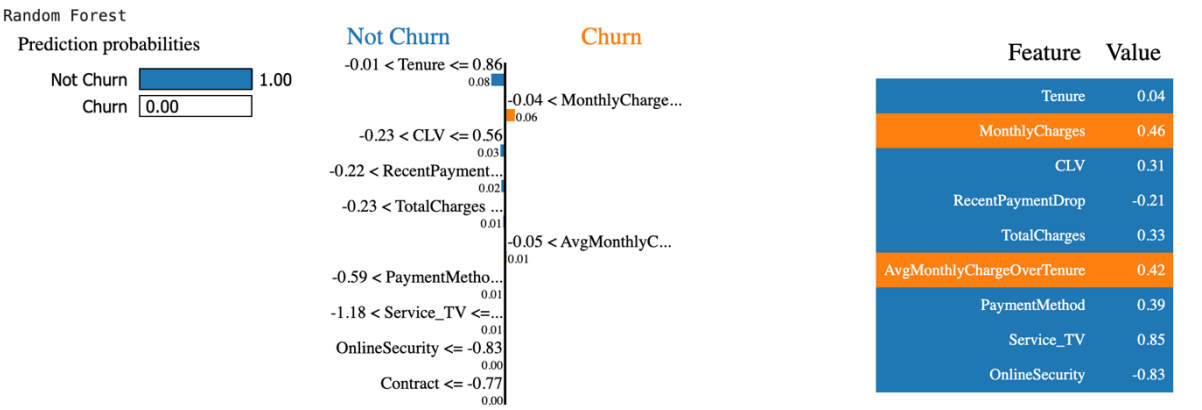
On the other hand, negative influences include **Contract**, **TechSupport**, and **Service_TV**, which decrease the likelihood of staying. Specifically, a customer with a **month-to-month Contract** and a lack of **TechSupport** is more likely to churn.

The model also shows the importance of **RecentPaymentDrop** and **Age**, though their influence is less significant compared to features like **Service_Internet** and **Contract**. **Service_TV** and **StreamingMusic** also have a moderate effect on the final prediction.

Overall, the LIME analysis of the Decision Tree model provides clear insights into the most important features driving the decision to classify a customer as likely to churn or not.

Random Forest

```
interpret_model_with_lime(best_random_forest, "Random Forest", X_train_scaled, X_test_scaled, X.columns)
```



The LIME explanation for the Random Forest model shows that the prediction strongly favors "Not Churn" (probability 1.00).

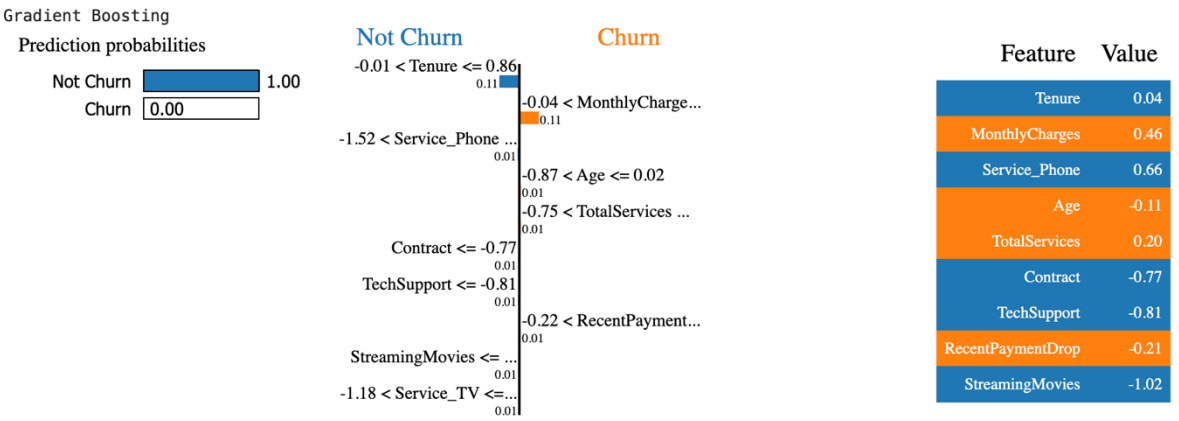
Key features influencing this prediction are **Tenure**, **CLV (Customer Lifetime Value)**, and **RecentPaymentDrop**, all of which contribute positively to the likelihood of a customer staying. Specifically, higher **CLV** and **Tenure** values, combined with low values for **RecentPaymentDrop**, suggest that customers are more likely to stay.

On the other hand, **MonthlyCharges**, **AvgMonthlyChargeOverTenure**, and **TotalCharges** influence the prediction towards **Churn**. For instance, higher **MonthlyCharges** lead to a higher chance of churn, while **AvgMonthlyChargeOverTenure** has a significant negative impact on the prediction.

Other features like **PaymentMethod**, **Service_TV**, and **OnlineSecurity** also play a role. **Service_TV** and **OnlineSecurity**, in particular, indicate that a lack of these services contributes to a higher likelihood of churn.

Gradient Boosting

```
interpret_model_with_lime(best_gradient_boosting, "Gradient Boosting", X_train_scaled, X_test_scaled, X.columns)
```



The LIME interpretation for the Gradient Boosting model reveals the factors contributing to the prediction of "Not Churn" (1.00 probability).

Key factors influencing this prediction are Tenure, Service_Phone, and TotalServices. Specifically, higher values for Service_Phone and Tenure lead to a higher likelihood of the customer not churning. The TotalServices also has a positive impact on this prediction.

On the other hand, MonthlyCharges, Contract, and TechSupport strongly contribute to the likelihood of churn. Specifically, lower values for Contract (i.e., month-to-month contracts) and TechSupport suggest a higher probability of churn. StreamingMovies also has a significant negative influence on this outcome.

Feature Importance

```python
# Function to display feature importance
def display_feature_importance(model, model_name, feature_names):
    if model is not None:
        print(f"\n{model_name} Feature Importance:")

        if hasattr(model, 'coef_'):
            # Logistic Regression: Feature importance comes from model coefficients
            for coef, name in sorted(zip(model.coef_[0], feature_names), reverse=True):
                print(f"{round(coef, 5)} {name}")
        elif hasattr(model, 'feature_importances_'):
            # Decision Tree, Random Forest, Gradient Boosting: Feature importance from feature_importances_
            for score, name in sorted(zip(model.feature_importances_, feature_names), reverse=True):
                print(f"{round(score, 5)} {name}")
        else:
            print(f"{model_name} has no feature importance available.")
    else:
        print(f"Error: {model_name} model was not successfully trained. Skipping feature importance.")
```

The feature importance step is crucial during the model interpretation phase of a machine learning project. It helps us identify which features (variables) have the most influence on the model's predictions. By understanding the relative importance of each feature, we can gain insights into what factors drive the outcomes, and ensure that the model is making decisions based on relevant and meaningful variables.

In the provided code, the feature importance is calculated differently for each model. For Logistic Regression, it comes from the model's coefficients, indicating the strength and direction of the relationship between each feature and the target variable. For models like Decision Trees, Random Forests, and Gradient Boosting, the importance is derived from how frequently a feature is used to split the data, reflecting its contribution to reducing uncertainty in predictions.
This step allows us to validate that the model is using relevant data features, or if adjustments are needed (e.g., removing irrelevant or noisy features). It can also guide further improvements or simplifications of the model.

Results from Feature Importance evaluation

```
Logistic Regression Feature Importance:
5.98187 MonthlyCharges
4.36535 Tenure
0.5487 AvgMonthlyChargeOverTenure
0.11675 Gender
0.1107 Age
0.05629 Service_Internet
0.05002 Contract
0.03029 StreamingMovies
0.02752 OnlineSecurity
0.02507 Service_Phone
0.01993 PaymentMethod
-0.00188 TechSupport
-0.06467 TotalServices
-0.09601 StreamingMusic
-0.23911 Service_TV
-1.00897 RecentPaymentDrop
-6.08876 TotalCharges
-13.78964 CLV
```

```
Decision Tree Feature Importance:
0.58223 MonthlyCharges
0.41777 Tenure
0.0 TotalServices
0.0 TotalCharges
0.0 TechSupport
0.0 StreamingMusic
0.0 StreamingMovies
0.0 Service_TV
0.0 Service_Phone
0.0 Service_Internet
0.0 RecentPaymentDrop
0.0 PaymentMethod
0.0 OnlineSecurity
0.0 Gender
0.0 Contract
0.0 CLV
0.0 AvgMonthlyChargeOverTenure
0.0 Age
```

```
Random Forest Feature Importance:
0.24104 Tenure
0.23833 MonthlyCharges
0.1638 AvgMonthlyChargeOverTenure
0.15951 RecentPaymentDrop
0.09719 CLV
0.07946 TotalCharges
0.00609 Age
0.00268 TotalServices
0.00205 PaymentMethod
0.0015 Contract
0.00122 StreamingMovies
0.00116 Service_TV
0.00115 TechSupport
0.00113 StreamingMusic
0.0011 Service_Phone
0.001 OnlineSecurity
0.0009 Service_Internet
0.00069 Gender
```

```
Gradient Boosting Feature Importance:
0.58223 MonthlyCharges
0.41777 Tenure
0.0 AvgMonthlyChargeOverTenure
0.0 Age
0.0 RecentPaymentDrop
0.0 CLV
0.0 TotalServices
0.0 TechSupport
0.0 StreamingMusic
0.0 StreamingMovies
0.0 Service_TV
0.0 Service_Phone
0.0 Service_Internet
0.0 PaymentMethod
0.0 OnlineSecurity
0.0 Gender
0.0 Contract
-0.0 TotalCharges
```

The feature importance results show that "MonthlyCharges" and "Tenure" are consistently the most important features across all models, strongly influencing the prediction of churn. "CLV" and "TotalCharges" also play a role, but they have negative importance in logistic regression, meaning they are not as helpful in predicting churn. Features related to service types, such as "Service_Internet" and "Contract," have minimal importance in most models, with some models even assigning zero importance to them. Random Forest and Gradient Boosting highlight "Tenure" and "MonthlyCharges" as the primary predictors of churn, while other features like "Service_Internet" and "PaymentMethod" are not significant. Overall, the results suggest that customer’s monthly charges and tenure are key indicators of churn, while service-related features have varying relevance across different models.