University of Messina

Bachelor of Data Analysis

ACADEMIC YEAR - 2023/2024

Machine Learning

(Project report)

Supervisor:
Prof. Giacomo Fiumara

Student:
Yoshan Gunaratna(528176)

Table of Content

Introduction

In day today life business environment, machine learning one of the biggest tool for predicting and managing customer relationships. This technology not only making decision processes but also enables businesses to proactively address challenges such as customer churn. Customer churn, or attrition, occurs when customers or subscribers end their relationship with a company within a specified timeframe. It is a critical measurement that can significantly affect a company's revenue and its future prediction.

Understanding the Dataset

In this project, our primary objective is to analyze and understand the factors influencing customer churn. To begin, we import all necessary libraries which will support the implementation of our analysis.

In order to understand our dataset, we first upload and review it. The dataset consists of 3,749 rows and 17 columns. However, the first two columns represent the indexes of the entities and do not provide meaningful information for our analysis. Therefore, we will remove those columns from our dataset.

```
data=pd.read_csv('dataset_churn.csv')
data.head()
```

```
[ ]  data=data.drop(columns=['Unnamed: 0'])
     data=data.drop(columns=['CustomerID'])
```

Distribution about the data:

In the dataset, there are four numerical features, each with its own statistical distribution.

```
Age                float64
Gender              object
Tenure               int64
Service_Internet    object
Service_Phone       object
Service_TV          object
Contract            object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges       float64
StreamingMovies     object
StreamingMusic      object
OnlineSecurity      object
TechSupport         object
Churn               object
dtype: object
```

Age: The customers' ages range from 18 to 69 years, with an average age of about 44 years.

Tenure: Customers have been with the company for 1 to 71 months, with an average tenure of about 36 months.

MonthlyCharges: Monthly charges vary widely, ranging from $20 to $1179.30, with an average of about $75.84.

TotalCharges: Total charges also vary significantly, ranging from $13.19 to $79951.80, with an average of about $2718.97. This wide range suggests that some customers have been with the company for a long time or have high service usage.

	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

Data Preprocessing

Handling the missing values

The next step is to deal the missing values, as a hint in the statistical summary, the count of non-missing entries in the age column does not correspond with the total number of rows in the dataset. This indicates the presence of missing values.

these missing values can lead to incorrect results because they often represent incomplete information. Handling these missing values ensures that our dataset more accurate and ready to analysis. So in our dataset there are 3 features with the missing values Age(int), Service_Internet (categorical), PaymentMethod(categorical).

```
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
```

We use the median value to fill in missing entries in the 'Age', The reason why I use it, making a strong measure for central tendency and ensure that distribution does not significantly affect the original distribution of the age data.

```
data['Age'].fillna(data['Age'].median(), inplace=True)
```

We use the mode value to fill missing values in the 'PaymentMethod' and 'Service_Internet' columns. So using this most frequently occurring value and this cannot affect the original distribution.

```
data['PaymentMethod'].fillna(data['PaymentMethod'].mode()[0], inplace=True)

data['Service_Internet'].fillna(data['Service_Internet'].mode()[0], inplace=True)
```

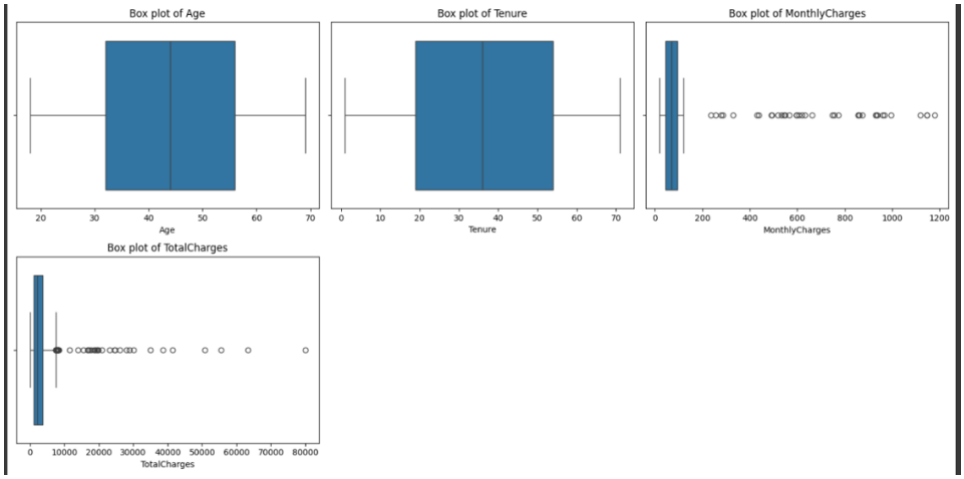
Label Encoding for the categorical features

In our dataset, several features are categorical, represented as strings. Most machine learning algorithms require input to be numeric, which means we need to convert categorical string data into a numeric values using LabelEncoder() function. For Example, the categorical value “yes” to 1 and “No” to 0.

```
categorical_feature=data.select_dtypes(include=['object'])
for feature in categorical_feature.columns:
    data[feature]=LabelEncoder().fit_transform(data[feature])
data.head()
```

Handling the Outliers

Outliers are datapoints that deviate significantly from the rest of the dataset and can have a substantial impact on analysis.



In the above box-plot, we can see there are outliers in two features —which are ‘MonthlyCharges’ and ‘TotalCharges’. And we are going to handle it with the technique called inter quartile range.

Then I’m writing a function to detect the outliers, at very beginning calculates the first and third quartiles of the feature, which represent the 25th and 75th percentiles, respectively. After I will check for the outlier bounds any data point falling outside 1.5 times the IQR below the first quartile or above the third quartile is considered as an outlier. Finally, I’m checking for the about the indices of these outliers.

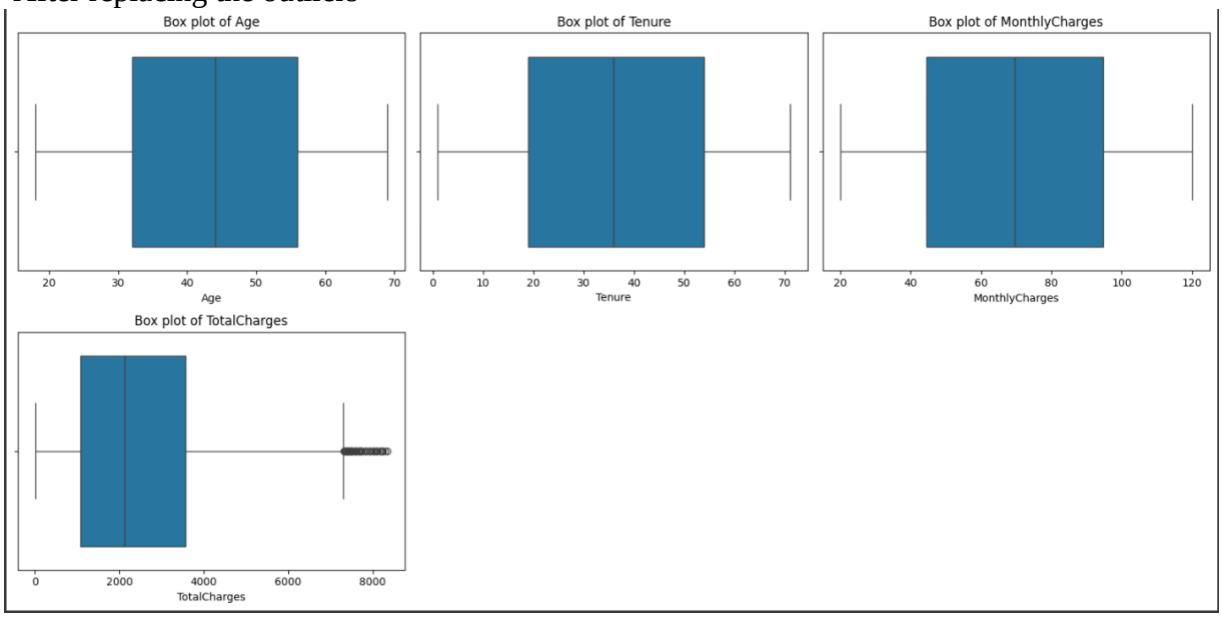
```
def iqr_outliers(dataset,feature_name, multiplier=2):

    Q1 = dataset[feature_name].quantile(0.25)
    Q3 = dataset[feature_name].quantile(0.75)

    IQR = Q3 - Q1

    lwr_bound = Q1 - multiplier * IQR
    upp_bound = Q3 + multiplier * IQR

    outliers_indices= dataset.index[np.logical_or(dataset[feature_name]<lwr_bound,
                                dataset[feature_name]>upp_bound)]
    return outliers_indices

outliers_detected={}
for feature in numerical_feature:
    outliers = iqr_outliers(data,feature)
    outliers_detected[feature] = outliers
    print(outliers)
```

After that I’m replacing these outliers with the median values, now I have list of the integer indices corresponded outlier. By using iloc, ensure that the

replacement operation for outliers is performed exactly at the locations that contain outlier values. It is crucial for ensuring that only the outliers are replaced, and all other data points are remained unchanged. So, after these outliers indices are replacing with the median value.

```
for i in numerical_feature:
    data[i] = data[i].replace(data[i].iloc[outliers_detected[i]].values,data[i].median())
```

After replacing the outliers



EDA (Exploratory Data Analysis)

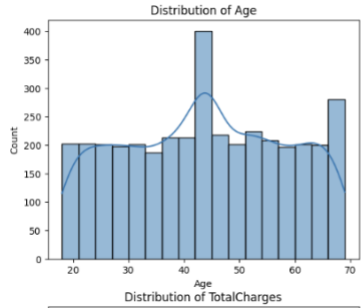
Exploratory Data Analysis (EDA) is a foundational to understand the dataset, to see spotting anomalies and checking assumptions. It helps data scientists understand the distribution of the data.

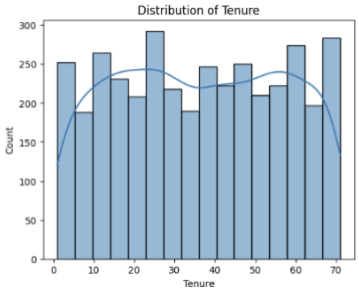
Overview of the Dataset

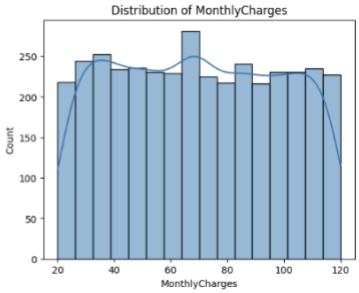
Our dataset is consists of 3749 entries with 17 columns, excluding 'Unnamed: 0' and 'CustomerID', which are irrelevant for analysis.

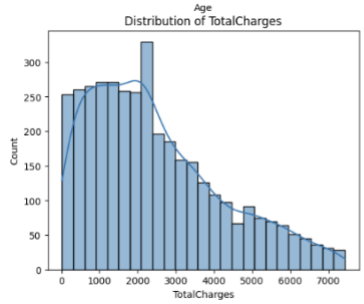
Histogram of Numerical Features

First, we are going to make a distribution graph for the numerical features using histogram.









Age:

The distribution of age is divided into 2 parts from the ages between 40s and late 50s. These mean most of the customers are Kind of senior citizens. And this can guide to improve services to full feel the distinct needs of these age groups.

Tenure:

Tenure shows a somehow uniform distribution with a small peak in the 30–40-month range. This pattern indicates that while many customers stay with the service for a moderate duration, fewer remain for very long periods and other for short period.

MonthlyCharges:

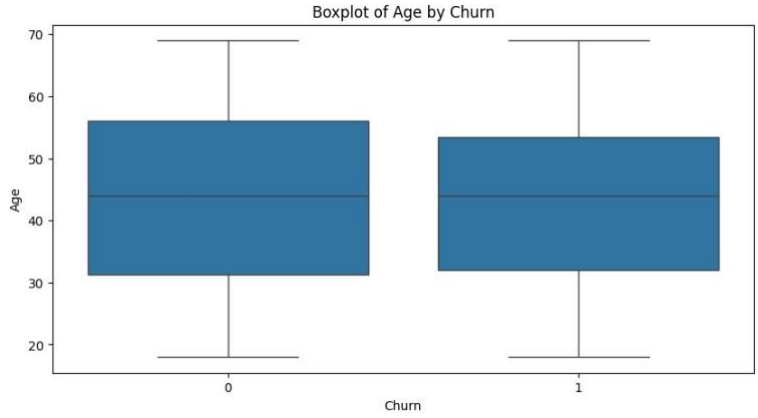
The distribution of monthly charges is uniform, indicating that our customer base utilises a diverse range of service plans.

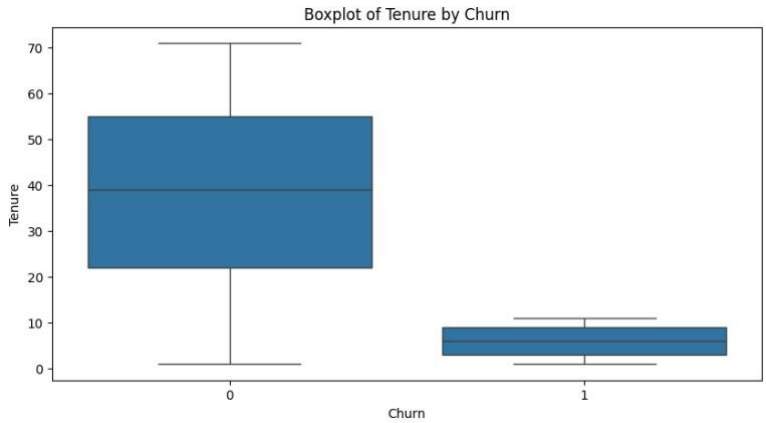
TotalCharges:

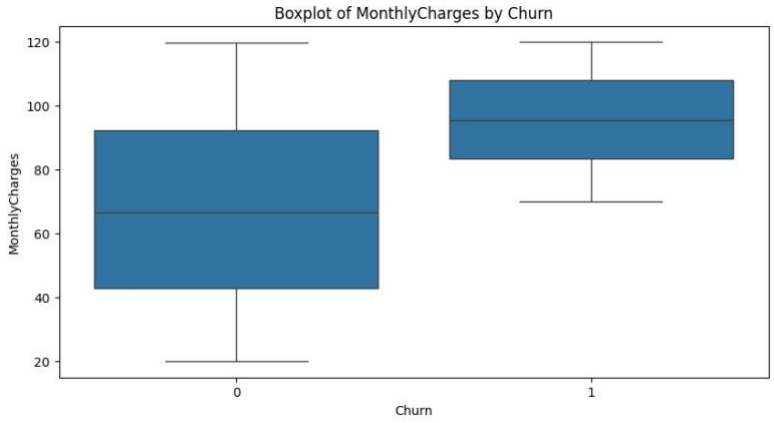
Total charges are right skewed, with many customers have total charges less than 3000. This skewness points improvement in customer retention.

Relationship Between Features and Target Variable

Now here is the relation between a numerical feature and the target feature which is Churn. This relation is visualized with the box-plot and the Scater Plot.





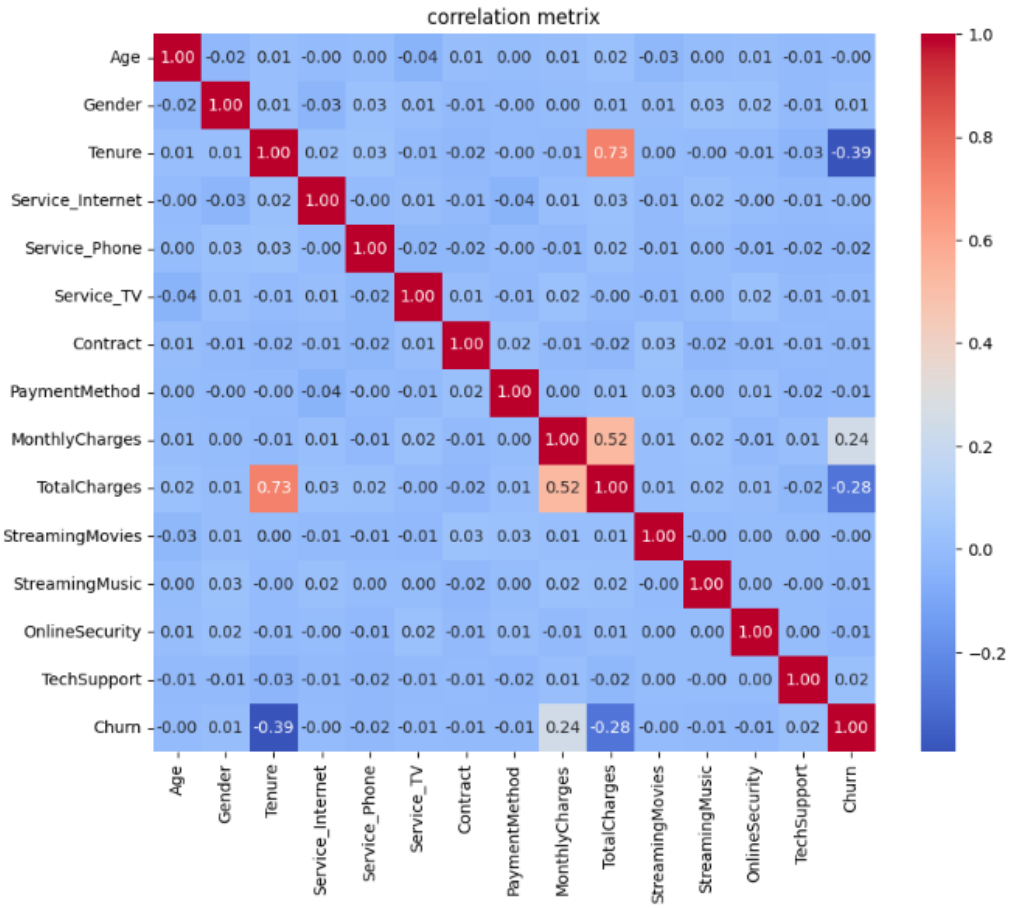


Here, we can see some relationship and trends between the feature and target. For example, the box illustrates that customers with higher monthly charges are more likely to churn. This trend highlights 'MonthlyCharges' as a significant predictor of churn.

Correlation Analysis

The heatmap of the correlation matrix visualizes the relationships between features. For example, 'MonthlyCharges' and 'TotalCharges' show a strong positive correlation, indicating that customers with higher monthly charges tend to accumulate higher total charges over time. This heatmap also helps in identifying multicollinearity.

By correlation matrix, we can also look the percentage of dependencies of the features on other features. Because, correlation coefficient ranges between -1 and 1. Where 1 mean strongly positive correlation and -1 means strongly negative correlation.



Feature Engineering

Feature Engineering I'm are introducing new features with modified existing features. In this project, the main focus on creating features that will help to understand customer behaviour and improve the predictability of churn.

MonthlyCharges_per_Tenure

This feature is calculating average monthly charge per tenure by dividing 'MonthlyCharges' by 'Tenure' plus one to avoid division by zero in cases where tenure is zero.

This feature normalizes monthly charges over the tenure length, which will provide whether long-term customers tend to pay more or less per month compared to newer customers, potentially influencing their decision to churn.

TotalCharges_per_Tenure

this feature represents the total charges a customer has paid over their tenure, normalized by the tenure length, which will allowing the model to assess whether higher overall payments correlate with churn risk.

TotalServices

The total number of services used by a customer might impact to churn. Customers with more services will be having less probability churn.

Churn_high_probability

Then we are giving binary flag indicating whether a customer has a high probability of churning, set to '1' for customers with a tenure of 20 months or less, and '0' more then wo months.

The reason behind this feature is that newer customers are more likely to churn than those who have been with the company for a longer period. It provides a direct way for models to understand tenure information in predicting churn.

```
# Function to engineer new features
def feature_engineering(data):

    data['MonthlyCharges_per_Tenure'] = data['MonthlyCharges'] / (data['Tenure'] + 1)

    data['TotalCharges_per_Tenure'] = data['TotalCharges'] / (data['Tenure'] + 1)

    # Total number of services
    data['TotalServices'] = (data['Service_Internet'] + data['Service_Phone'] + data['Service_TV'] +
                             data['StreamingMovies'] + data['StreamingMusic'] + data['OnlineSecurity'] + data['TechSupport'])

    # Flag for high probability of Churn status

    data['churn_high_probability'] = (data['Tenure'] <= 20).astype(int)

    return data


data = feature_engineering(data)

data.head()
```

Modelling and Evaluation

In this part I'm splitting data in to features and target variable. Which represented by X (features) and y (target variable). After that there is division of the dataset into training and testing subsets in order to model to train. After the model has been trained, it is then evaluated using the testing data.

```
X = data.drop('Churn', axis=1)
y = data['Churn']

X.head()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

After that we are scaling our data with the range of 0 to 1 to normalize the features, so they contribute equally to the mode this is done by using StandardScaler(). First, I'm scaling the training data (X_train) by calculating the mean and standard deviation of each feature.

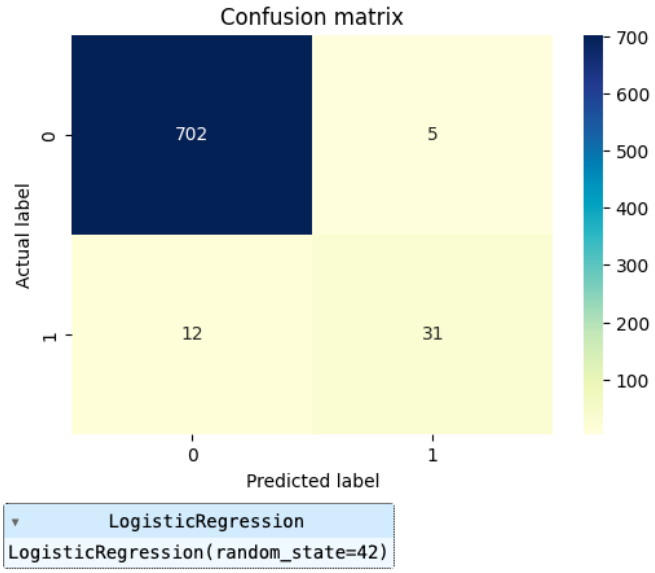
Then test data (X_test) is scaled using the same mean and standard deviation calculated from the training data.

Model selection

Then I used one function to work on the training and evaluation of all three models and later calling it for each model.

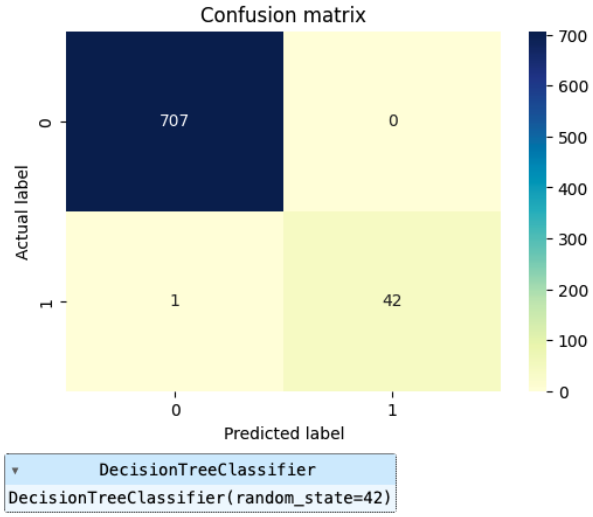
```python
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):

    model.fit(X_train_scaled, y_train)

    # Prediction on test set
    y_pred = model.predict(X_test)

    # display  accuracy
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    print(f'\nAccuracy: {accuracy}')
    print(f'Classification Report:\n{report}')


    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(pd.DataFrame(conf_matrix), annot=True, cmap="YlGnBu", fmt='g')
    plt.title('Confusion matrix')
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    return model
```

1. **Logistic Regression Classifier** with accuracy level of 0.97.



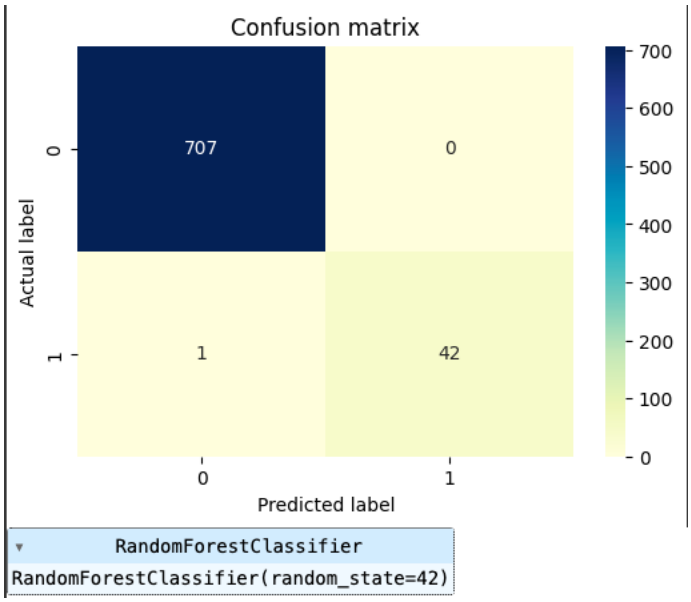
- **Logistic Regression**: it was the least performance among all models and the confusion matrix is presented as follows

2. **Decision Tree Classifier** with accuracy level of 0.99



- **Decision Tree**: has better performance and the confusion matrix is presented as follows as well

3. **Random Forest Classifier** with accuracy level of 0.99



- **Random Forest**: provided same performance as Decision Tree, the confusion matrix is presented below

At the end we can end up with that the Decision Tree and the Random Forest models showed the best performance across other metric, making it the most suitable model for predicting customer churn in this dataset.

Model Tuning

Here, I'm using the GridSearchCV() function on my models to explore a range of hyperparameters and find the best combination. For this scenario, this step is not strictly necessary, as I am already achieving 100% accuracy. However, I thought it would be better to implement it for the other models with less than 100% accuracy just to see if this step affects the training and evaluation of the models.

I'm achieving Hyperparameter Tuning with Grid Search, so, GridSearchCV() is a function from the scikit-learn library in Python, used to tune hyperparameters of a model to find the best possible combination. It tests a specified range of values for each hyperparameter using cross-validation in order to optimizing model performance.

```python
def evaluate_models(models, param_grids, X_train, y_train, X_test, y_test):
    grid_search = GridSearchCV(estimator=models, param_grid=param_grids, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)


    # Get the best model from GridSearchCV
    best_model = grid_search.best_estimator_

    # Predict (variable) best_model: Any | type
    y_pred = best_model.predict(X_test)
    # Best parameters
    best_parameters = grid_search.best_params_
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    print(f'\nAccuracy: {accuracy}')
    print(f'Classification Report:\n{report}')
    print("Best Parameters:", best_parameters)



    # Evaluation
    conf_matrix = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(6, 4))
    sns.heatmap(pd.DataFrame(conf_matrix), annot=True, cmap="YlGnBu", fmt='g')
    plt.title('Confusion matrix')
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()


    return best_model
```

The following hyperparameters were tuned for these models:

- n_estimators
- max_depth
- min_samples_split

- min_samples_leaf

Hyperparameters

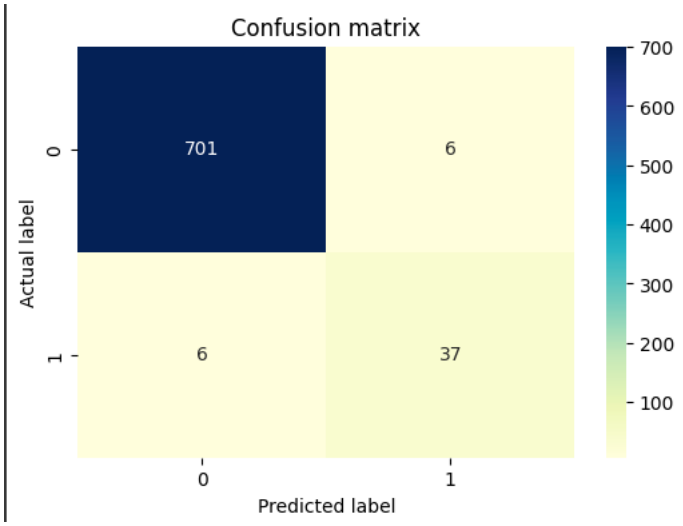
```
Logistic_regression = {
    'C': [0.1, 1, 10],
    'solver': ['liblinear']
}
```

```
Random_Forest= {
        'n_estimators': [50,100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 10, 20],
        'min_samples_leaf': [1, 2, 4],
},
```

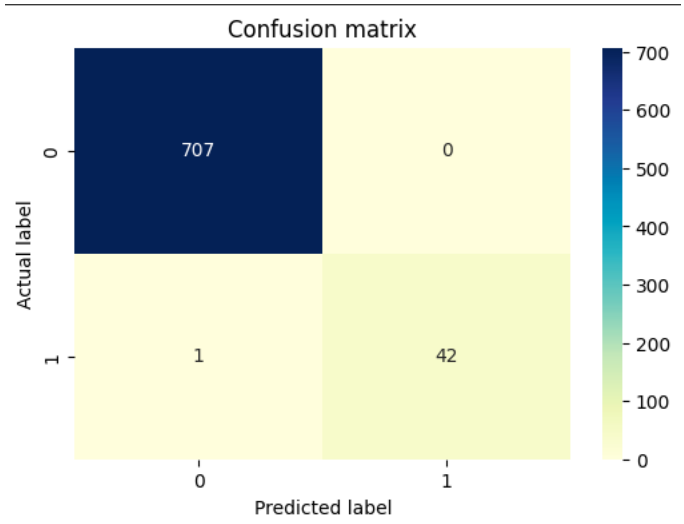
```
Decision_Tree={
        'max_depth': [None, 10, 20, 30, 40],
        'min_samples_split': [2, 10, 20],
        'min_samples_leaf': [1, 2, 4],
}
```

After performing hyperparameter tuning on our three models, we can see that Logistic Regression has shown performance improvement, while the other two models remain unchanged.

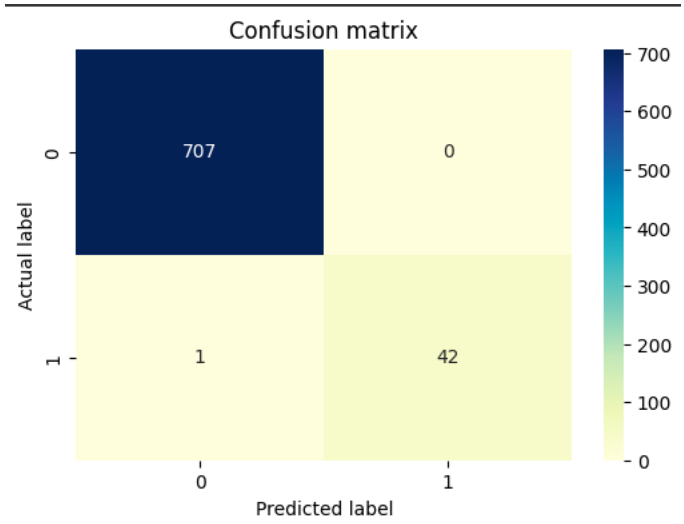
1. **Logistic Regression Classifier** with accuracy level of 0.98.



2. **Decision Tree Classifier** with accuracy level of 0.99



3. **Random Forest Classifier** with accuracy level of 0.99



Discussion

The main goal of this project was to predict customer churn. In a way that company can take necessarily actions to retain customers, there by improving customer satisfaction.

The models evaluated in this project include Logistic Regression, Decision Tree, and Random Forest classifiers. Each model based on its accuracy. The Decision Tree, and Random Forest classifiers emonstrated robust performance across multiple evaluation metrics.

Impact of Data Preprocessing: Handling missing values, removing outliers, and correctly encoding categorical variables were key to improving the model's accuracy and reliability. These preprocessing steps made sure the models were trained on high-quality data, which led to better performance on new data.

Feature Importance and Engineering: Feature engineering played a crucial role in enhancing model accuracy. New features such created to provide the models with more information about customer behaviour.

Model Performance and Tuning: Using Grid Search for hyperparameter tuning greatly improved the model's predictions, showing how important it is to define-tune in machine learning.

Conclusion

This project successfully demonstrated the approaches used to customer churn using machine learning. Through detailed data analysis and feature engineering, the Decision Tree and Random Forest models emerged as highly effective, achieving nearly 100% accuracy in identifying potential churn. These models provide robust tools for the company to manage and retain customers by targeting those most likely to churn by taking some appropriate actions.