



PREDICTING CUSTOMER CHURN FOR A TELECOMMUNICATIONS COMPANY

Machine learning project

Professor Giacomo Fiumara

Project by Mujibullah Rabin (539269)

University of Messina
Machine learning

Predicting Customer Churn for a Telecommunications Company.

CONTENTS:

- INTRODUCTION
- Understanding the dataset
- Data preprocessing
- Exploratory Data analysis (EDA)
- Feature engineering
- Model evaluation
- Model tuning
- Discussion
- Conclusion

INTRODUCTION

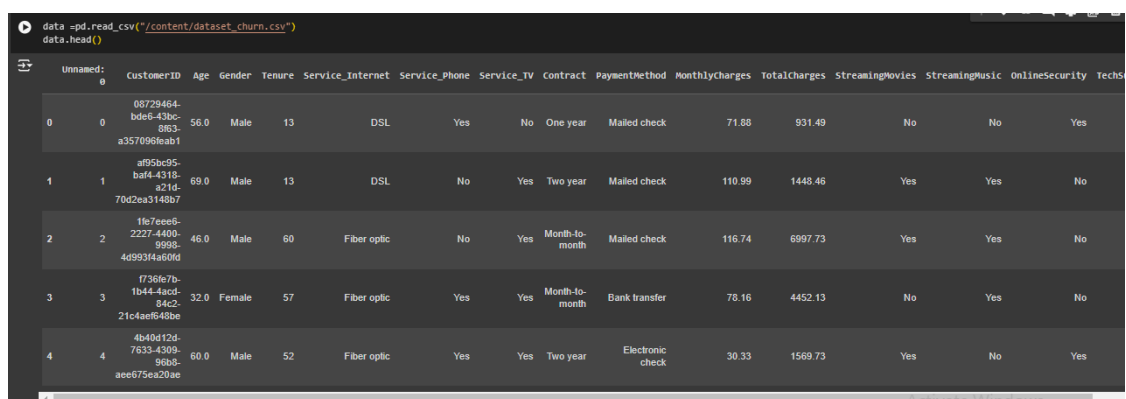
Customer churn is a critical issue for telecommunications companies as retaining existing customers is often more cost-effective than acquiring new ones. This project aims to predict customer churn using a synthetic dataset, analyse the factors influencing churn, and explore customer segmentation. By understanding these factors, the company can devise strategies to retain customers and improve service quality.

1.Understanding the dataset

the dataset contains various features related to customer demographics, account information, and service usage. Key features include customer tenure, contract type, payment method, monthly charges, and total charges.

The dataset used for the project consists of customer information from a telecommunications company. And the dataset is including 3749 entries with 17 columns.

So first we will upload the data and review it.



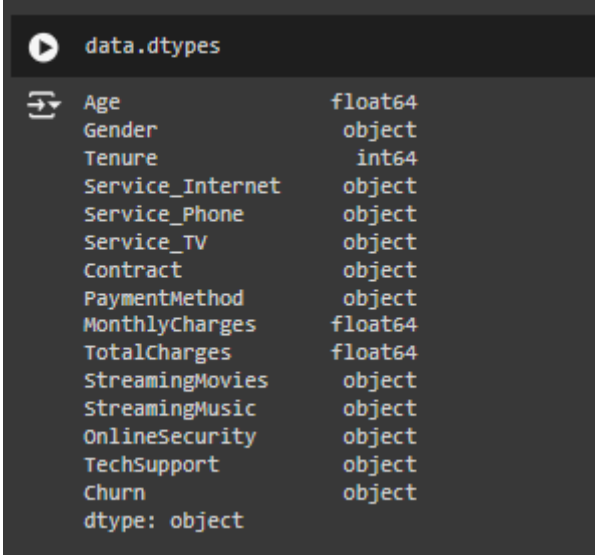
	Unnamed: 0	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechS
0	0	08729464-bde6-43bc-883-a357096feab1	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.86	931.49	No	No	Yes	
1	1	a195bcd5-baf4-4318-a21d-70d2ea3148b7	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46	Yes	Yes	No	
2	2	1fe7eee6-2227-4400-9998-4d9934a60dd	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73	Yes	Yes	No	
3	3	f736fe7b-1b44-4ae1-84c2-21c4ae6f48be	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	76.16	4452.13	No	Yes	No	
4	4	4b40d12d-7633-4309-96b8-ae675ea20ae	60.0	Male	52	Fiber optic	Yes	Yes	Two year	Electronic check	30.33	1569.73	Yes	No	Yes	

Then we will remove the unnecessary columns which they are not important for our analysis.

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

Dataset types

The dataset consists of various features related to customer demographics, account information, and service usage.



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

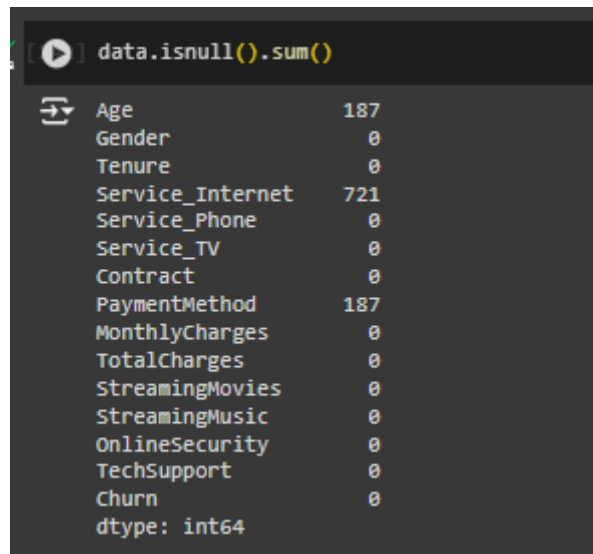
- **Age (float64):** The age of the customer. This feature is of type float, indicating it includes decimal values.
- **Gender (object):** The gender of the customer. This is a categorical variable stored as an object.
- **Tenure (int64):** The number of months the customer has been with the company. This is an integer value.
- **MonthlyCharges (float64):** The monthly charges incurred by the customer. This feature is of type float.
- **TotalCharges (float64):** The total charges incurred by the customer. This feature is also of type float.

Understanding the structure and types of the dataset's features is crucial for data preprocessing and subsequent analysis. The mixture of numerical and categorical data types requires specific handling techniques, which will be discussed in the Data Preprocessing section.

2.Data preprocessing

Data preprocessing involves handling missing values through imputation, detecting and treating outliers, and encoding categorical variables using one-hot encoding. Numerical features are scaled to ensure consistent ranges, and any anomalies are addressed to maintain data integrity.

as here we must address to the missing values.



```
data.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	

Handling Missing Values

For numerical features, missing values can be replaced with the mean, median, or mode of the feature. For categorical features, the most frequent category can be used.

Missing values in the numerical feature Age were replaced with the median value of the Age column.

For the categorical feature Service_Internet, missing values were filled with the mode (most frequent value) of the Service_Internet column.

```
[244] data['Age'] = data['Age'].fillna(data['Age'].median())  
  
data['Service_Internet'] = data['Service_Internet'].fillna(data['Service_Internet'].mode().iloc[0])  
data['PaymentMethod'] = data['PaymentMethod'].fillna(data['PaymentMethod'].mode().iloc[0])
```

Encoding

Machine learning models require numerical input, so categorical variables need to be converted to numerical form

Label Encoding: Applied to the following categorical features: Gender, Service_Internet, Service_Phone, Service_TV, Contract, PaymentMethod, StreamingMovies, StreamingMusic, OnlineSecurity, TechSupport, and Churn.

```
categorical_features = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',  
                        'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',  
                        'OnlineSecurity', 'TechSupport', 'Churn']  
for featurew in categorical_features:  
    data[featurew] = LabelEncoder().fit_transform(data[featurew])  
data.head()
```

	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechSupport
0	56.0	1	13	0	1	0	1	3	71.88	931.49	0	0	1	
1	69.0	1	13	0	0	1	2	3	110.99	1448.46	1	1	0	
2	46.0	1	60	1	0	1	0	3	116.74	6997.73	1	1	0	
3	32.0	0	57	1	1	1	0	0	78.16	4452.13	0	1	0	
4	60.0	1	52	1	1	1	2	2	30.33	1569.73	1	0	1	

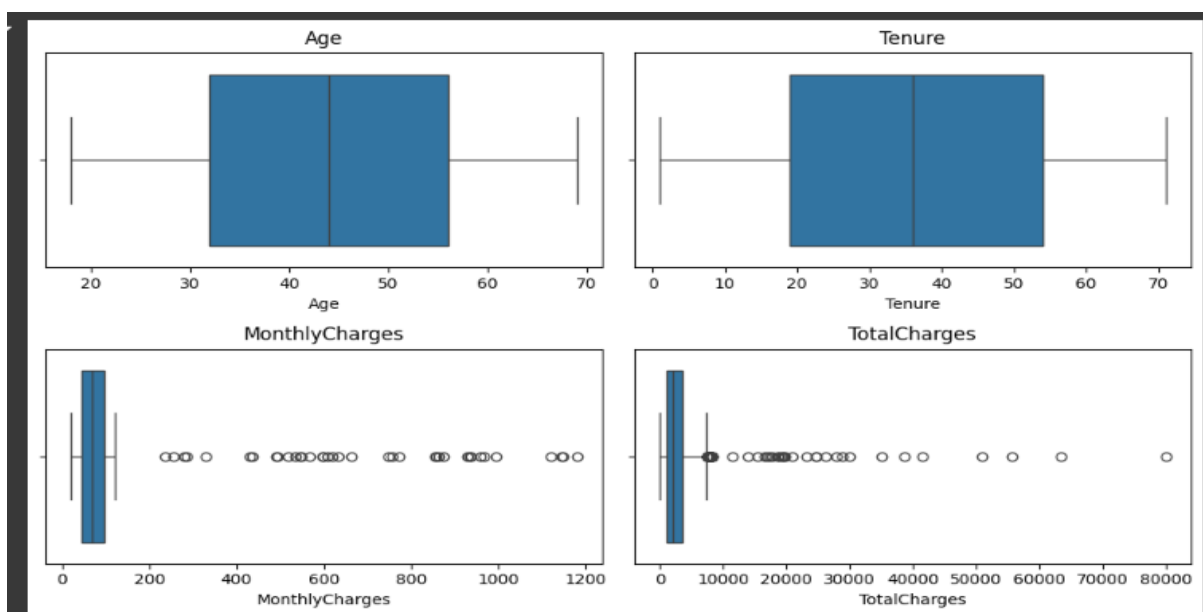
This ensures that all categorical variables are converted into numerical format, making them suitable for machine learning algorithms.

Handling the outlier

Outliers are data points that differ significantly from other observations and can distort the training process of a machine learning model, leading to poor performance.

Identifying outliers:

The first step in handling outliers is to identify them. Box plots were utilized to visualize the distribution of numerical features and to spot outliers.



The following box plots display the distributions and outliers for the numerical features Age, Tenure, MonthlyCharges, and TotalCharges:

Where **Age and Tenure**: Both features show a clean and symmetrical distribution with no significant outliers, suggesting stable and uniform data for these characteristics.

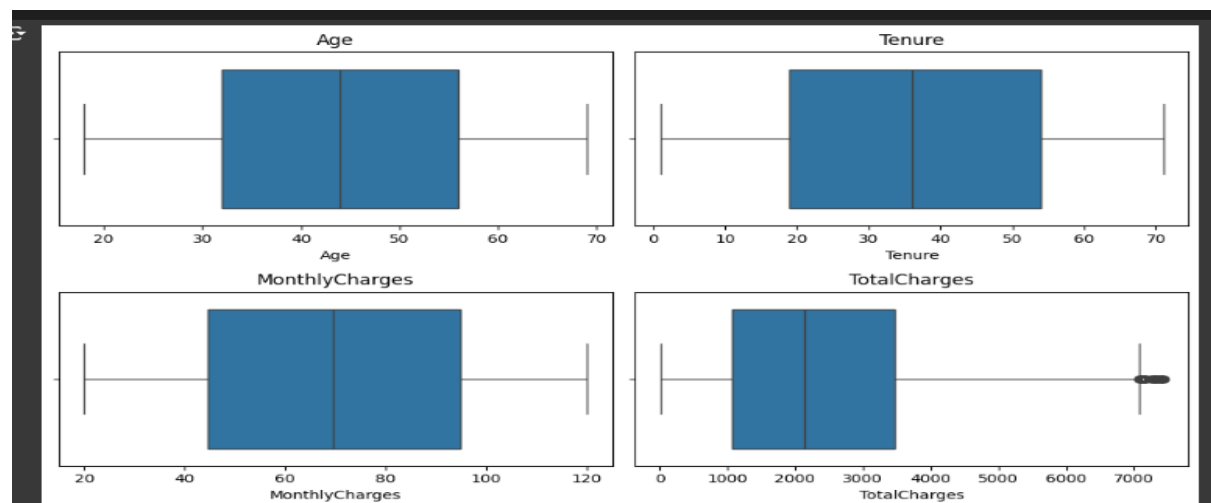
And **MonthlyCharges and TotalCharges**: These features display a notable number of outliers. The presence of these outliers indicates variability in customer spending habits. While MonthlyCharges captures the monthly expenditure, TotalCharges accumulates over time, hence showing a broader range and more outliers.

Handling the outlier:

Now to handle the outlier we replace them with median value. This ensures that the extreme values do not skew the data distribution while preserving the data points.

```
def replace_outliers_with_median(data, column):  
    Q1 = data[column].quantile(0.25)  
    Q3 = data[column].quantile(0.75)  
    IQR = Q3 - Q1  
    lower_bound = Q1 - 1.5 * IQR  
    upper_bound = Q3 + 1.5 * IQR  
  
    median_value = data[column].median()  
  
    data[column] = np.where(  
        (data[column] < lower_bound) | (data[column] > upper_bound),  
        median_value,  
        data[column]  
    )  
    return data  
  
for features in numerical_features:  
    data = replace_outliers_with_median(data, features)
```

After treating the outliers by replacing them with the median, the distribution of the numerical features became more normalized. The following box plots display the distributions after the handling of outlier.

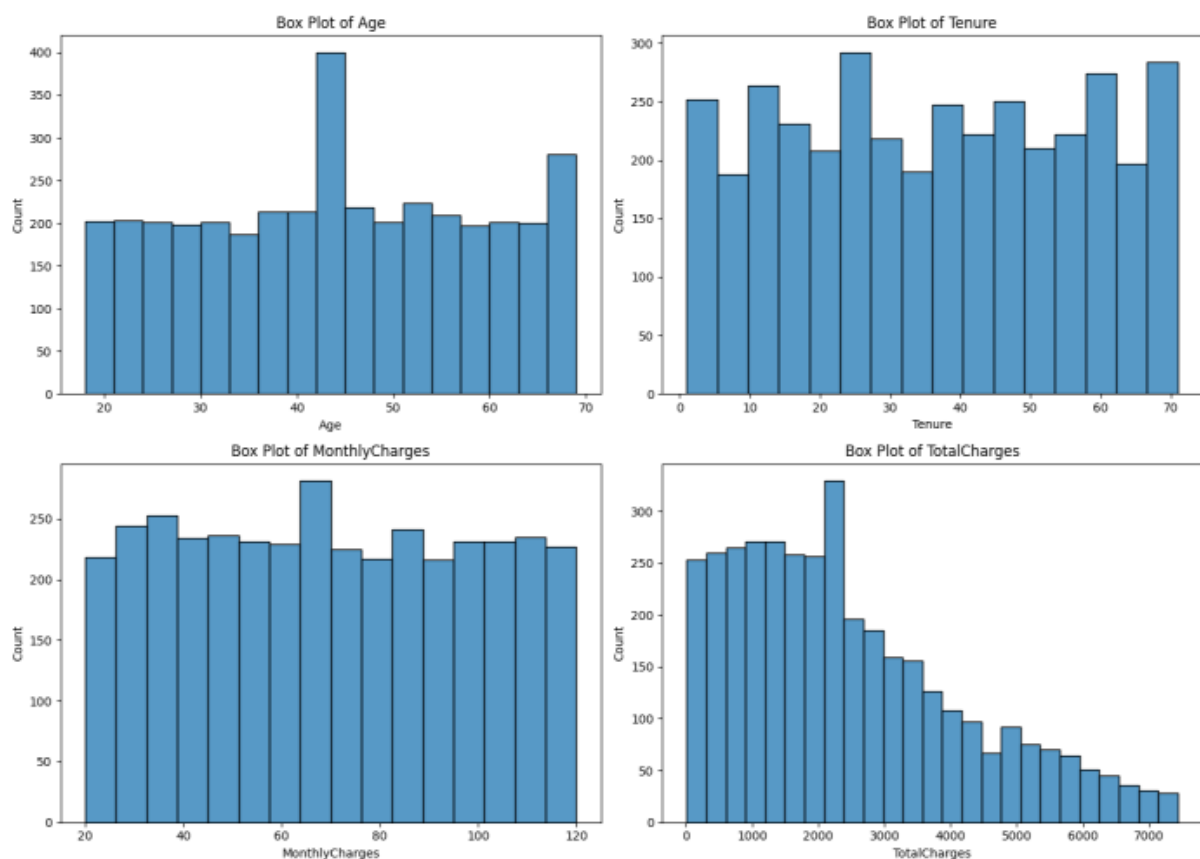


3.Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in understanding the underlying patterns and relationships in the dataset. It involves summarizing the main characteristics of the data often using visual methods.

Numerical features

Here I have used the histogram to visualize the distribution of numerical features.



- **Age**
The Age feature shows a uniform distribution across different age groups with a peak around the age of 40. This indicates a diverse age range among the customers.
- **Tenure**
The Tenure histogram displays a varied distribution with multiple peaks, suggesting different clusters of tenure durations among customers. This indicates that customers have a wide range of loyalty periods with the company, with no single tenure period dominating.

- **Monthly charges**

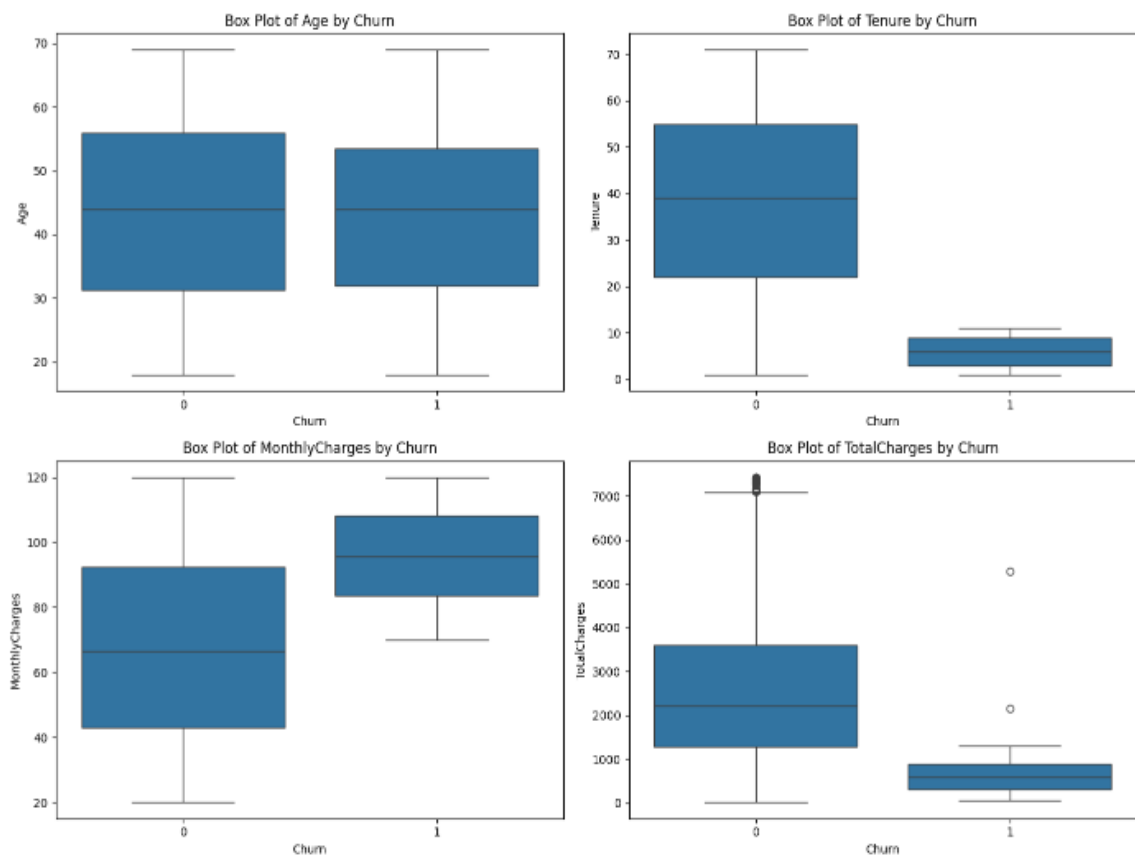
The histogram for MonthlyCharges shows a relatively uniform distribution across various charge ranges. This suggests that customers are evenly distributed in terms of their monthly spending, indicating a well-balanced pricing structure without any dominant charge range.

- **Total charges**

The TotalCharges histogram is right-skewed, indicating that while most customers have lower accumulated charges, a smaller number have very high total charges. The skewness towards higher total charges implies that long-term customers or those with higher monthly charges contribute significantly to the total revenue.

relationship between features and the target variable (Churn):

The following box plots display the distributions of numerical features (Age, Tenure, MonthlyCharges, and TotalCharges) segmented by the target variable Churn (where 0 indicates no churn and 1 indicates churn):



- **Age by Churn**

The box plots for Age show a similar distribution for both churned and non-churned customers, indicating that age may not be a significant factor in predicting churn. Since the median age and the spread of ages are similar for both groups, age does not appear to have a strong relationship with churn.

- **Tenure by churn**

- The Tenure box plot shows a clear difference between churned and non-churned customers. Non-churned customers tend to have a higher tenure, while churned customers have significantly lower tenure.

This suggests that customers with longer tenure are less likely to churn, highlighting tenure as an important factor in predicting churn.

- **Monthly charges by churn:**

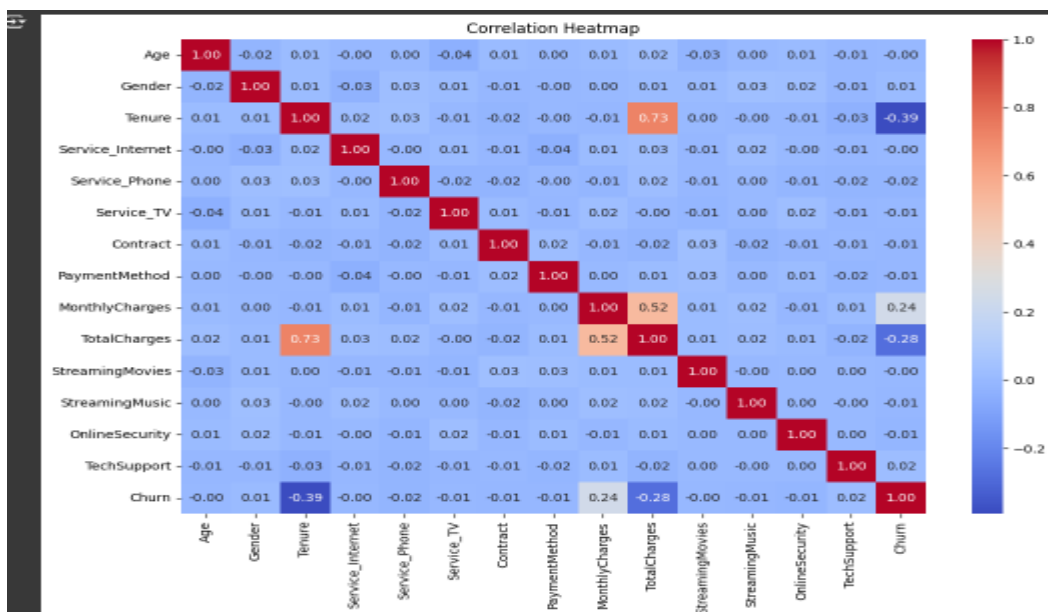
The box plots for MonthlyCharges indicate that customers who churn tend to have slightly higher monthly charges compared to those who do not churn. Higher monthly charges may be associated with increased likelihood of churn, suggesting that pricing strategies could be an area to focus on for churn reduction.

- **TotalCharges by Churn:**

The TotalCharges box plot shows that churned customers tend to have lower total charges compared to non-churned customers. Since total charges accumulate over time, this again reflects that long-term customers (with higher total charges) are less likely to churn. This underscores the importance of customer retention strategies to maintain long-term customer relationships.

Correlation between features:

A correlation heatmap is a visual representation of the correlation matrix, showing the correlation coefficients between pairs of features in the dataset.



Here in this heatmap the values range from -1 to 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation, and 0 indicates no correlation.

Additionally, this correlation heatmap underscores the critical factors influencing customer churn in the dataset. Key features such as Tenure, MonthlyCharges, and TotalCharges play significant roles in determining whether a customer will churn. By focusing on these features, the predictive models can be more accurately tuned to identify at-risk customers.

4.Feature engineering

Feature engineering is the process of creating new features or modifying existing ones to improve the performance of machine learning models. This step is crucial as it helps to extract more relevant information from the raw data, enabling the model to make more accurate predictions.

Creating new features

```
[255] data['Tenure_MonthlyCharges'] = data['MonthlyCharges'] / data['Tenure']

data['AvgMonthlyCharges'] = data['TotalCharges'] / (data['Tenure'] + 1e-10)

[257] data['TotalServices'] = (data['Service_Internet'] +
                             data['Service_Phone'] +
                             data['Service_TV'] +
                             data['StreamingMusic'] + data['OnlineSecurity'] + data['TechSupport'])

[258] data['AvgChargesPerService'] = data['MonthlyCharges'] / (data['TotalServices'] + 1e-10)

[259] data.head()
```

	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechSupport
0	56.0	1	13.0	0	1	0	1	3	71.88	931.49	0	0	1	
1	69.0	1	13.0	0	0	1	2	3	110.99	1448.46	1	1	0	
2	46.0	1	60.0	1	0	1	0	3	116.74	6997.73	1	1	0	
3	32.0	0	57.0	1	1	1	0	0	78.16	4452.13	0	1	0	
4	60.0	1	52.0	1	1	1	2	2	30.33	1569.73	1	0	1	

- **Total Services:**
- This feature counts the total number of services a customer subscribes to, including internet, phone, TV, streaming music, streaming movies, online security, and tech support. Created by By summing up binary indicators for various services.
- **Average Charges per Service:**
This feature calculates the average monthly charge per service a customer uses. created by By dividing MonthlyCharges by TotalServices

- **Tenure_MonthlyCharges:**

This feature represents the average monthly charges over the tenure period. Created by dividing MonthlyCharges by Tenure

- **AvgMonthlyCharges:**

This feature represents the average monthly charges, normalized to avoid division by zero. Created by dividing TotalCharges by Tenure with a small constant added to avoid division by zero.

dataset now includes several new features that capture the relationships between service usage, billing amounts, and tenure. These features aim to provide a deeper insight into customer behavior and their likelihood to churn.

5. Modeling

In the modeling phase, various machine learning algorithms are trained and evaluated to predict customer churn. This process involves splitting the data into training and testing sets, training multiple models, and selecting the best model based on evaluation metrics.

Splitting the Data

The dataset is split into training and testing sets to evaluate the model's performance on unseen data.

```
[260] x = data.drop('Churn', axis=1) #features
      y = data['Churn'] # Target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Here data is split into the training and testing whereas I have putted the 0.2% of the data in testing and the rest in the training.

Scaling the features

In scaling we normalize the feature means Scaling numerical features ensures that they are on a similar scale, which is particularly important for algorithms that rely on distance metrics.

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

training models

Now I will use default models

I have used 4 models:

1. Logistic regression
2. Decision tree
3. Random forest
4. Gradient boosting

```
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}

def evaluate_models(models, X_train, y_train, X_test, y_test):
    results = {}
    for model_name, model in models.items():
        print(f"Training and evaluating {model_name}...")

        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)
        print(f"Classification Report for {model_name}:")
        print(classification_report(y_test, y_pred))

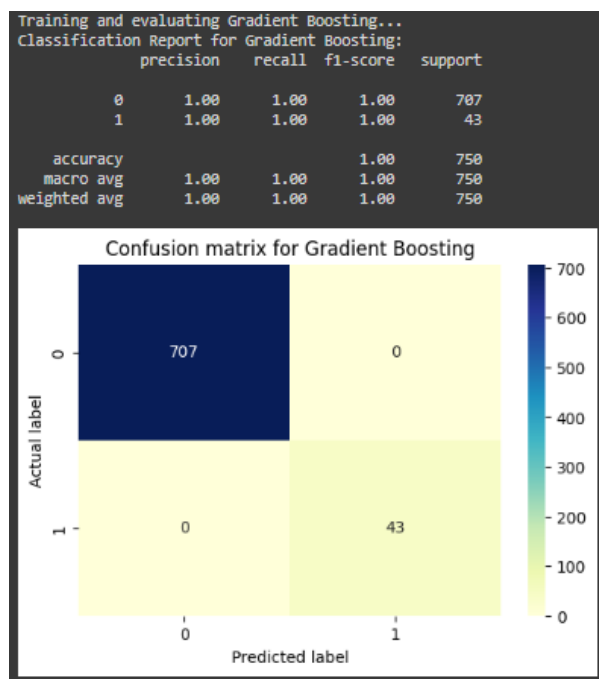
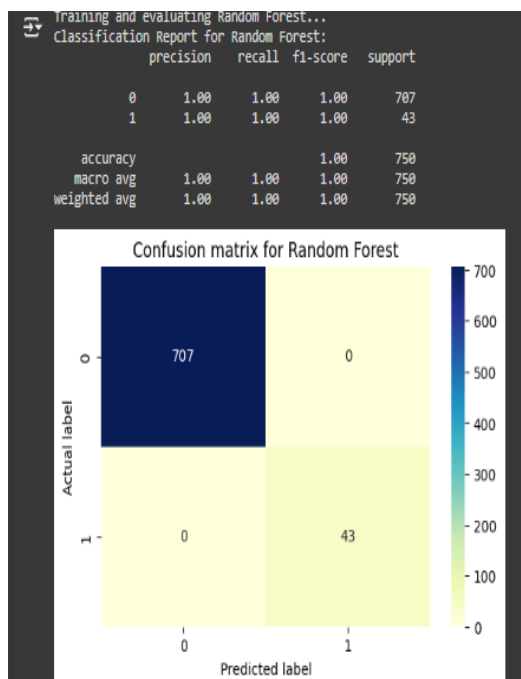
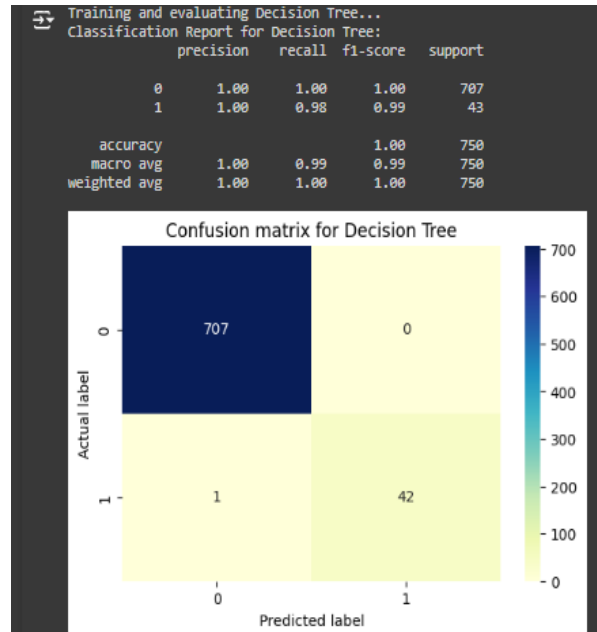
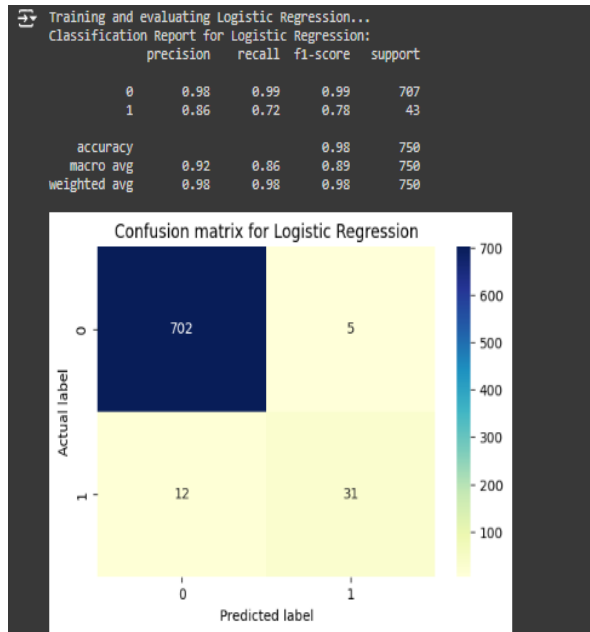
        conf_matrix = confusion_matrix(y_test, y_pred)

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

So here I used the default models without hypertunning so then I can use the model with grid search and hypertunning to see the differences.

Evaluating the model using metrics

After applying the models here I visualized all models I have used.



In this evaluation, we used the default parameters for each model to establish baseline performance. These default settings provide a good starting point for understanding the capabilities and limitations of each model before applying more advanced techniques like hyperparameter tuning

6. Model tuning

now I have used the grid and models

In the context of machine learning, a "grid" refers to a set of hyperparameters that we want to explore to find the best configuration for a given model.

Hyper parameters are parameters that they don't learn from the data instead the set prior the training set.

```
param_grids = {
    "Logistic Regression": {
        'C': [0.1, 1, 10],
        'solver': ['liblinear']
    },
    "Decision Tree": {
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 10, 20]
    },
    "Random Forest": {
        'n_estimators': [100, 200, 500],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 10, 20]
    },
    "Gradient Boosting": {
        'n_estimators': [100, 200, 500],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 10]
    }
}
```

Here in each param_grids correspond to a model and specifies the hyperparameters and their respective values to be tested.

```
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}
```

Here I have initialized all the models.

Then I have used a function to evaluate the models

```
def evaluate_models(models, param_grids, X_train, y_train, X_test, y_test):
    results = {}
    for model_name, model in models.items():
        print(f"Performing GridSearchCV for {model_name}...")
```

The `evaluate_models` function is designed to train, tune, and evaluate each model using `GridSearchCV` for hyperparameter tuning and cross-validation.

The process of exploring these hyperparameter grids is called "grid search". `GridSearchCV` from the `sklearn.model_selection`

Now for each model I have initialized the grid search.

```
grid_search = GridSearchCV(model, param_grids[model_name], cv=5, scoring='roc_auc', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
```

Here grid search is initialized, and it gets the best model for grid search.

After the best model has been identified using `GridSearchCV` it is used to make predictions on the test dataset:

```
# Predict on the test set
y_pred = best_model.predict(X_test)
print(f"Classification Report for {model_name}:")
print(classification_report(y_test, y_pred))
```

Now we can visualize it using the confusion matrix to provide the detail of each model's performance on each class.

```
conf_matrix = confusion_matrix(y_test, y_pred)

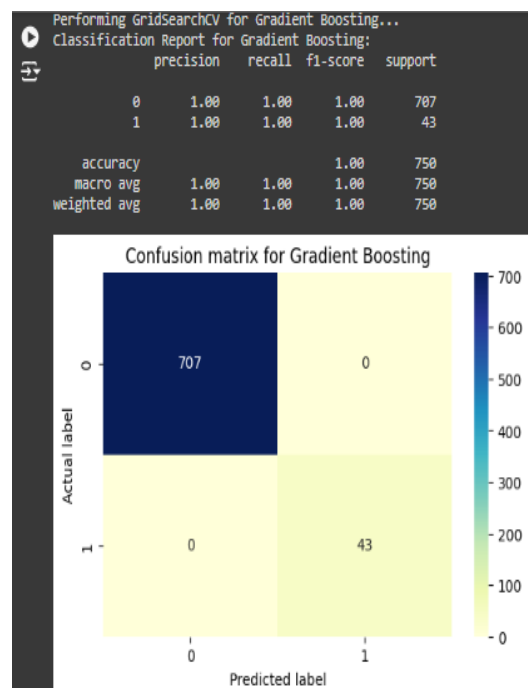
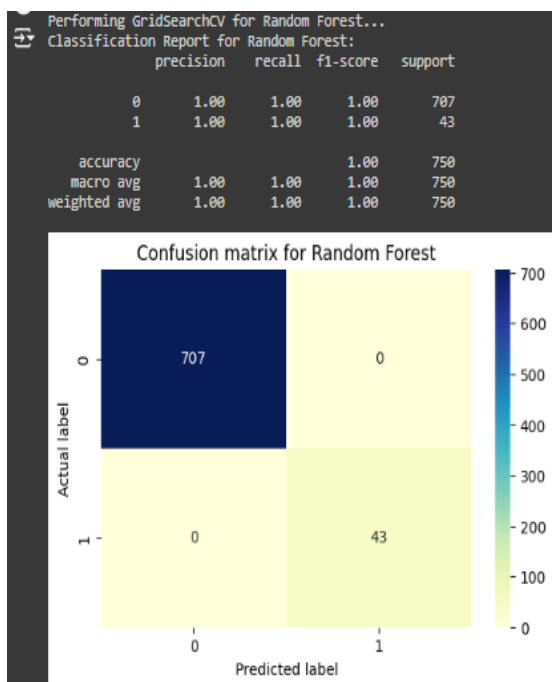
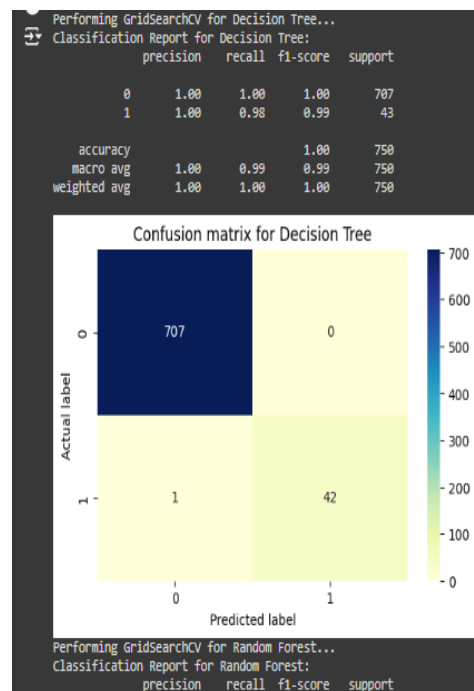
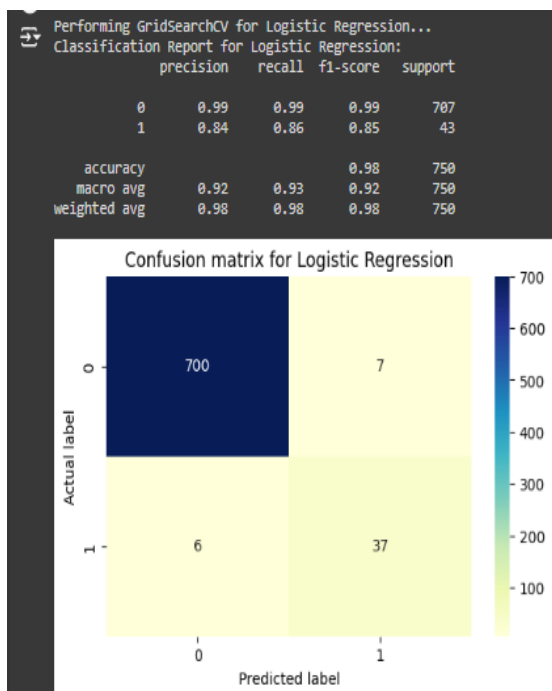
plt.figure(figsize=(6, 4))
sns.heatmap(pd.DataFrame(conf_matrix), annot=True, cmap="YlGnBu", fmt='g')
plt.title(f'Confusion matrix for {model_name}')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```

Then the results dictionary is converted into a `DataFrame` for better visualization and analysis:

The evaluation function is executed with the specified models, parameter grids, and scaled training and testing datasets

Finally, the results `DataFrame` is displayed

And here are the result with advance technique hyperparameter tuning:



Discussion

The main goal of this project was to predict customer churn, enabling the company to take necessary actions to retain customers and improve overall customer satisfaction. The models evaluated include Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting classifiers. The Decision Tree and Random Forest classifiers demonstrated robust performance across multiple evaluation metrics.

Impact of Data Preprocessing: Handling missing values, removing outliers, and encoding categorical variables were essential for improving model accuracy and reliability. These steps ensured the models were trained on high-quality data, leading to better performance on new data.

Feature Importance and Engineering: Feature engineering played a crucial role in enhancing model accuracy. New features, such as Total Services and Average Charges per Service, were created to provide the models with more information about customer behavior. Identifying key features like Tenure, Monthly Charges, and Total Charges helped improve the predictive power of the models.

Model Performance and Tuning: Using Grid Search for hyperparameter tuning greatly improved the model's predictions, showing the importance of fine-tuning in machine learning. This process helped in selecting the optimal settings for each model, enhancing their performance on unseen data.

Conclusion:

This project successfully identified the key factors influencing customer churn and developed predictive models to help the company retain at-risk customers. By integrating robust preprocessing, feature engineering, model tuning, and interpretation techniques, the models provided accurate and actionable insights for business decision-making.

