Faculty of Computer Science, Data Analysis Program
University Of Messina

Predicting Customer Churn for a Telecommunications Company

Professor: **Giacomo Fiumara** Student:**Yahia Ahmed**(532118)

Academic Year 2023/2024

Table of Contents:

Project Introduction and Overview

In the highly competitive telecommunications industry, retaining customers is crucial for sustained growth and profitability. Customer churn, which occurs when customers discontinue their service, poses a significant challenge for telecom companies. Understanding the factors that lead to churn and developing predictive models to identify potential churners can provide valuable insights for creating effective retention strategies.

This project aims to predict customer churn using a synthetic dataset, analyze the factors that contribute to churn, and explore customer segmentation. By leveraging various machine learning techniques, I seek to develop robust models that not only predict churn but also offer interpretability, enabling telecom companies to devise targeted interventions. The project is structured into several key activities: data exploration, preprocessing, detailed exploratory data analysis, feature engineering, model building and evaluation, model tuning, and interpretation of results.

Through this comprehensive approach, I aim to identify patterns and insights that help in formulating strategies to reduce churn and enhance customer satisfaction. The final outcome will include actionable recommendations based on data-driven insights, enabling telecom companies to implement effective measures to retain their customers and improve overall business performance. This project will provide a detailed understanding of customer behavior, helping companies to make informed decisions and stay competitive in the market.

Dataset Description

The dataset used in this project is a synthetic dataset aimed at predicting customer churn for a telecommunications company. It includes various features that describe customer demographics, services subscribed to, account information, and their churn status.

Below is a detailed description of each feature in the raw dataset (before **prepossessing,encoding and feature-engineering**):

1. **CustomerID**
    - **Description**: A unique identifier for each customer.
    - **Type**: Categorical (unique string).
2. **Age**
    - **Description**: The age of the customer.
    - **Type**: Numerical (continuous).
    - **Range**: Values typically range from 18 to 90 years.
3. **Gender**
    - **Description**: The gender of the customer.
    - **Type**: Categorical (binary).
    - **Values**: 'Male', 'Female'.
4. **Tenure**
    - **Description**: The number of months the customer has stayed with the company.
    - **Type**: Numerical (continuous).
    - **Range**: Values typically range from 1 to 72 months.
5. **Service_Internet**

- **Description**: The type of internet service the customer has subscribed to.
- **Type**: Categorical.
- **Values**: 'None', 'DSL', 'Fiber optic'.

6. **Service_Phone**
   - **Description**: Indicates if the customer has a phone service.
   - **Type**: Categorical (binary).
   - **Values**: 'Yes', 'No'.
7. **Service_TV**
   - **Description**: Indicates if the customer has a TV service.
   - **Type**: Categorical (binary).
   - **Values**: 'Yes', 'No'.
8. **Contract**
   - **Description**: The contract term of the customer.
   - **Type**: Categorical.
   - **Values**: 'Month-to-month', 'One year', 'Two year'.
9. **PaymentMethod**
   - **Description**: The method of payment used by the customer.
   - **Type**: Categorical.
   - **Values**: 'Electronic check', 'Mailed check', 'Bank transfer', 'Credit card'.
10. **MonthlyCharges**
    - **Description**: The amount charged to the customer monthly.
    - **Type**: Numerical (continuous).
    - **Range**: Values typically range from $18.25 to $118.75.
11. **TotalCharges**
    - **Description**: The total amount charged to the customer over the tenure period.
    - **Type**: Numerical (continuous).
    - **Range**: Values typically range from $18.80 to $8684.80.
12. **StreamingMovies**
    - **Description**: Indicates if the customer has a streaming movies service.
    - **Type**: Categorical (binary).
    - **Values**: 'Yes', 'No'.
13. **StreamingMusic**
    - **Description**: Indicates if the customer has a streaming music service.
    - **Type**: Categorical (binary).
    - **Values**: 'Yes', 'No'.
14. **OnlineSecurity**
    - **Description**: Indicates if the customer has online security services.
    - **Type**: Categorical (binary).
    - **Values**: 'Yes', 'No'.
15. **TechSupport**

- **Description**: Indicates if the customer has tech support services.
- **Type**: Categorical (binary).
- **Values**: 'Yes', 'No'.

16. **Churn**

    - **Description**: Indicates if the customer has churned (left the company).
    - **Type**: Categorical (binary).
    - **Values**: 'Yes', 'No'.

And Here is a detailed description of each feature in the raw dataset (after **prepossessing and encoding**) (all the details are discussed in the upcoming sections):

1. **CustomerID**
    - **Description**: A unique identifier for each customer.
    - **Type**: Categorical (unique string).
    - **Note**: Dropped after preprocessing as it's not needed for modeling.
2. **Age**
    - **Description**: The age of the customer.
    - **Type**: Numerical (continuous).
    - **Note**: Missing values handled by median imputation, scaled.
3. **Tenure**
    - **Description**: The number of months the customer has stayed with the company.
    - **Type**: Numerical (continuous).
    - **Note**: Missing values handled by median imputation, scaled.
4. **MonthlyCharges**
    - **Description**: The amount charged to the customer monthly.
    - **Type**: Numerical (continuous).
    - **Note**: Missing values handled by median imputation, scaled.
5. **TotalCharges**
    - **Description**: The total amount charged to the customer over the tenure period.
    - **Type**: Numerical (continuous).
    - **Note**: Missing values handled by median imputation, scaled.
6. **Gender_Male**
    - **Description**: Indicates if the customer is male.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
7. **Service_Internet_DSL**
    - **Description**: Indicates if the customer has DSL internet service.
    - **Type**: Categorical (binary).

- **Note**: One-hot encoded.

8. **Service_Internet_Fiber optic**
    - **Description**: Indicates if the customer has Fiber optic internet service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
9. **Service_Phone_Yes**
    - **Description**: Indicates if the customer has phone service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
10. **Service_TV_Yes**
    - **Description**: Indicates if the customer has TV service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
11. **Contract_One year**
    - **Description**: Indicates if the customer has a one-year contract.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
12. **Contract_Two year**
    - **Description**: Indicates if the customer has a two-year contract.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
13. **PaymentMethod_Credit card**
    - **Description**: Indicates if the customer pays by credit card.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
14. **PaymentMethod_Electronic check**
    - **Description**: Indicates if the customer pays by electronic check.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
15. **PaymentMethod_Mailed check**
    - **Description**: Indicates if the customer pays by mailed check.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
16. **StreamingMovies_Yes**
    - **Description**: Indicates if the customer has streaming movies service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
17. **StreamingMusic_Yes**
    - **Description**: Indicates if the customer has streaming music service.

- **Type**: Categorical (binary).
- **Note**: One-hot encoded.

18. **OnlineSecurity_Yes**
    - **Description**: Indicates if the customer has online security service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
19. **TechSupport_Yes**
    - **Description**: Indicates if the customer has tech support service.
    - **Type**: Categorical (binary).
    - **Note**: One-hot encoded.
20. **Churn**
    - **Description**: Indicates if the customer has churned (left the company).
    - **Type**: Categorical (binary).
    - **Note**: Converted to binary (1 for 'Yes', 0 for 'No').

And Here is a detailed description of each feature in the raw dataset (after **prepossessing, encoding and FEATURE ENGINEERING**) (all the details are discussed in the upcoming sections):

1. **CustomerID**
    - **Description**: A unique identifier for each customer.
    - **Type**: Categorical (unique string).
    - **Note**: Dropped after preprocessing as it's not needed for modeling.
2. **Age**
    - **Description**: The age of the customer.
    - **Type**: Numerical (continuous).
    - **Note**: Scaled.
3. **Tenure**
    - **Description**: The number of months the customer has stayed with the company.
    - **Type**: Numerical (continuous).
    - **Note**: Scaled.
4. **MonthlyCharges**
    - **Description**: The amount charged to the customer monthly.
    - **Type**: Numerical (continuous).
    - **Note**: Scaled.
5. **TotalCharges**

- **Description**: The total amount charged to the customer over the tenure period.
- **Type**: Numerical (continuous).
- **Note**: Scaled.

6. **Gender_Male**
    - **Description**: Indicates if the customer is male.
    - **Type**: Categorical (binary).
7. **Service_Internet_DSL**
    - **Description**: Indicates if the customer has DSL internet service.
    - **Type**: Categorical (binary).
8. **Service_Internet_Fiber optic**
    - **Description**: Indicates if the customer has Fiber optic internet service.
    - **Type**: Categorical (binary).
9. **Service_Phone_Yes**
    - **Description**: Indicates if the customer has phone service.
    - **Type**: Categorical (binary).
10. **Service_TV_Yes**
    - **Description**: Indicates if the customer has TV service.
    - **Type**: Categorical (binary).
11. **Contract_One year**
    - **Description**: Indicates if the customer has a one-year contract.
    - **Type**: Categorical (binary).
12. **Contract_Two year**
    - **Description**: Indicates if the customer has a two-year contract.
    - **Type**: Categorical (binary).
13. **PaymentMethod_Credit card**
    - **Description**: Indicates if the customer pays by credit card.
    - **Type**: Categorical (binary).
14. **PaymentMethod_Electronic check**
    - **Description**: Indicates if the customer pays by electronic check.
    - **Type**: Categorical (binary).
15. **PaymentMethod_Mailed check**
    - **Description**: Indicates if the customer pays by mailed check.
    - **Type**: Categorical (binary).
16. **StreamingMovies_Yes**
    - **Description**: Indicates if the customer has streaming movies service.
    - **Type**: Categorical (binary).
17. **StreamingMusic_Yes**
    - **Description**: Indicates if the customer has streaming music service.
    - **Type**: Categorical (binary).

18. **OnlineSecurity_Yes**
    - **Description**: Indicates if the customer has online security service.
    - **Type**: Categorical (binary).
19. **TechSupport_Yes**
    - **Description**: Indicates if the customer has tech support service.
    - **Type**: Categorical (binary).
20. **Churn**
    - **Description**: Indicates if the customer has churned (left the company).
    - **Type**: Categorical (binary).
    - **Note**: Converted to binary (1 for 'Yes', 0 for 'No').
21. **TotalServices**
    - **Description**: The total number of services a customer subscribes to (internet, phone, TV).
    - **Type**: Numerical (discrete).
    - **Calculation**: Sum of binary indicators for internet, phone, and TV services.
22. **MonthlyChargesPerService**
    - **Description**: The average monthly charge per service subscribed to by the customer.
    - **Type**: Numerical (continuous).
    - **Calculation**: `MonthlyCharges / TotalServices`.
    - **Note**: NaNs and infinite values handled by setting to 0 if no services are subscribed.
23. **TenurePerService**
    - **Description**: The average tenure per service subscribed to by the customer.
    - **Type**: Numerical (continuous).
    - **Calculation**: `Tenure / TotalServices`.
    - **Note**: NaNs and infinite values handled by setting to 0 if no services are subscribed.

Python libraries used.

The main python libraries or frameworks I used were :

Pandas: Pandas is a powerful Python library used for data manipulation and analysis, providing data structures like DataFrames and Series that are essential for handling and processing structured data. It is widely used for cleaning, transforming, and visualizing data due to its intuitive syntax and robust functionality.

Numpy: a fundamental library for numerical computing in Python, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It is the backbone of scientific computing in Python, providing computational efficiency and array manipulation capabilities required for data analysis and machine learning.

Seaborn: Seaborn is a statistical data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics. It makes it easy to create complex visualizations like heatmaps, violin plots, and pair plots, which are useful for exploring and understanding data patterns and relationships.

Sklearn:(or scikit-learn) is a robust machine learning library for Python, offering simple and efficient tools for data mining and data analysis. It covers a wide range of algorithms for classification, regression, clustering, and dimensionality reduction. Sklearn is known for its ease of use and integration with other scientific libraries like Numpy and Pandas, making it a popular choice for building machine learning models.

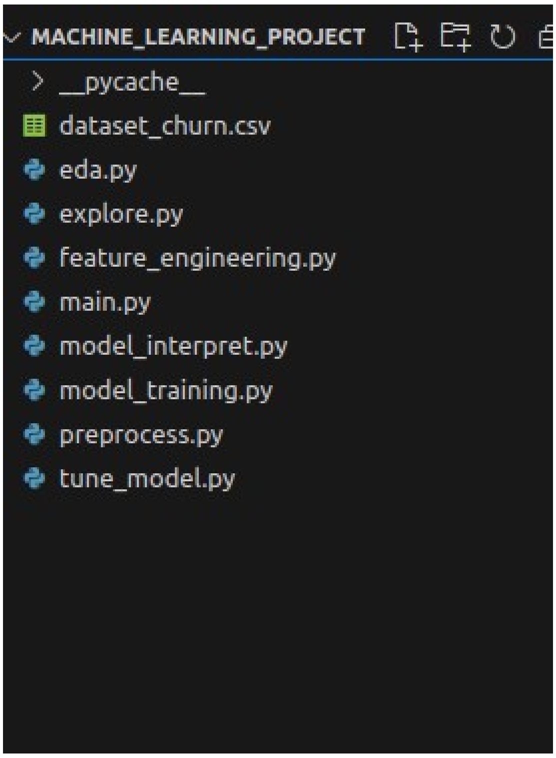
LIME (Local Interpretable Model-agnostic Explanations): LIME is a library designed to explain the predictions of machine learning models by approximating them with simpler, interpretable models in the vicinity of the prediction. This increases the interpretability of complex models, providing insights into why a model made a particular decision and helping users understand and trust their machine learning models.

Project and Code Base Architecture and Modules.

In order to make the project a good and understandable architacture I decided to make the project to be splitted into **8** modules or files, in which each and every file has a specific task to do.

The modules name I created are as follows:

1. **explore.py**
2. **preprocess.py**
3. **eda.py**
4. **feature engineering.py**
5. **model_training.py**
6. **tune_model.py**
7. **model_interpret.py**
8. **main.py**



Here I am going to give a detailed overview about each of the modules.

1. `explore.py`

Purpose: This file is responsible for the initial exploration of the dataset.

Functions and Operations:

- Load the dataset.
- Understand the structure of the dataset.
- Perform initial exploratory data analysis (EDA).
- Describe each feature and its possible values.
- Visualize distributions of numerical features (e.g., histograms, box plots).
- Analyze relationships between features using scatter plots and correlation matrices.

2. `preprocess.py`

Purpose: This file handles data cleaning and preparation.

Functions and Operations:

- Handle missing values (imputation or removal).
- Detect and treat outliers.
- Encode categorical variables using one-hot encoding or label encoding.
- Normalize or scale numerical features if necessary.
- Save the cleaned dataset.

3. `eda.py`

Purpose: This file performs in-depth exploratory data analysis (EDA) to understand key factors influencing customer churn.

Functions and Operations:

- Investigate the relationship between features and the target variable (Churn).
- Use visualization techniques like bar charts, box plots, and heatmaps.
- Perform hypothesis testing where applicable (e.g., t-tests, chi-square tests).
- Provide insights into patterns and correlations within the data.

4. `feature_engineering.py`

Purpose: This file handles the creation of new features to improve model performance.

Functions and Operations:

- Combine existing features to create new ones (e.g., interaction terms, ratios).
- Use domain knowledge to add relevant features (e.g., total service usage).
- Normalize the newly created features.

- Save the enhanced dataset with new features.

5. `model_training.py`

Purpose: This file is responsible for building and evaluating machine learning models.

Functions and Operations:

- Split the data into training and testing sets.
- Initialize and train multiple models (e.g., logistic regression, decision trees, random forest, gradient boosting).
- Evaluate models using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
- Perform cross-validation to ensure robust performance estimates.

6. `tune_model.py`

Purpose: This file handles the hyperparameter tuning of machine learning models to improve performance.

Functions and Operations:

- Use techniques like grid search or randomized search for hyperparameter tuning.
- Compare the performance of tuned models against default models.
- Document the tuning process and the chosen hyperparameters.
- Save the best-tuned models for further use.

7. `model_interpret.py`

Purpose: This file interprets the trained models to understand which features are most important for predicting churn.

Functions and Operations:

- Use feature importance scores from models like decision trees or random forests.
- Apply LIME (Local Interpretable Model-agnostic Explanations) for detailed model interpretation.
- Discuss how the insights align with domain knowledge and business logic.
- Visualize the model's decision-making process for individual predictions.

8. `main.py`

Purpose: This is the main file that orchestrates the entire process, from data loading to model interpretation.

Functions and Operations:

- Load the initial dataset.
- Call functions from other files to perform preprocessing, EDA, feature engineering, model training, tuning, and interpretation.
- Ensure the workflow is executed in the correct order.
- Save final models and results for reporting.

Project Activities

1- Pre–Exploration Phase:

In the `exploreTheDataSet` function, I performed a comprehensive initial exploration of the dataset. This process is essential to understand the data's structure, identify any potential issues, and gain insights that will guide subsequent data preprocessing and modeling steps.

Here's a summary of what was done in this function:

1. **Initial Inspection**: I started by displaying the first few rows of the dataset to get a quick overview of the data and its structure. This helped us understand the types of features present and the format of the data.
2. **Data Summary**: I then provided a summary of the dataset, including the data types of each column and the number of non-null values. This step was crucial for identifying any missing values and understanding the types of data (e.g., numerical, categorical) I are dealing with.
3. **Missing Values**: I counted the number of missing values in each column to understand the extent of missing data. Identifying missing values early on helps in planning how to handle them during preprocessing.
4. **Statistical Summary**: I generated summary statistics for numerical features, including measures like count, mean, standard deviation, minimum, and maximum values. This statistical summary provided insights into the distribution and central tendency of numerical features, helping us identify any anomalies or outliers.
5. **Feature Distribution**: I examined the unique values and their counts for each feature. This step was particularly useful for categorical features, allowing us to understand their distribution and identify any unusual patterns or inconsistencies.
6. **Visualizations**: n the `exploreTheDataSet` function, we utilized several visualizations to comprehensively analyze the dataset. We employed histograms with kernel density estimates (KDE) to understand the distribution of numerical features, identifying patterns such as skewness and outliers. Box plots were used to visualize the spread, central tendency, and detect outliers within numerical features. To explore relationships between pairs of numerical features, we created pair plots that display scatter plots for each feature pair and histograms along the diagonal. Additionally, we generated a heatmap of the correlation matrix to examine the correlations between numerical features, highlighting strong positive or negative relationships. These visualizations collectively provided a thorough understanding of the data, revealing key patterns and relationships that inform further data preprocessing and model development.

So, the `exploreTheDataSet` function provided a thorough initial exploration of the dataset, offering valuable insights into its structure, the presence of missing values, and the distribution of features. These insights are essential for guiding the subsequent steps in data preprocessing and feature engineering.

(In all the code snippets provided, there is a descriptive comment for each code line to show for what reason it was used)

```python
def exploreTheDataSet(dataset):

    # Display the first few rows of the dataset
    print(dataset.head())

    # Display the summary of the dataset
    print(dataset.info())

    # Describe the dataset to get summary statistics
    print(dataset.describe())

    # Get the column names
    columns = dataset.columns

    # Describe each feature
    for column in columns:
        print(f"Feature: {column}")
        print(dataset[column].value_counts())
        print("\n")

    # Visualize distributions with histograms
    numerical_features = dataset.select_dtypes(include=['float64', 'int64']).columns

    for feature in numerical_features:
        plt.figure(figsize=(10, 6))
        sns.histplot(dataset[feature], kde=True)
        plt.title(f'Distribution of {feature}')
        plt.show()

    # Visualize distributions with box plots
    for feature in numerical_features:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x=dataset[feature])
        plt.title(f'Box plot of {feature}')
        plt.show()


    # Scatter plot for numerical features
    sns.pairplot(dataset[numerical_features])
    plt.show()

    # Correlation matrix
    correlation_matrix = dataset[numerical_features].corr()

    plt.figure(figsize=(12, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Matrix')
    plt.show()
```

Outputs and results :

```
   Unnamed: 0                            CustomerID   Age  Gender  Tenure Service_Internet  ... TotalCharges StreamingMovies StreamingMusic OnlineSecurity  TechSupport  Churn
0           0  08729464-bde6-43bc-8f63-a357096feab1  56.0    Male      13              DSL  ...       931.49              No             No            Yes           No     No
1           1  af95bc95-baf4-4318-a21d-70d2ea3148b7  69.0    Male      13              DSL  ...      1448.46             Yes            Yes             No           No     No
2           2  1fe7eee6-2227-4400-9998-4d993f4a60fd  46.0    Male      60      Fiber optic  ...      6997.73             Yes            Yes             No           No     No
3           3  f736fe7b-1b44-4acd-84c2-21c4aef648be  32.0  Female      57      Fiber optic  ...      4452.13              No            Yes             No          Yes     No
4           4  4b40d12d-7633-4309-96b8-aee675ea20ae  60.0    Male      52      Fiber optic  ...      1569.73             Yes             No            Yes          Yes     No

[5 rows x 17 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3749 entries, 0 to 3748
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        3749 non-null   int64
 1   CustomerID        3749 non-null   object
 2   Age               3562 non-null   float64
 3   Gender            3749 non-null   object
 4   Tenure            3749 non-null   int64
 5   Service_Internet  3028 non-null   object
 6   Service_Phone     3749 non-null   object
 7   Service_TV        3749 non-null   object
 8   Contract          3749 non-null   object
 9   PaymentMethod     3562 non-null   object
 10  MonthlyCharges    3749 non-null   float64
 11  TotalCharges      3749 non-null   float64
 12  StreamingMovies   3749 non-null   object
 13  StreamingMusic    3749 non-null   object
 14  OnlineSecurity    3749 non-null   object
 15  TechSupport       3749 non-null   object
 16  Churn             3749 non-null   object
dtypes: float64(3), int64(2), object(12)
memory usage: 498.0+ KB
None
Unnamed: 0            0
CustomerID            0
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
        Unnamed: 0          Age       Tenure  MonthlyCharges  TotalCharges
count  3749.000000  3562.000000  3749.000000     3749.000000   3749.000000
mean   1874.000000    43.655531    36.264070       75.844318   2718.968266
std    1082.387408    14.914474    20.505528       73.062971   3211.879149
min       0.000000    18.000000     1.000000       20.000000     13.190000
25%     937.000000    31.000000    19.000000       44.570000   1076.240000
50%    1874.000000    44.000000    36.000000       69.590000   2132.260000
```

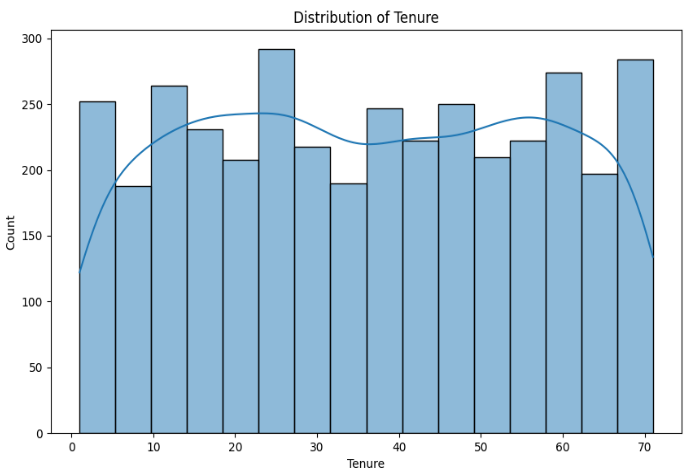
```
46.0    71
19.0    70
47.0    69
53.0    69
23.0    69
31.0    69
21.0    69
28.0    69
29.0    69
57.0    69
41.0    68
32.0    68
40.0    68
26.0    67
18.0    67
58.0    67
37.0    67
63.0    66
69.0    66
34.0    66
22.0    65
20.0    65
30.0    64
35.0    63
36.0    62
67.0    61
59.0    61
27.0    60
55.0    60
33.0    58
65.0    57
60.0    53
24.0    53
48.0    52
44.0    51
Name: count, dtype: int64


Feature: Gender
Gender
Male      1895
Female    1854
Name: count, dtype: int64


Feature: Tenure
Tenure
26    71
58    69
71    66
17    63
33    63
      ..
32    42
34    41
40    40
8     39
5     33
```

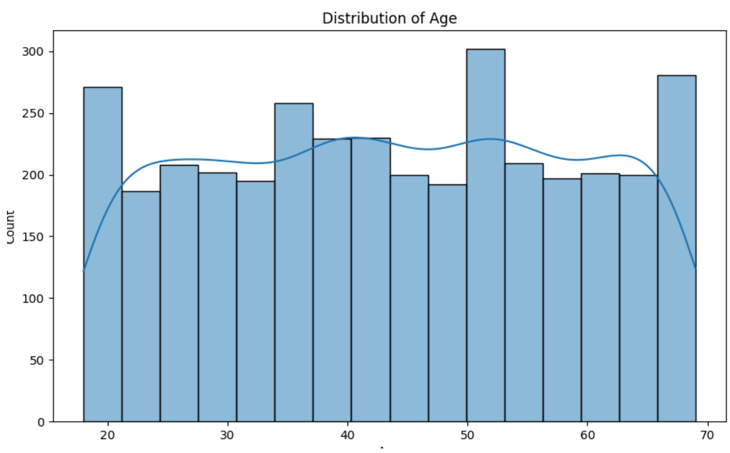
```
Feature: CustomerID
CustomerID
08729464-bde6-43bc-8f63-a357096feab1    1
578f1394-3626-4c86-8b28-41515fc06c21    1
71910967-4c63-43bf-9068-01c7c3718079    1
5bdfe923-70d3-4946-b301-84048129819c    1
f20f30a7-360a-462c-a5df-a7e71eae0f93    1
                                        ..
a790630d-8554-484c-843f-b09ea6b3b710    1
6c904f7d-87ea-42f0-ad6c-8a60fc61cee4    1
8fac7ee2-15a7-4932-b71e-072ae65ea586    1
2776537e-b18b-4a1d-abc0-2b49c55c353f    1
a6c038b7-b5c4-4813-9705-6a000a6a7d7f    1
Name: count, Length: 3749, dtype: int64


Feature: Age
Age
38.0    84
66.0    83
52.0    82
25.0    81
43.0    81
42.0    81
54.0    78
45.0    78
50.0    78
64.0    77
39.0    77
62.0    76
51.0    73
61.0    72
56.0    71
49.0    71
```

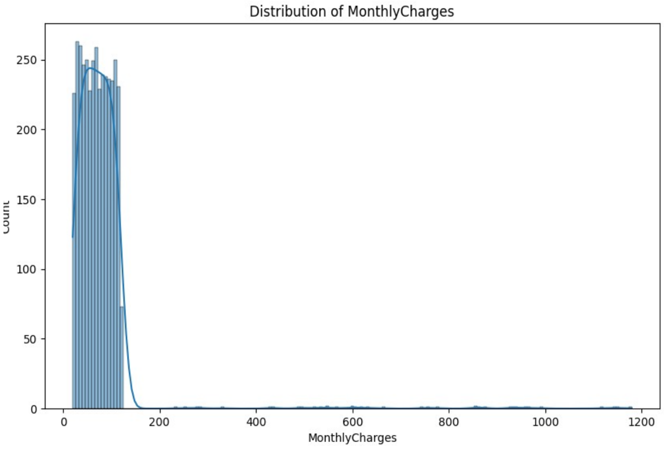
```
81.02     1
55.22     1
102.82    1
Name: count, Length: 3118, dtype: int64


Feature: TotalCharges
TotalCharges
7303.09    2
4002.23    2
3016.73    2
2741.22    2
2034.36    2
          ..
6831.20    1
2130.65    1
5756.27    1
285.36     1
2262.98    1
Name: count, Length: 3735, dtype: int64


Feature: StreamingMovies
StreamingMovies
Yes    1913
No     1836
Name: count, dtype: int64


Feature: StreamingMusic
StreamingMusic
No     1890
Yes    1859
Name: count, dtype: int64


Feature: OnlineSecurity
OnlineSecurity
No     2215
Yes    1534
Name: count, dtype: int64


Feature: TechSupport
TechSupport
No     2263
Yes    1486
Name: count, dtype: int64


Feature: Churn
Churn
No     3498
Yes     251
Name: count, dtype: int64

yahiaahmed@yahia:~/Documents/machine_learning_project$
```

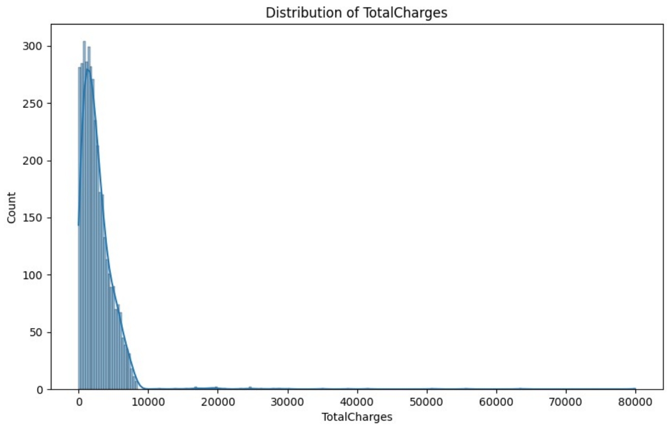
```
32    42
34    41
40    40
8     39
5     33
Name: count, Length: 71, dtype: int64


Feature: Service_Internet
Service_Internet
Fiber optic    1908
DSL            1120
Name: count, dtype: int64


Feature: Service_Phone
Service_Phone
Yes    2621
No     1128
Name: count, dtype: int64


Feature: Service_TV
Service_TV
Yes    2188
No     1561
Name: count, dtype: int64


Feature: Contract
Contract
Month-to-month    2183
Two year           792
One year           774
Name: count, dtype: int64


Feature: PaymentMethod
PaymentMethod
Electronic check    1400
Bank transfer        742
Mailed check         730
Credit card          690
Name: count, dtype: int64


Feature: MonthlyCharges
MonthlyCharges
96.12     4
75.44     4
103.10    4
50.98     4
32.64     4
         ..
52.43     1
53.27     1
81.02     1
55.22     1
```

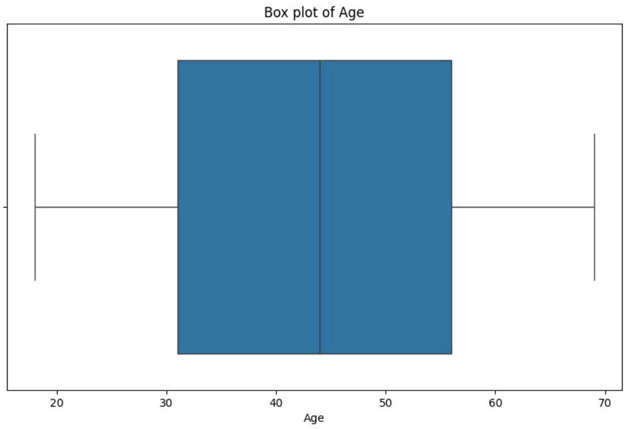
Visualization and charts (each one has a legend with the name of the feature):

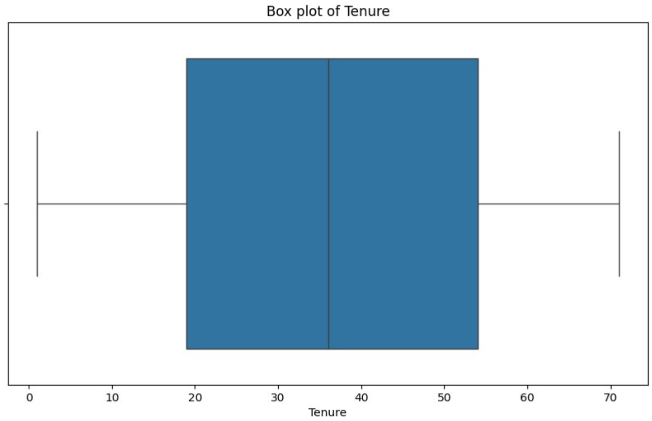


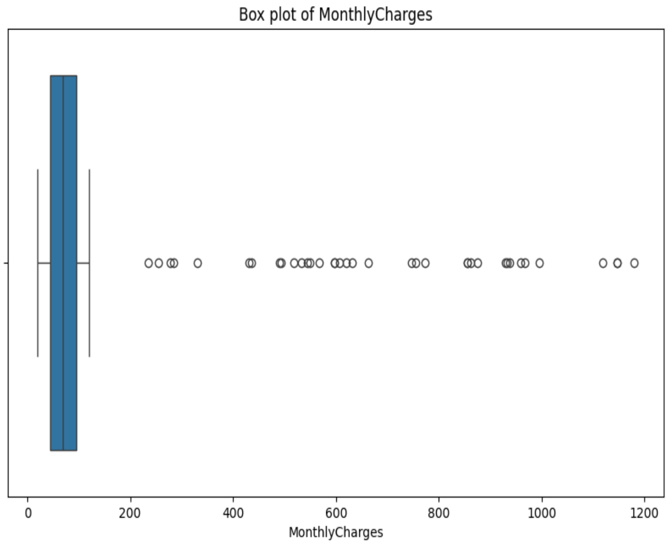


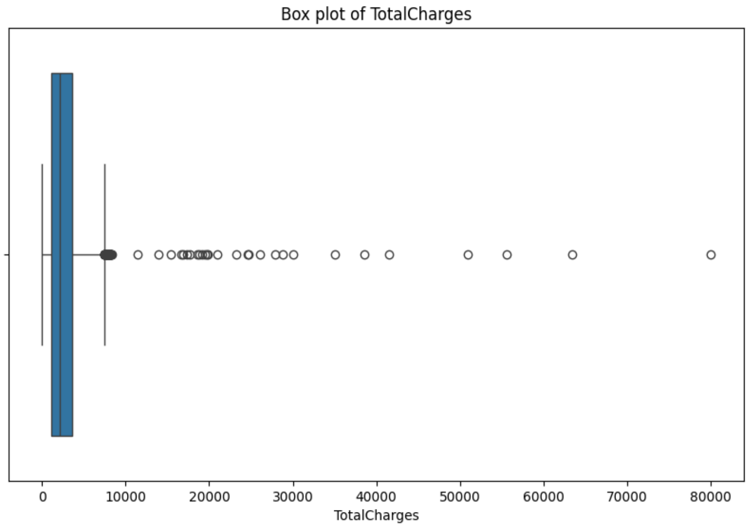

Distribution of MonthlyCharges
0
50
100
150
200
250
0
200
400
600
800
1000
1200
MonthlyCharges

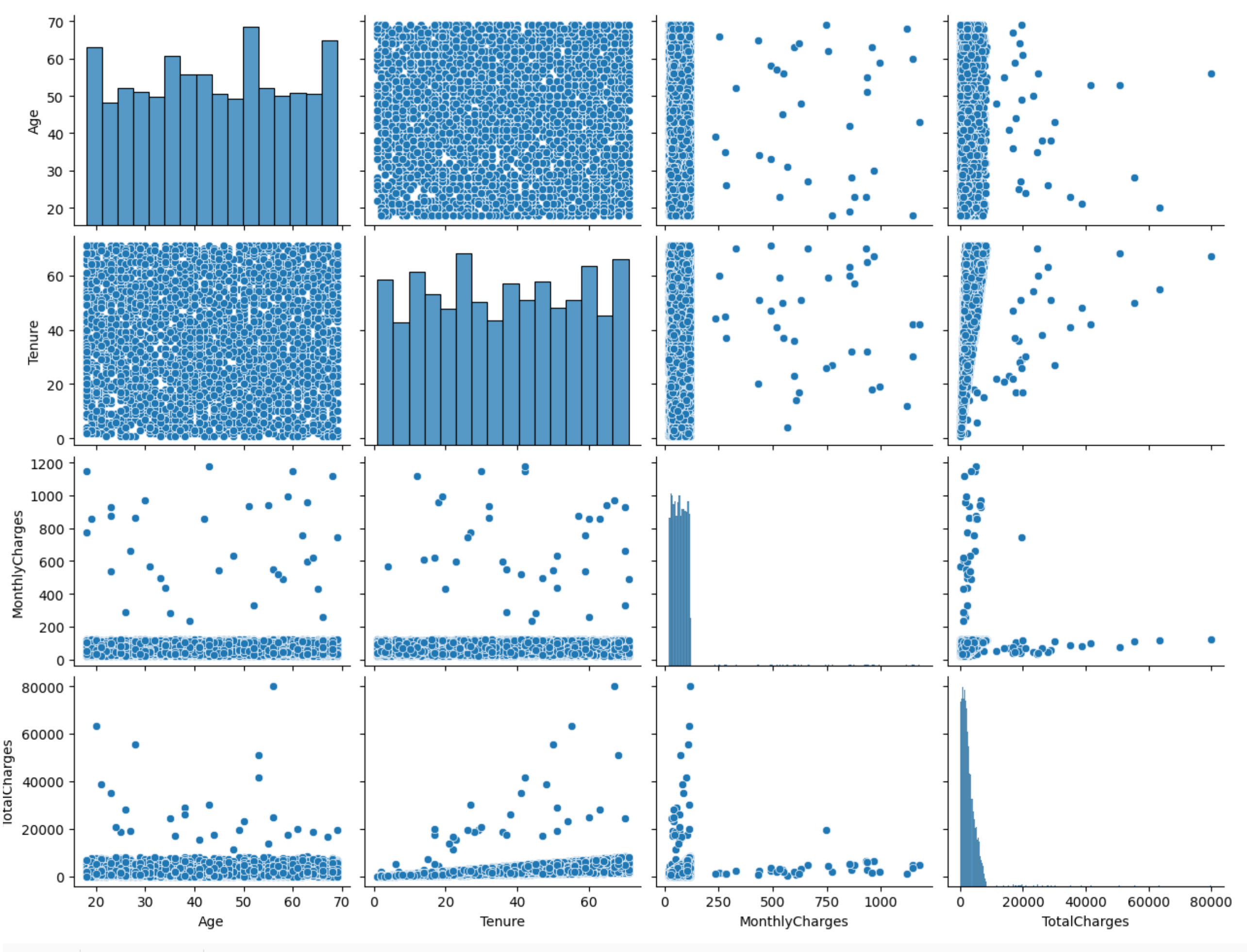

Distribution of TotalCharges
Count
0
50
100
150
200
250
300
0
10000
20000
30000
40000
50000
60000
70000
80000
TotalCharges


Box plot of Age
20
30
40
50
60
70
Age


Box plot of Tenure
0
10
20
30
40
50
60
70
Tenure


Box plot of MonthlyCharges
0
200
400
600
800
1000
1200
MonthlyCharges


Box plot of TotalCharges
0
10000
20000
30000
40000
50000
60000
70000
80000
TotalCharges


Age
Tenure
MonthlyCharges
TotalCharges
Age
Tenure
MonthlyCharges
TotalCharges

2- Preprocessing Phase:

In the preprocessing phase, I applied several steps to clean and prepare the dataset for further analysis and modeling. Here's a detailed description of the preprocessing steps implemented:

1. **Remove Unnamed First Column:**
    - **Purpose:** To eliminate any redundant or irrelevant columns, specifically those that are automatically generated by data export processes and typically do not contain useful information.
    - **Implementation:** The first column was checked if its name starts with 'Unnamed' and removed if true, ensuring only relevant data is retained.
2. **Handle Missing Values:**
    - **Purpose:** To address any missing data within the dataset, which can adversely affect model performance.
    - **Implementation:** I printed the count of missing values for each column using `dataset.isnull().sum()`, which informed my strategy for handling missing data. Numerical features were filled with the median value using `dataset[numerical_features].fillna(dataset[numerical_features].median(), inplace=True)`, and categorical features were filled with the mode using `dataset[categorical_features].fillna(dataset[categorical_features].mode()[0], inplace=True)`.
3. **Encode Categorical Variables:**
    - **Purpose:** To convert categorical variables into a numerical format that machine learning algorithms can interpret.
    - **Implementation:** One-hot encoding was applied to categorical features using `pd.get_dummies(dataset, columns=categorical_features, drop_first=True)`, resulting in binary columns for each category while avoiding multicollinearity by dropping the first category.
4. **Normalize Numerical Features:**
    - **Purpose:** To standardize the range of numerical features, ensuring that each feature contributes equally to the model.
    - **Implementation:** Numerical features were scaled using `StandardScaler` from `sklearn.preprocessing`. The scaler was fitted and applied to the numerical features, transforming them to have a mean of 0 and a standard deviation of 1.

These preprocessing steps were crucial in transforming the raw dataset into a clean and structured format, suitable for machine learning models. By addressing missing values, encoding categorical variables, and normalizing numerical features, I ensured that the dataset was ready for effective feature engineering and model training.

eprocess.py > preprocessTheDataSet

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

def preprocessTheDataSet(dataset):

    # Drop the 'Unnamed: 0' column if it exists
    if 'Unnamed: 0' in dataset.columns:
        dataset.drop(columns=['Unnamed: 0'], inplace=True)

    numerical_features = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']
    categorical_features = [
        'Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',
        'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',
        'OnlineSecurity', 'TechSupport'
    ]

    for feature in numerical_features:
        dataset[feature].fillna(dataset[feature].median(), inplace=True)

    for feature in categorical_features:
        dataset[feature].fillna(dataset[feature].mode()[0], inplace=True)

    # Treat outliers
    def cap_outliers(series):
        Q1 = series.quantile(0.25)
        Q3 = series.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        return series.clip(lower_bound, upper_bound)

    for feature in numerical_features:
        dataset[feature] = cap_outliers(dataset[feature])

    # Encode categorical variables
    dataset_encoded = pd.get_dummies(dataset, columns=categorical_features, drop_first=True)

    # Normalize numerical features
    scaler = StandardScaler()
    dataset_encoded[numerical_features] = scaler.fit_transform(dataset_encoded[numerical_features])

    # Save cleaned dataset
    cleaned_dataset_path = 'cleaned_data.csv'
    dataset_encoded.to_csv(cleaned_dataset_path, index=False)

    print(f"Cleaned dataset saved to '{cleaned_dataset_path}'")
```

And here is some data about the new dataset so we can make a comparison between the original one and the cleaned one:

<u>the new dataset with be attached in the public github repository</u>

3- Exploratory Data Analysis (EDA in depth)

In the in-depth exploration phase, I conducted a comprehensive analysis of the cleaned dataset to uncover deeper insights and understand the key factors influencing customer churn. Here's a detailed description of the steps and visualizations implemented:

1. **Initial Data Display:**
    - **Purpose:** To get a quick overview of the cleaned data.
    - **Implementation:** Displayed the first few rows using `print(dataset.head())`.
2. **Data Summary:**
    - **Purpose:** To understand the structure and completeness of the cleaned data.
    - **Implementation:** Used `dataset.info()` to display the summary, including data types and non-null counts, and `dataset.isnull().sum()` to count any remaining missing values.
3. **Summary Statistics:**
    - **Purpose:** To get statistical summaries of numerical features in the cleaned dataset.
    - **Implementation:** Used `dataset.describe()` to get count, mean, standard deviation, min, and max values for numerical features.
4. **Feature Descriptions:**
    - **Purpose:** To understand the distribution of values in each feature of the cleaned dataset.
    - **Implementation:** Printed unique value counts for each feature using a loop and `dataset[column].value_counts()`.
5. **Visualizations:**
    - **Histograms and Box Plots for Numerical Features:**
        - **Purpose:** To visualize the distribution, central tendency, spread, and detect outliers in the cleaned data.
        - **Implementation:** For each numerical feature, created a combined plot of histogram with KDE and box plot side-by-side using `sns.histplot` and `sns.boxplot`.
    - **Bar Charts for Categorical Features:**
        - **Purpose:** To visualize the distribution of categories in the cleaned data.
        - **Implementation:** Created bar charts for each categorical feature using `sns.countplot`.
6. **Scatter Plots for Numerical Features:**
    - **Purpose:** To explore relationships and correlations between pairs of numerical features in the cleaned dataset.
    - **Implementation:** Created pair plots using `sns.pairplot`.

7. **Correlation Matrix Heatmap:**
    - **Purpose:** To examine the correlations between numerical features in the cleaned dataset and identify strong positive or negative relationships.
    - **Implementation:** Created a heatmap of the correlation matrix using `sns.heatmap`.
8. **Hypothesis Testing:**
    - **T-tests for Numerical Features:**
        - **Purpose:** To compare the means of numerical features between two groups defined by a binary target variable (e.g., Churn).
        - **Implementation:** Performed t-tests for each numerical feature using `ttest_ind` from `scipy.stats`.
    - **Chi-square Tests for Categorical Features:**
        - **Purpose:** To examine the independence between categorical features and a binary target variable.
        - **Implementation:** Performed chi-square tests using `chi2_contingency` from `scipy.stats`.

```python
def exploreInDepthDataSet(dataset):

    # Print the count of missing values in each column
    print(dataset.isnull().sum())

    # Describe the dataset to get summary statistics
    print(dataset.describe())

    # Get the column names
    columns = dataset.columns

    # Describe each feature
    for column in columns:
        print(f"Feature: {column}")
        print(dataset[column].value_counts())
        print("\n")

    # Visualize distributions with histograms
    numerical_features = dataset.select_dtypes(include=['float64', 'int64']).columns
    categorical_features = dataset.select_dtypes(include=['object', 'bool']).columns

    # Visualize numerical feature distributions with histograms and box plots
    for feature in numerical_features:
        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

        sns.histplot(dataset[feature], kde=True, ax=axes[0])
        axes[0].set_title(f'Distribution of {feature}')

        sns.boxplot(x=dataset[feature], ax=axes[1])
        axes[1].set_title(f'Box plot of {feature}')

        plt.show()

    # Visualize categorical feature distributions with bar charts
    for feature in categorical_features:
        plt.figure(figsize=(10, 6))
        sns.countplot(y=dataset[feature], order=dataset[feature].value_counts().index)
        plt.title(f'Distribution of {feature}')
        plt.show()

    # Pair plot for numerical features
    sns.pairplot(dataset[numerical_features])
    plt.show()

    # Correlation matrix heatmap
    correlation_matrix = dataset[numerical_features].corr()
    plt.figure(figsize=(12, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Matrix')
    plt.show()

    # Hypothesis testing
    target_variable = 'Churn'  # Example target variable for testing

    if target_variable in dataset.columns:
        for feature in numerical_features:
            if feature != target_variable and dataset[target_variable].nunique() == 2:
                groups = dataset[target_variable].unique()
                group1 = dataset[dataset[target_variable] == groups[0]][feature]
                group2 = dataset[dataset[target_variable] == groups[1]][feature]
                t_stat, p_val = ttest_ind(group1, group2, nan_policy='omit')
                print(f'T-test for {feature} by {target_variable}: t-stat={t_stat:.4f}, p-value={p_val:.4f}')

        for feature in categorical_features:
            if feature != target_variable:
                contingency_table = pd.crosstab(dataset[feature], dataset[target_variable])
                chi2, p_val, dof, ex = chi2_contingency(contingency_table)
                print(f'Chi-square test for {feature} by {target_variable}: chi2={chi2:.4f}, p-value={p_val:.4f}')
```

Dataset Insights:

```
                            CustomerID       Age    Tenure  MonthlyCharges  TotalCharges  ... PaymentMethod_Mailed check  StreamingMovies_Yes  StreamingMusic_Yes  OnlineSecurity_Yes  TechSupport_Yes
0  08729464-bde6-43bc-8f63-a357096feab1  0.848058 -1.134678        0.039241     -0.865123  ...                       True                False               False                True            False
1  af95bc95-baf4-4318-a21d-70d2ea3148b7  1.742395 -1.134678        1.323960     -0.590069  ...                       True                 True                True               False            False
2  1fe7eee6-2227-4400-9998-4d993f4a60fd  0.160106  1.157693        1.512841      2.362417  ...                       True                 True                True               False            False
3  f736fe7b-1b44-4acd-84c2-21c4aef648be -0.803026  1.011371        0.245532      1.008032  ...                      False                False                True               False             True
4  4b40d12d-7633-4309-96b8-aee675ea20ae  1.123239  0.767502       -1.325629     -0.525548  ...                      False                 True               False                True             True

[5 rows x 19 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3749 entries, 0 to 3748
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   CustomerID                    3749 non-null   object
 1   Age                           3749 non-null   float64
 2   Tenure                        3749 non-null   float64
 3   MonthlyCharges                3749 non-null   float64
 4   TotalCharges                  3749 non-null   float64
 5   Churn                         3749 non-null   object
 6   Gender_Male                   3749 non-null   bool
 7   Service_Internet_Fiber optic  3749 non-null   bool
 8   Service_Phone_Yes             3749 non-null   bool
 9   Service_TV_Yes                3749 non-null   bool
 10  Contract_One year             3749 non-null   bool
 11  Contract_Two year             3749 non-null   bool
 12  PaymentMethod_Credit card     3749 non-null   bool
 13  PaymentMethod_Electronic check  3749 non-null   bool
 14  PaymentMethod_Mailed check    3749 non-null   bool
 15  StreamingMovies_Yes           3749 non-null   bool
 16  StreamingMusic_Yes            3749 non-null   bool
 17  OnlineSecurity_Yes            3749 non-null   bool
 18  TechSupport_Yes               3749 non-null   bool
dtypes: bool(13), float64(4), object(2)
memory usage: 223.5+ KB
None
CustomerID                       0
Age                              0
Tenure                           0
MonthlyCharges                   0
TotalCharges                     0
Churn                            0
Gender_Male                      0
Service_Internet_Fiber optic     0
Service_Phone_Yes                0
Service_TV_Yes                   0
Contract_One year                0
Contract_Two year                0
PaymentMethod_Credit card        0
PaymentMethod_Electronic check   0
PaymentMethod_Mailed check       0
StreamingMovies_Yes              0
StreamingMusic_Yes               0
OnlineSecurity_Yes               0
TechSupport_Yes                  0
dtype: int64
                Age        Tenure  MonthlyCharges  TotalCharges
count  3.749000e+03  3.749000e+03    3.749000e+03  3.749000e+03
mean  -3.695808e-17  1.241412e-16   -7.268422e-16 -1.326700e-16
std    1.000133e+00  1.000133e+00    1.000133e+00  1.000133e+00
min   -1.766159e+00 -1.719964e+00   -1.664958e+00 -1.353704e+00
25%   -8.030264e-01 -8.420351e-01   -8.578612e-01 -7.881087e-01
50%    2.251579e-02 -1.287973e-02   -3.598265e-02 -2.262539e-01
```

```
5bdfe923-70d3-4946-b301-84048129819c    1
f20f30a7-360a-462c-a5df-a7e71eae0f93    1
                                       ..
a790630d-8554-484c-843f-b09ea6b3b710    1
6c904f7d-87ea-42f0-ad6c-8a60fc61cee4    1
8fac7ee2-15a7-4932-b71e-072ae65ea586    1
2776537e-b18b-4a1d-abc0-2b49c55c353f    1
a6c038b7-b5c4-4813-9705-6a000a6a7d7f    1
Name: count, Length: 3749, dtype: int64


Feature: Age
Age
 0.022516    238
-0.390255     84
 1.536010     83
 0.572877     82
-0.115075     81
-1.284593     81
-0.046279     81
 0.091311     78
 0.435287     78
 0.710468     78
-0.321460     77
 1.398419     77
 1.260829     76
 0.504082     73
 1.192034     72
 1.673600     71
 0.366492     71
 0.848058     71
 0.160106     71
-1.697364     70
-1.078207     69
-1.422183     69
-0.871822     69
-1.559773     69
 0.641672     69
 0.916853     69
-1.009412     69
 0.228901     69
-0.183870     68
-0.803026     68
-0.252665     68
-1.766159     67
 0.985648     67
-0.459050     67
-1.215797     67
-0.665436     66
 1.329624     66
 1.742395     66
-1.490978     65
-1.628569     65
-0.940617     64
-0.596641     63
-0.527846     62
 1.604805     61
 1.054443     61
-1.147002     60
 0.779263     60
-0.734231     58
 1.467215     57
```

```
Feature: Tenure
Tenure
-0.500618    71
 1.060145    69
 1.694205    66
-0.939583    63
-0.159201    63
             ..
-0.207975    42
-0.110427    41
 0.182216    40
-1.378547    39
-1.524869    33
Name: count, Length: 71, dtype: int64


Feature: MonthlyCharges
MonthlyCharges
 3.327905    37
 0.835498     4
 1.064783     4
 0.156183     4
-0.647300     4
             ..
-0.121061     1
 0.409448     1
 0.559896     1
 0.167680     1
 1.055585     1
Name: count, Length: 3082, dtype: int64


Feature: TotalCharges
TotalCharges
 2.595021    73
 2.524883     2
-0.937556     2
 0.097743     2
-0.278342     2
             ..
-0.351174     1
-1.002684     1
 1.025642     1
 2.083250     1
-0.156704     1
Name: count, Length: 3663, dtype: int64


Feature: Churn
Churn
No     3498
Yes     251
Name: count, dtype: int64


Feature: Gender_Male
Gender_Male
True     1895
False    1854
Name: count, dtype: int64
```

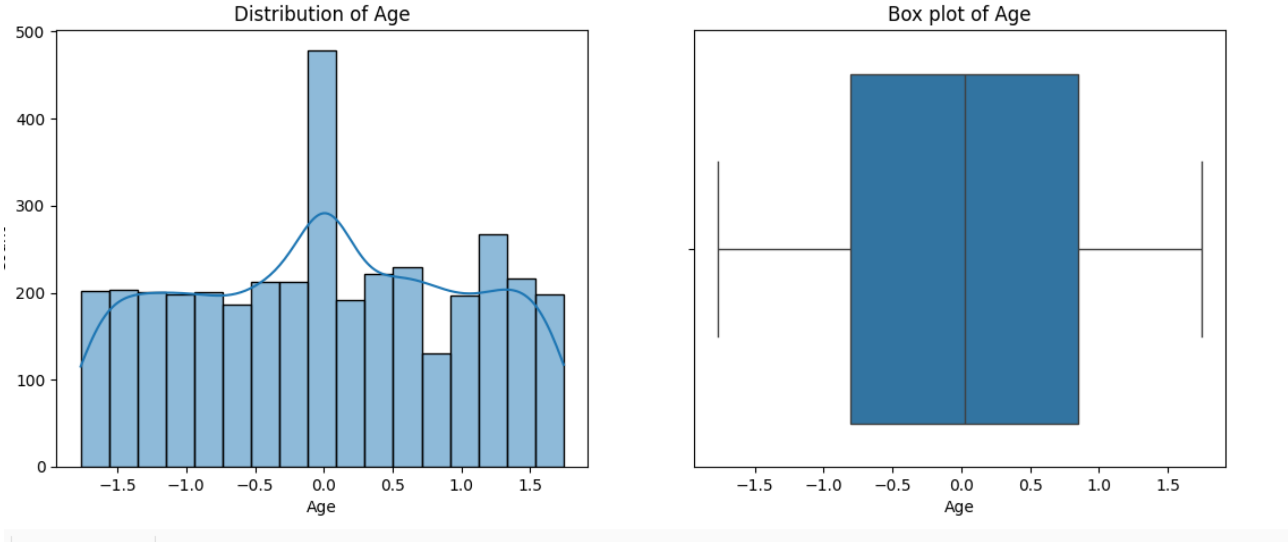
```
True      774
Name: count, dtype: int64


Feature: Contract_Two year
Contract_Two year
False    2957
True      792
Name: count, dtype: int64


Feature: PaymentMethod_Credit card
PaymentMethod_Credit card
False    3059
True      690
Name: count, dtype: int64


Feature: PaymentMethod_Electronic check
PaymentMethod_Electronic check
False    2162
True     1587
Name: count, dtype: int64


Feature: PaymentMethod_Mailed check
PaymentMethod_Mailed check
False    3019
True      730
Name: count, dtype: int64


Feature: StreamingMovies_Yes
StreamingMovies_Yes
True     1913
False    1836
Name: count, dtype: int64


Feature: StreamingMusic_Yes
StreamingMusic_Yes
False    1890
True     1859
Name: count, dtype: int64


Feature: OnlineSecurity_Yes
OnlineSecurity_Yes
False    2215
True     1534
Name: count, dtype: int64


Feature: TechSupport_Yes
TechSupport_Yes
False    2263
True     1486
Name: count, dtype: int64


T-test for Age by Churn: t-stat=0.2106, p-value=0.8332
T-test for Tenure by Churn: t-stat=26.1583, p-value=0.0000
```

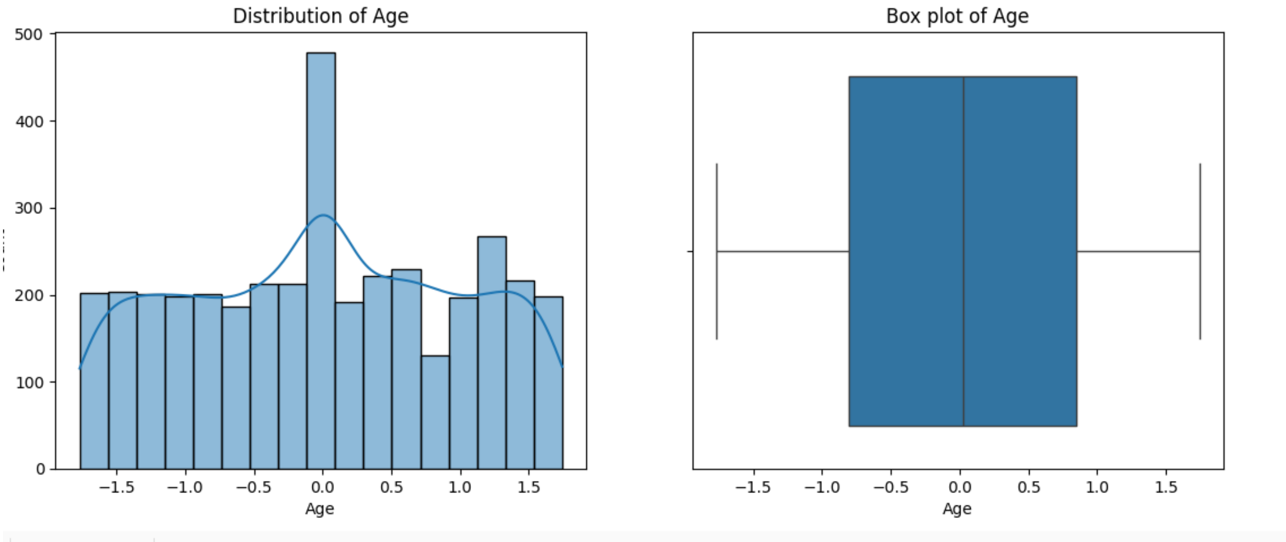
hypothesis testing results

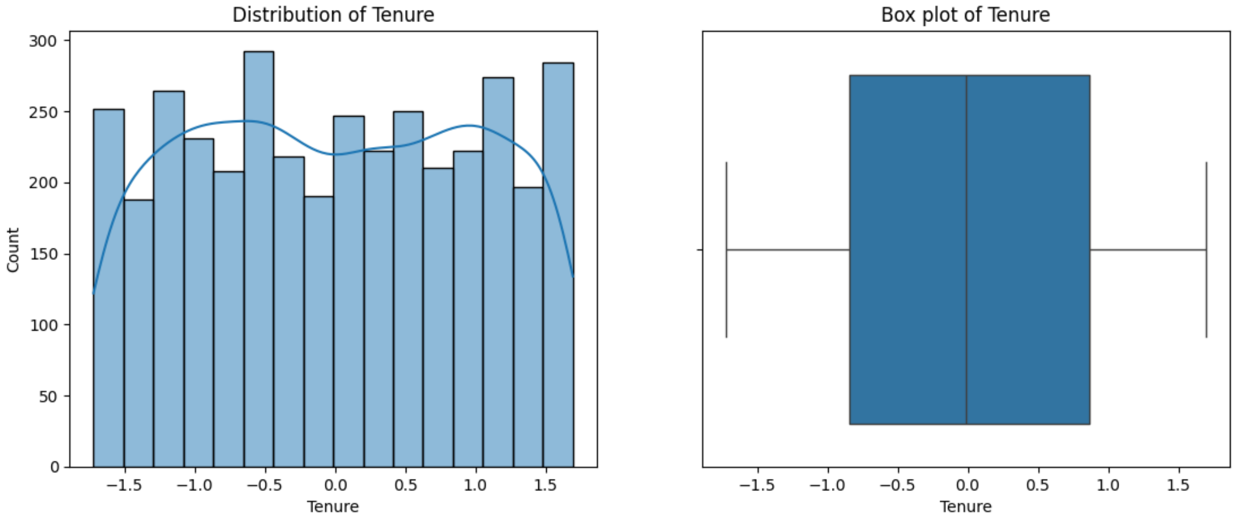
such that T-score for numerical values and Chi-Square in the categorical ones:

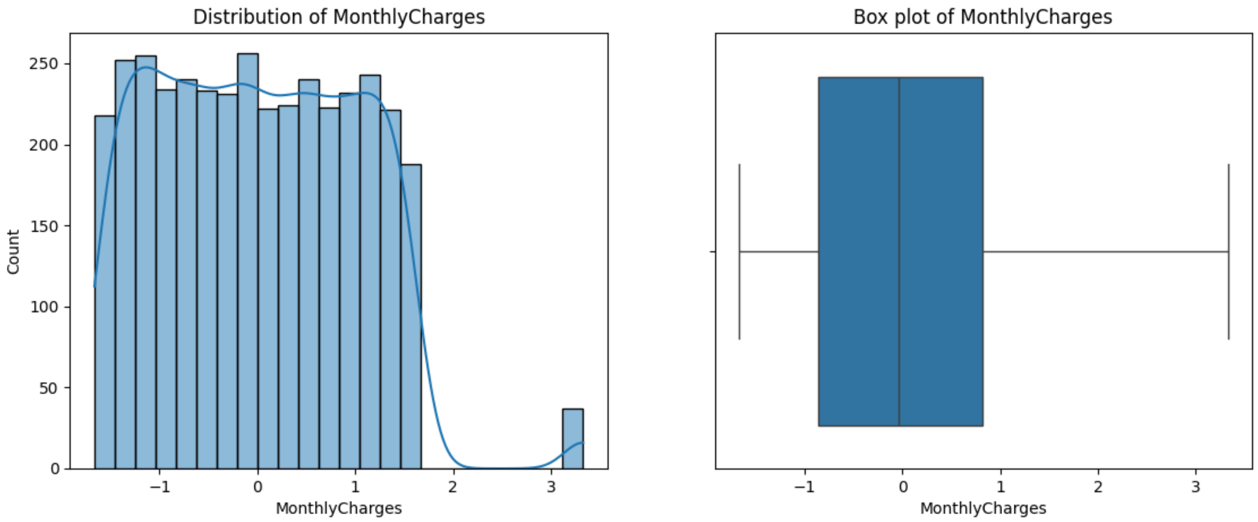
```
T-test for Age by Churn: t-stat=0.2106, p-value=0.8332
T-test for Tenure by Churn: t-stat=26.1583, p-value=0.0000
T-test for MonthlyCharges by Churn: t-stat=-13.5928, p-value=0.0000
T-test for TotalCharges by Churn: t-stat=17.6297, p-value=0.0000
Chi-square test for CustomerID by Churn: chi2=3749.0000, p-value=0.4923
Chi-square test for Gender_Male by Churn: chi2=0.1179, p-value=0.7313
Chi-square test for Service_Internet_Fiber optic by Churn: chi2=0.0468, p-value=0.82
Chi-square test for Service_Phone_Yes by Churn: chi2=1.2924, p-value=0.2556
Chi-square test for Service_TV_Yes by Churn: chi2=0.0695, p-value=0.7920
Chi-square test for Contract_One year by Churn: chi2=1.0411, p-value=0.3076
Chi-square test for Contract_Two year by Churn: chi2=0.0071, p-value=0.9330
Chi-square test for PaymentMethod_Credit card by Churn: chi2=0.1509, p-value=0.6977
Chi-square test for PaymentMethod_Electronic check by Churn: chi2=0.0000, p-value=1.
Chi-square test for PaymentMethod_Mailed check by Churn: chi2=0.3101, p-value=0.5776
Chi-square test for StreamingMovies_Yes by Churn: chi2=0.0425, p-value=0.8366
Chi-square test for StreamingMusic_Yes by Churn: chi2=0.2682, p-value=0.6046
Chi-square test for OnlineSecurity_Yes by Churn: chi2=0.4782, p-value=0.4893
Chi-square test for TechSupport_Yes by Churn: chi2=1.1452, p-value=0.2846
yahiaahmed@yahia:~/Documents/machine_learning_project$
```

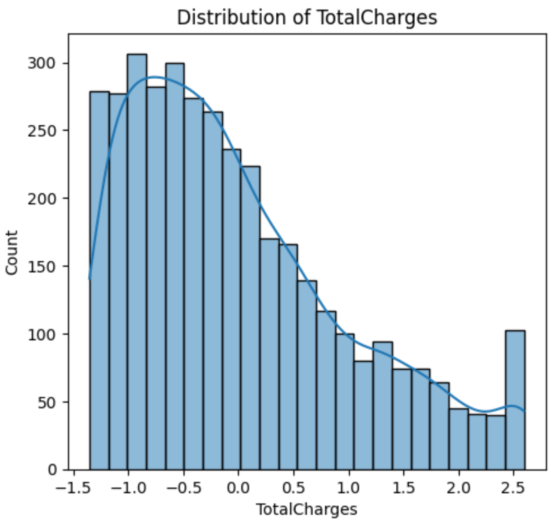
Charts and visulaization:

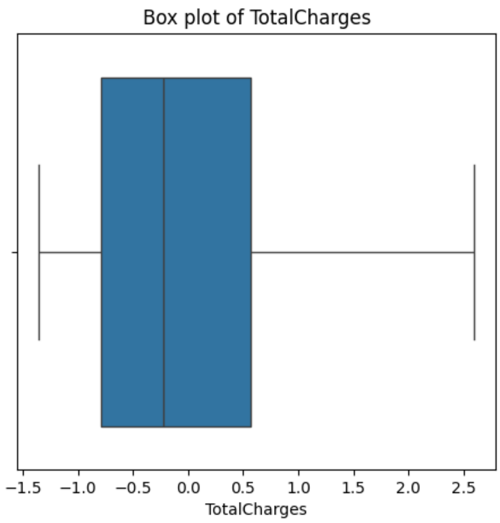

Distribution of Age
500
400
300
200
100
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age
Box plot of Age
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age

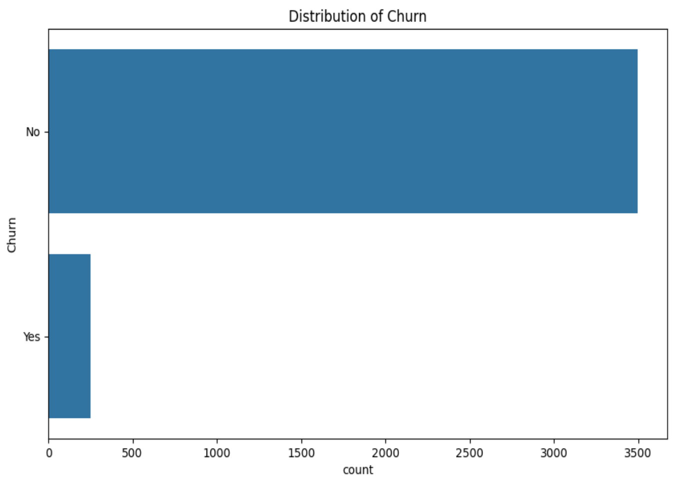

Distribution of Age
500
400
300
200
100
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age
Box plot of Age
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age

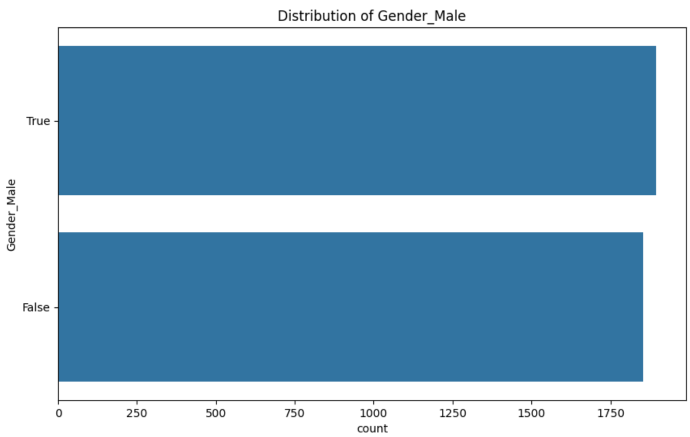

Distribution of Tenure
Count
Tenure
Box plot of Tenure
Tenure

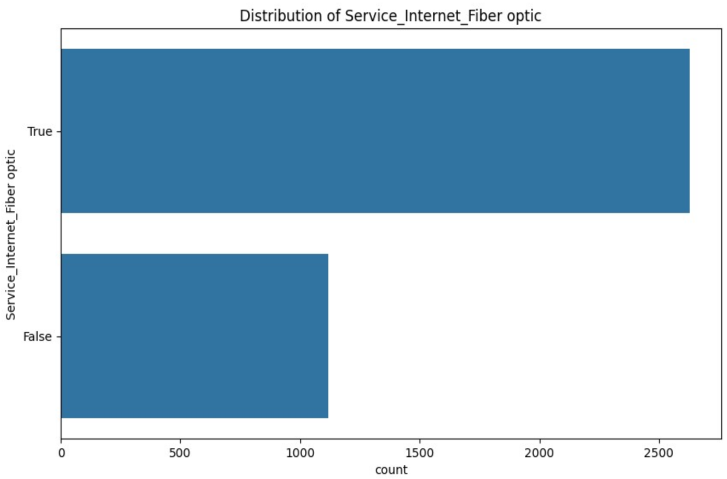

Distribution of MonthlyCharges
Count
MonthlyCharges
Box plot of MonthlyCharges
MonthlyCharges

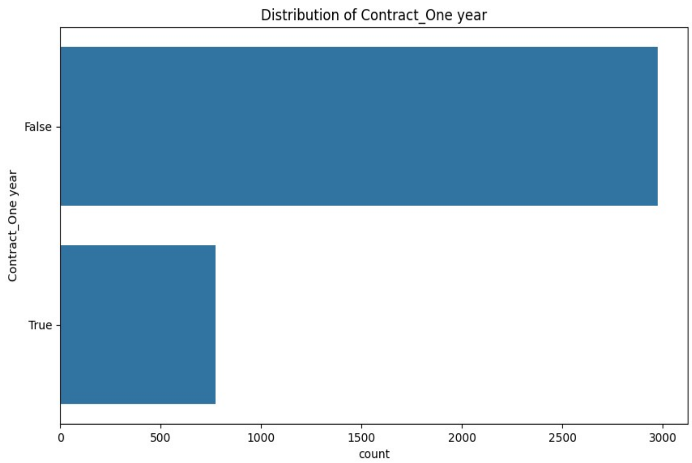

Distribution of TotalCharges
Count
TotalCharges

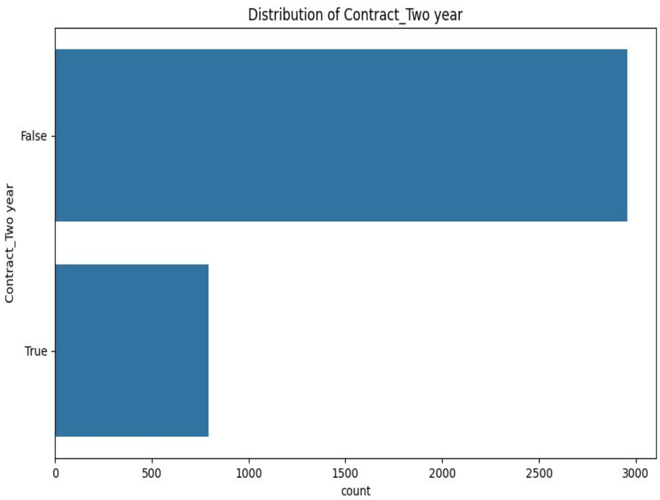

Box plot of TotalCharges
TotalCharges

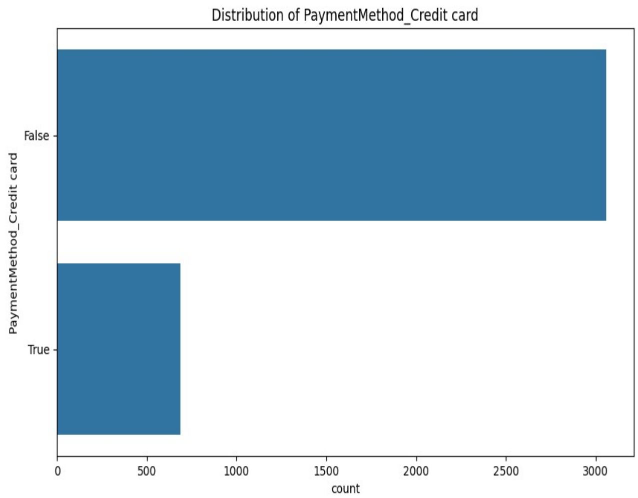

Distribution of Churn
Churn
No
Yes
count

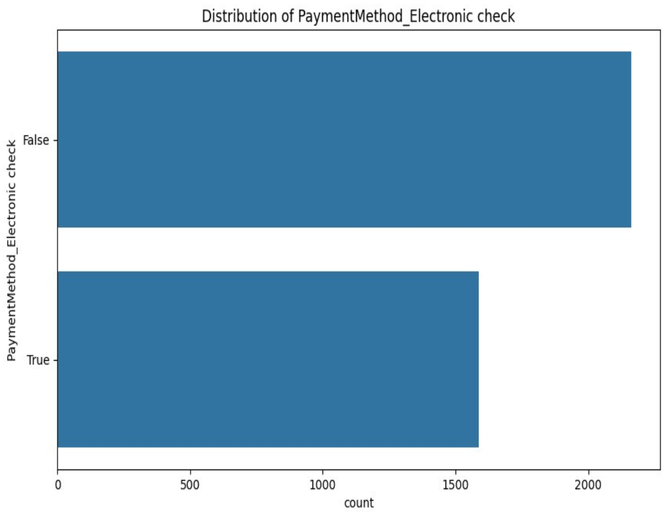

Distribution of Gender_Male
Gender_Male
True
False
count

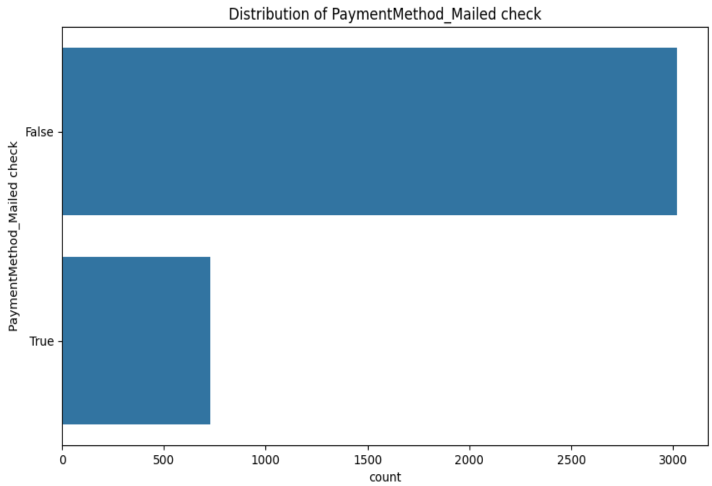

Distribution of Service_Internet_Fiber optic
Service_Internet_Fiber optic
True
False
count

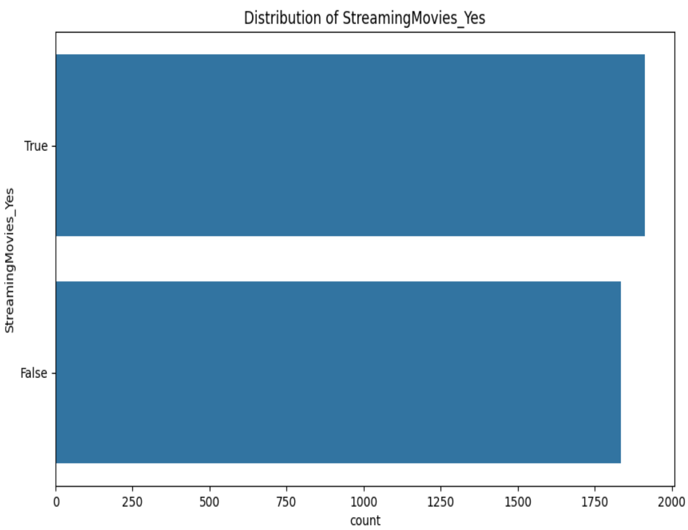

Distribution of Contract_One year
Contract_One year
False
True
count

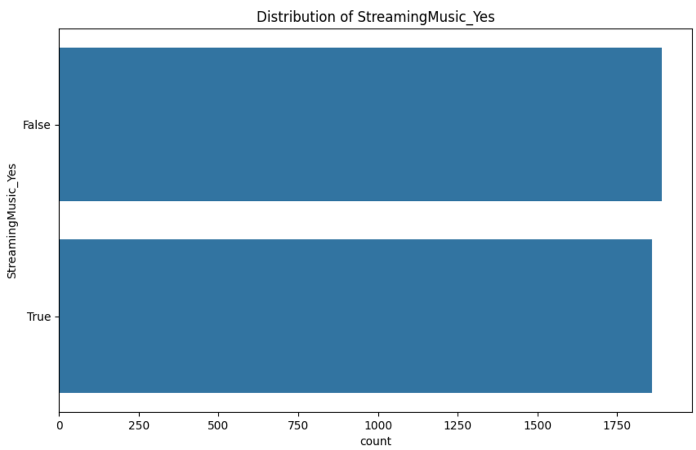

Distribution of Contract_Two year
Contract_Two year
False
True
0
500
1000
1500
2000
2500
3000
count

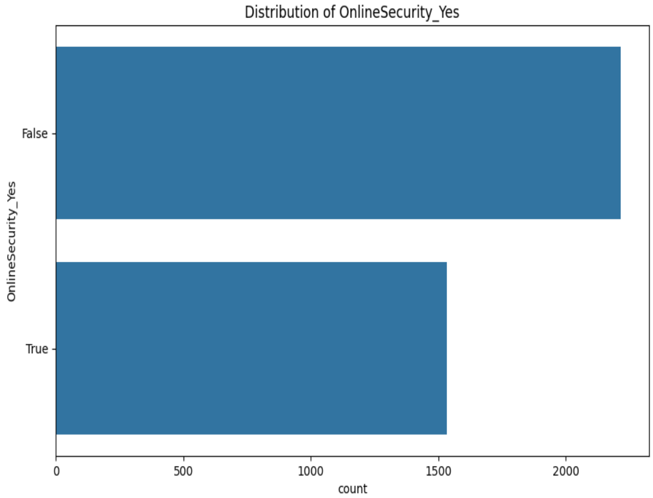

Distribution of PaymentMethod_Credit card
PaymentMethod_Credit card
False
True
0
500
1000
1500
2000
2500
3000
count

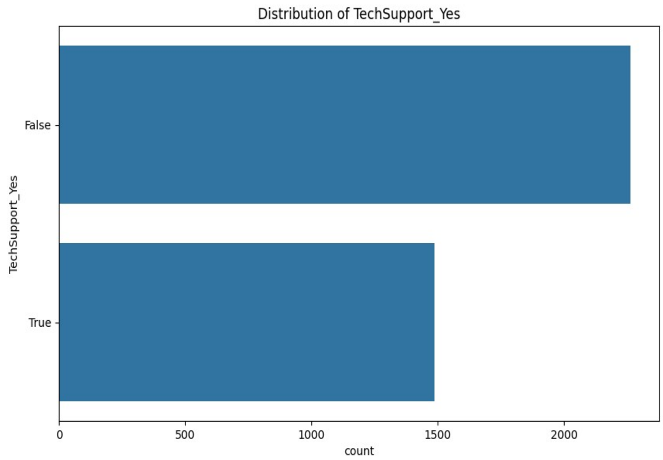

Distribution of PaymentMethod_Electronic check
PaymentMethod_Electronic check
False
True
0
500
1000
1500
2000
count

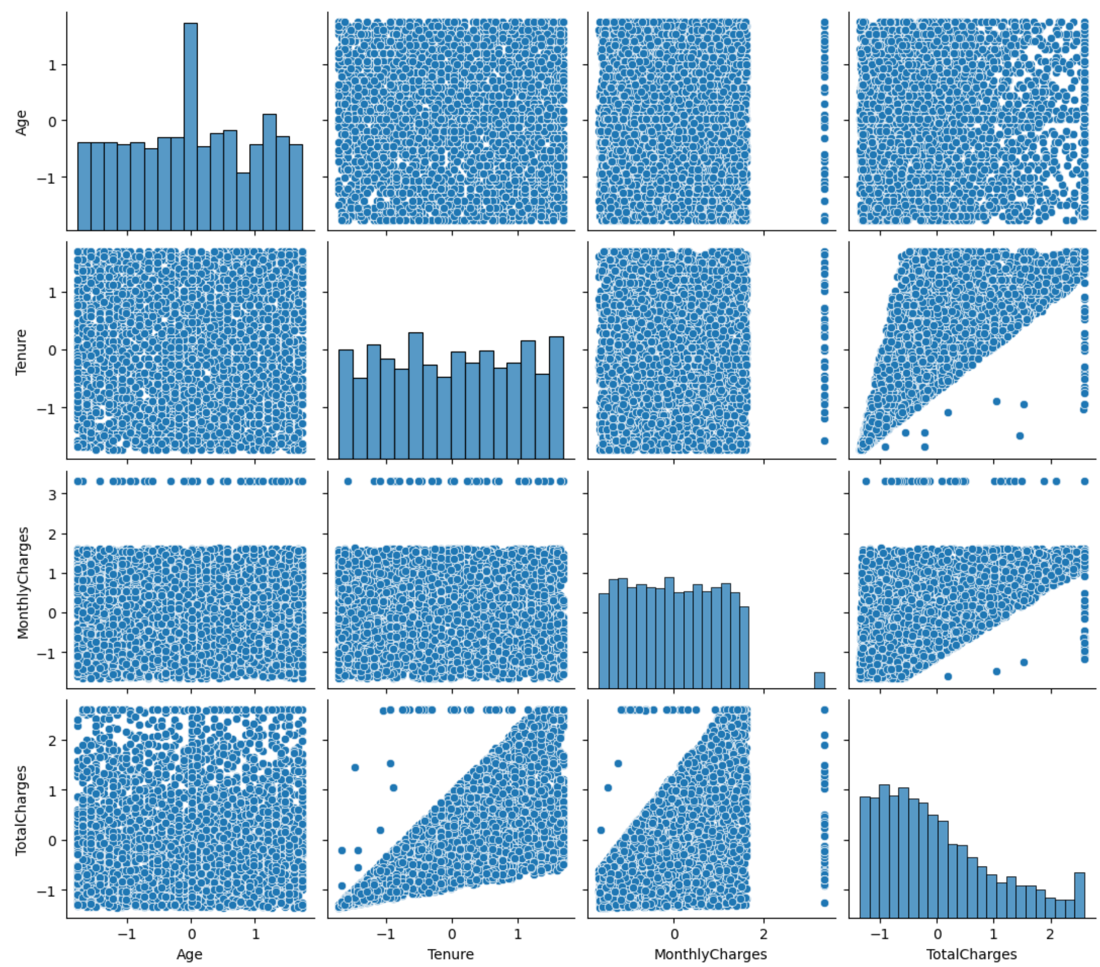

Distribution of PaymentMethod_Mailed check
PaymentMethod_Mailed check
False
True
0
500
1000
1500
2000
2500
3000
count


Distribution of StreamingMovies_Yes
StreamingMovies_Yes
True
False
0
250
500
750
1000
1250
1500
1750
2000
count


Distribution of StreamingMusic_Yes
StreamingMusic_Yes
False
True
0
250
500
750
1000
1250
1500
1750
count


Distribution of OnlineSecurity_Yes
False
True
OnlineSecurity_Yes
0
500
1000
1500
2000
count


Distribution of TechSupport_Yes
False
True
TechSupport_Yes
0
500
1000
1500
2000
count


Age
Tenure
MonthlyCharges
TotalCharges

4- Feature Engineering

In the feature engineering phase, I created new features to improve model performance by providing additional relevant information derived from existing features. Here's a detailed description of the steps implemented:

1. **Load the Cleaned Dataset:**
    - **Purpose:** To start with a clean and preprocessed dataset.
    - **Implementation:** Loaded the cleaned dataset from a CSV file using `pd.read_csv('cleaned_data.csv')`.
2. **Verify Dataset Columns:**
    - **Purpose:** To ensure the dataset has the expected columns before proceeding with feature engineering.
    - **Implementation:** Printed the column names to verify the dataset structure.
3. **Feature Engineering: Create New Features:**
    - **Total Service Usage:**
        - **Purpose:** To represent the total number of services a customer subscribes to, providing a measure of customer engagement.
        - **Implementation:** Created a new feature `TotalServices` by summing binary indicators for internet, phone, and TV services.
    - **Monthly Charges per Service:**
        - **Purpose:** To identify customers who might be paying disproportionately higher amounts, indicating potential dissatisfaction or high-value customers.
        - **Implementation:** Created a new feature `MonthlyChargesPerService` by dividing monthly charges by the total number of services. Replaced infinite values with 0 and filled missing values with 0 to handle cases where the total number of services is zero.
    - **Tenure per Service:**
        - **Purpose:** To provide insights into customer loyalty by showing the average tenure per service.
        - **Implementation:** Created a new feature `TenurePerService` by dividing tenure by the total number of services. Replaced infinite values with 0 and filled missing values with 0 to handle cases where the total number of services is zero.
4. **Normalize Numerical Features:**
    - **Purpose:** To standardize the range of numerical features, ensuring each feature contributes equally to the model.
    - **Implementation:** Scaled the numerical features to have a mean of 0 and a standard deviation of 1 using `StandardScaler` from `sklearn.preprocessing`.
5. **Save the Enhanced featured Dataset:**

- **Purpose:** To save the dataset with the newly created features for further analysis and model training.
- **Implementation:** Saved the enhanced dataset to a new CSV file using `dataset.to_csv('featured_data.csv', index=False)`.

Code Snippet:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

def featureEngineeringDataSet():
    # Load the cleaned dataset
    dataset = pd.read_csv('cleaned_data.csv')

    # Verify the columns in the dataset
    print("Dataset columns:", dataset.columns)

    # Feature Engineering: Create new features




    # 1. Total Service Usage
    # Combines service-related columns to create a feature representing the total number of services a cus
    dataset['TotalServices'] = (
        dataset['Service_Internet_Fiber optic'].astype(int) +
        dataset['Service_Phone_Yes'].astype(int) +
        dataset['Service_TV_Yes'].astype(int)
    )

    # 2. Monthly Charges per Service
    # Calculates the average monthly charges per service. This helps in identifying customers who might be
    dataset['MonthlyChargesPerService'] = dataset['MonthlyCharges'] / dataset['TotalServices']
    dataset['MonthlyChargesPerService'].replace([float('inf'), -float('inf')], 0, inplace=True)
    dataset['MonthlyChargesPerService'].fillna(0, inplace=True)

    # 3. Tenure per Service
    # Calculates the average tenure per service. This provides insights into customer loyalty.
    dataset['TenurePerService'] = dataset['Tenure'] / dataset['TotalServices']
    dataset['TenurePerService'].replace([float('inf'), -float('inf')], 0, inplace=True)
    dataset['TenurePerService'].fillna(0, inplace=True)

    # Normalize the numerical features (including the new features)
    numerical_features = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges', 'MonthlyChargesPerService', '
    scaler = StandardScaler()
    dataset[numerical_features] = scaler.fit_transform(dataset[numerical_features])

    # Save the enhanced dataset to a new CSV file
    enhanced_dataset_path = 'featured_data.csv'
    dataset.to_csv(enhanced_dataset_path, index=False)

    print(f"featured_data dataset saved to {enhanced_dataset_path}")
```

and a result :

all the features including the new ones:

```
CustomerID
Age
Tenure
MonthlyCharges
TotalCharges
Churn
Gender_Male
Service_Internet_Fiber optic
Service_Phone_Yes
Service_TV_Yes
Contract_One year
Contract_Two year
PaymentMethod_Credit card
PaymentMethod_Electronic check
PaymentMethod_Mailed check
StreamingMovies_Yes
StreamingMusic_Yes
OnlineSecurity_Yes
TechSupport_Yes
TotalServices
MonthlyChargesPerService
TenurePerService
dtype: int64
```

Here, info about the three new features that were engineering to impact in a better future model.

```
Feature: TotalServices
TotalServices
2    1675
3    1070
1     878
0     126
Name: count, dtype: int64


Feature: MonthlyChargesPerService
MonthlyChargesPerService
 0.004090    126
 2.738729     21
 1.827183      9
 5.473368      5
 0.132430      4
            ...
-1.343271      1
-0.578832      1
-0.317403      1
 1.078531      1
 0.871496      1
Name: count, Length: 3341, dtype: int64


Feature: TenurePerService
TenurePerService
-0.001678    126
 0.860201     37
 0.820549     34
 0.067156     31
 1.336028     31
            ...
-0.101924      7
 1.325557      7
-2.401754      6
 0.056685      5
 0.294598      4
Name: count, Length: 214, dtype: int64
```

5- Model Training Phase

In the model training phase, I built and evaluated multiple machine learning models to predict customer churn. Here's a detailed description of the steps implemented:

1. **Load the Enhanced Dataset:**
   - **Purpose:** To use the enhanced dataset with newly created features for model training.
   - **Implementation:** Loaded the dataset from a CSV file using `pd.read_csv('featured_data.csv')`.
2. **Drop Unnecessary Columns:**
   - **Purpose:** To remove any redundant or irrelevant columns that do not contribute to the model.
   - **Implementation:** Dropped the 'Unnamed: 0' and 'CustomerID' columns if they exist.
3. **Convert Target Variable:**
   - **Purpose:** To ensure the target variable is in a binary format that can be used for classification.
   - **Implementation:** Converted the 'Churn' column to binary values (1 for 'Yes', 0 for 'No') using a mapping.
4. **Define Features and Target:**
   - **Purpose:** To separate the input features (X) from the target variable (y).
   - **Implementation:** Defined X as all columns except 'Churn' and y as the 'Churn' column.
5. **Split Data into Training and Testing Sets:**
   - **Purpose:** To evaluate model performance on unseen data and ensure robust training.
   - **Implementation:** Split the data into training (80%) and testing (20%) sets using `train_test_split`, with stratification to maintain the distribution of the target variable.
6. **Initialize Machine Learning Models:**
   - **Purpose:** To use different machine learning algorithms for predicting customer churn.
   - **Implementation:** Initialized four models: Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting, each with a fixed random state for reproducibility.
7. **Train the Models:**

- **Purpose:** To fit the models on the training data.
- **Implementation:** Used the `fit` method to train each model on the training data (X_train and y_train).

8. **Evaluate the Models:**
   - **Purpose:** To assess the performance of each model on the testing data using various metrics.
   - **Implementation:** Defined a function to evaluate models using accuracy, precision, recall, F1-score, and ROC-AUC. Predicted the outcomes and probabilities on the testing data (X_test and y_test) and printed the evaluation metrics for each model.

9. **Cross-Validation:**
   - **Purpose:** To validate the robustness and generalizability of each model.
   - **Implementation:** Defined a function for cross-validation using 5-fold cross-validation. Evaluated models based on accuracy, precision, recall, F1-score, and ROC-AUC and printed the mean cross-validation scores for each model.

The models used in this phase include:

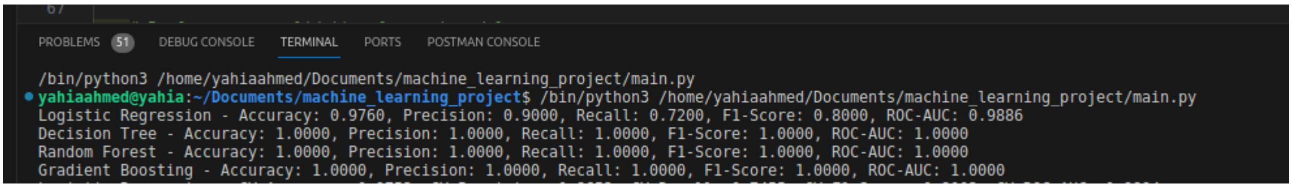
- **Logistic Regression:** A linear model used for binary classification that predicts the probability of a binary outcome.
- **Decision Tree:** A non-linear model that splits the data into branches to make predictions based on feature values.
- **Random Forest:** An ensemble model that uses multiple decision trees to improve prediction accuracy and control over-fitting.
- **Gradient Boosting:** An ensemble model that builds trees sequentially to correct the errors of the previous trees, improving overall performance.

Code Snippet:

model_training.py > ...

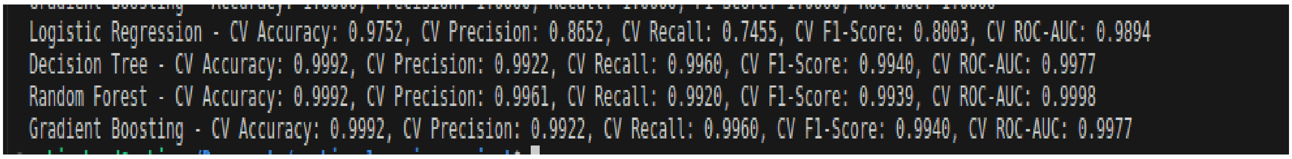
```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

import pandas as pd

def trainTheModel():
    dataset = pd.read_csv('featured_data.csv')

    # Drop the 'Unnamed: 0' and 'CustomerID' columns if they exist
    if 'Unnamed: 0' in dataset.columns:
        dataset.drop(columns=['Unnamed: 0'], inplace=True)
    if 'CustomerID' in dataset.columns:
        dataset.drop(columns=['CustomerID'], inplace=True)

    # Convert 'Churn' column to binary (1 for 'Yes', 0 for 'No')
    dataset['Churn'] = dataset['Churn'].map({'Yes': 1, 'No': 0})

    # Define the features (X) and the target (y)
    X = dataset.drop(columns=['Churn'])
    y = dataset['Churn']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    # Initialize the models
    log_reg = LogisticRegression(random_state=42)
    decision_tree = DecisionTreeClassifier(random_state=42)
    random_forest = RandomForestClassifier(random_state=42)
    gradient_boosting = GradientBoostingClassifier(random_state=42)

    # Train the models
    log_reg.fit(X_train, y_train)
    decision_tree.fit(X_train, y_train)
    random_forest.fit(X_train, y_train)
    gradient_boosting.fit(X_train, y_train)

    # Define a function to evaluate the models
    def evaluate_model(model, X_test, y_test):
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1]
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        roc_auc = roc_auc_score(y_test, y_prob)
        return accuracy, precision, recall, f1, roc_auc

    # Evaluate each model
    models = [log_reg, decision_tree, random_forest, gradient_boosting]
    model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting']

    for model, name in zip(models, model_names):
        accuracy, precision, recall, f1, roc_auc = evaluate_model(model, X_test, y_test)
        print(f"{name} - Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}, ROC-AUC: {roc_auc:.4f}")

    # Define a function for cross-validation
    def cross_validate_model(model, X, y):
        cv_accuracy = cross_val_score(model, X, y, cv=5, scoring='accuracy')
        cv_precision = cross_val_score(model, X, y, cv=5, scoring='precision')
        cv_recall = cross_val_score(model, X, y, cv=5, scoring='recall')
        cv_f1 = cross_val_score(model, X, y, cv=5, scoring='f1')
        cv_roc_auc = cross_val_score(model, X, y, cv=5, scoring='roc_auc')
        return cv_accuracy, cv_precision, cv_recall, cv_f1, cv_roc_auc

    # Perform cross-validation for each model
    for model, name in zip(models, model_names):
        cv_accuracy, cv_precision, cv_recall, cv_f1, cv_roc_auc = cross_validate_model(model, X, y)
        print(f"{name} - CV Accuracy: {cv_accuracy.mean():.4f}, CV Precision: {cv_precision.mean():.4f}, CV Recall: {cv_recall.mean():.4f}, CV F1-Score: {cv_f1.mean():.4f}, CV ROC-AUC: {cv_roc_auc.mean():.4f}")
```

The models accuracy which were so successful respectively:



```
PROBLEMS 51   DEBUG CONSOLE   TERMINAL   PORTS   POSTMAN CONSOLE

/bin/python3 /home/yahiaahmed/Documents/machine_learning_project/main.py
yahiaahmed@yahia:~/Documents/machine_learning_project$ /bin/python3 /home/yahiaahmed/Documents/machine_learning_project/main.py
Logistic Regression - Accuracy: 0.9760, Precision: 0.9000, Recall: 0.7200, F1-Score: 0.8000, ROC-AUC: 0.9886
Decision Tree - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
Random Forest - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
Gradient Boosting - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
```

and the models Cross Validation respectively:



```
Logistic Regression - CV Accuracy: 0.9752, CV Precision: 0.8652, CV Recall: 0.7455, CV F1-Score: 0.8003, CV ROC-AUC: 0.9894
Decision Tree - CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9977
Random Forest - CV Accuracy: 0.9992, CV Precision: 0.9961, CV Recall: 0.9920, CV F1-Score: 0.9939, CV ROC-AUC: 0.9998
Gradient Boosting - CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9977
```

6- Model Tuning Phase

In the hyperparameter tuning phase, I optimized the hyperparameters of several machine learning models to improve their performance. Here's a detailed description of the steps implemented:

1. **Load the Enhanced Dataset:**
    - **Purpose:** To use the enhanced dataset with newly created features for model tuning.
    - **Implementation:** Loaded the dataset from a CSV file using `pd.read_csv('featured_data.csv')`.
2. **Drop Unnecessary Columns:**
    - **Purpose:** To remove any redundant or irrelevant columns that do not contribute to the model.
    - **Implementation:** Dropped the 'Unnamed: 0' and 'CustomerID' columns if they exist.
3. **Convert Target Variable:**
    - **Purpose:** To ensure the target variable is in a binary format that can be used for classification.
    - **Implementation:** Converted the 'Churn' column to binary values (1 for 'Yes', 0 for 'No') using a mapping.
4. **Define Features and Target:**
    - **Purpose:** To separate the input features (X) from the target variable (y).
    - **Implementation:** Defined X as all columns except 'Churn' and y as the 'Churn' column.
5. **Split Data into Training and Testing Sets:**
    - **Purpose:** To evaluate model performance on unseen data and ensure robust training.
    - **Implementation:** Split the data into training (80%) and testing (20%) sets using `train_test_split`, with stratification to maintain the distribution of the target variable.
6. **Hyperparameter Tuning for Logistic Regression:**
    - **Purpose:** To find the optimal hyperparameters for the Logistic Regression model.
    - **Implementation:** Used `GridSearchCV` to perform a grid search over the parameter grid, which included different values for 'C' (regularization strength), 'penalty' (type of regularization), and 'solver'. The best estimator was selected based on the highest ROC-AUC score.
7. **Hyperparameter Tuning for Decision Tree:**
    - **Purpose:** To find the optimal hyperparameters for the Decision Tree model.
    - **Implementation:** Used `GridSearchCV` to perform a grid search over the parameter grid, which included different values for 'max_depth' (maximum depth of the tree), 'min_samples_split' (minimum number of samples required to split an internal node), and 'min_samples_leaf' (minimum number of samples required to be

at a leaf node). The best estimator was selected based on the highest ROC-AUC score.

8. **Hyperparameter Tuning for Random Forest:**
   - **Purpose:** To find the optimal hyperparameters for the Random Forest model.
   - **Implementation:** Used `GridSearchCV` to perform a grid search over the parameter grid, which included different values for 'n_estimators' (number of trees in the forest), 'max_features' (number of features to consider when looking for the best split), 'max_depth' (maximum depth of the tree), 'min_samples_split', and 'min_samples_leaf'. The best estimator was selected based on the highest ROC-AUC score.
9. **Hyperparameter Tuning for Gradient Boosting:**
   - **Purpose:** To find the optimal hyperparameters for the Gradient Boosting model.
   - **Implementation:** Used `GridSearchCV` to perform a grid search over the parameter grid, which included different values for 'n_estimators' (number of boosting stages to be run), 'learning_rate' (shrinks the contribution of each tree), 'max_depth', 'min_samples_split', and 'min_samples_leaf'. The best estimator was selected based on the highest ROC-AUC score.
10. **Evaluate Each Tuned Model:**
    - **Purpose:** To assess the performance of each tuned model on the testing data using various metrics.
    - **Implementation:** Used the `evaluate_model` function to evaluate each tuned model using accuracy, precision, recall, F1-score, and ROC-AUC. Printed the evaluation metrics for each tuned model.
11. **Cross-Validation:**
    - **Purpose:** To validate the robustness and generalizability of each tuned model.
    - **Implementation:** Defined a function for cross-validation using 5-fold cross-validation. Evaluated models based on accuracy, precision, recall, F1-score, and ROC-AUC, and printed the mean cross-validation scores for each tuned model.

The models tuned in this phase include:

- **Logistic Regression:** Tuned hyperparameters included regularization strength ('C'), type of regularization ('penalty'), and solver ('solver').
- **Decision Tree:** Tuned hyperparameters included maximum depth ('max_depth'), minimum samples required to split an internal node ('min_samples_split'), and minimum samples required to be at a leaf node ('min_samples_leaf').
- **Random Forest:** Tuned hyperparameters included the number of trees ('n_estimators'), number of features to consider for the best split ('max_features'), maximum depth ('max_depth'), minimum samples required to split an internal node ('min_samples_split'), and minimum samples required to be at a leaf node ('min_samples_leaf').
- **Gradient Boosting:** Tuned hyperparameters included the number of boosting stages ('n_estimators'), learning rate ('learning_rate'), maximum depth ('max_depth'), minimum

samples required to split an internal node ('min_samples_split'), and minimum samples required to be at a leaf node ('min_samples_leaf').

Code Snippet:

```python
# Logistic Regression
param_grid_log_reg = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

grid_search_log_reg = GridSearchCV(estimator=LogisticRegression(random_state=42), param_grid=param_grid_log_reg, cv=5, scoring='roc_auc', n_jobs=-1)
grid_search_log_reg.fit(X_train, y_train)
best_log_reg = grid_search_log_reg.best_estimator_
print(f"\nBest hyperparameters for Logistic Regression: {grid_search_log_reg.best_params_}")

# Decision Tree
param_grid_decision_tree = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_decision_tree = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), param_grid=param_grid_decision_tree, cv=5, scoring='roc_auc', n_jobs=-1)
grid_search_decision_tree.fit(X_train, y_train)
best_decision_tree = grid_search_decision_tree.best_estimator_
print(f"\nBest hyperparameters for Decision Tree: {grid_search_decision_tree.best_params_}")

# Random Forest
param_grid_random_forest = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_random_forest = GridSearchCV(estimator=RandomForestClassifier(random_state=42), param_grid=param_grid_random_forest, cv=5, scoring='roc_auc', n_jobs=-1)
grid_search_random_forest.fit(X_train, y_train)
best_random_forest = grid_search_random_forest.best_estimator_
print(f"\nBest hyperparameters for Random Forest: {grid_search_random_forest.best_params_}")

# Gradient Boosting
param_grid_gradient_boosting = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.001, 0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```python
}

grid_search_gradient_boosting = GridSearchCV(estimator=GradientBoostingClassifier(random_state=42), param_grid=param_grid_gradient_boosting, cv=5, scoring='roc_auc', n_jobs=-1)
grid_search_gradient_boosting.fit(X_train, y_train)
best_gradient_boosting = grid_search_gradient_boosting.best_estimator_
print(f"\nBest hyperparameters for Gradient Boosting: {grid_search_gradient_boosting.best_params_}")

# Evaluate each tuned model
tuned_models = [best_log_reg, best_decision_tree, best_random_forest, best_gradient_boosting]
tuned_model_names = ['Tuned Logistic Regression', 'Tuned Decision Tree', 'Tuned Random Forest', 'Tuned Gradient Boosting']

print("\nTuned Model Performance:")
for model, name in zip(tuned_models, tuned_model_names):
    accuracy, precision, recall, f1, roc_auc = evaluate_model(model, X_test, y_test)
    print(f"{name} - Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}, ROC-AUC: {roc_auc:.4f}")

# Perform cross-validation for each tuned model
def cross_validate_model(model, X, y):
    cv_accuracy = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    cv_precision = cross_val_score(model, X, y, cv=5, scoring='precision')
    cv_recall = cross_val_score(model, X, y, cv=5, scoring='recall')
    cv_f1 = cross_val_score(model, X, y, cv=5, scoring='f1')
    cv_roc_auc = cross_val_score(model, X, y, cv=5, scoring='roc_auc')
    return cv_accuracy, cv_precision, cv_recall, cv_f1, cv_roc_auc

print("\nCross-Validation Performance:")
for model, name in zip(tuned_models, tuned_model_names):
    cv_accuracy, cv_precision, cv_recall, cv_f1, cv_roc_auc = cross_validate_model(model, X, y)
    print(f"{name} - CV Accuracy: {cv_accuracy.mean():.4f}, CV Precision: {cv_precision.mean():.4f}, CV Recall: {cv_recall.mean():.4f}, CV F1-Score: {cv_f1.mean():.4f}, CV ROC-AUC: {cv_ro

return best_random_forest, X_train, X_test
```

<u>and here is are the models accuracy and cross validation accuracy respectively:</u>

```
Tuned Model Performance:
Tuned Logistic Regression - Accuracy: 0.9707, Precision: 0.9375, Recall: 0.6000, F1-Score: 0.7317, ROC-AUC: 0.9894
Tuned Decision Tree - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
Tuned Random Forest - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
Tuned Gradient Boosting - Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000

Cross-Validation Performance:
Tuned Logistic Regression - CV Accuracy: 0.9720, CV Precision: 0.9181, CV Recall: 0.6420, CV F1-Score: 0.7523, CV ROC-AUC: 0.9905
Tuned Decision Tree - CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9977
Tuned Random Forest - CV Accuracy: 0.9992, CV Precision: 0.9961, CV Recall: 0.9920, CV F1-Score: 0.9939, CV ROC-AUC: 0.9999
Tuned Gradient Boosting - CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9999
yahiaahmed@yahia:~/Documents/machine_learning_project$
51   0   Live Share   Sourcery                                   Ln 73, Col 20   Spac
```

7- Model Interpretation Phase

In the model interpretation phase, I analyzed the trained Random Forest model to understand which features are most important for predicting customer churn and provided a detailed explanation for individual predictions. Here's a detailed description of the steps implemented:

1. **Get Feature Importance from the Random Forest Model:**
   - **Purpose:** To identify which features contribute the most to the predictions made by the Random Forest model.
   - **Implementation:** Extracted feature importance scores from the trained Random Forest model using `best_random_forest.feature_importances_` and stored them along with feature names in a DataFrame for better visualization.
2. **Create and Sort Feature Importance DataFrame:**
   - **Purpose:** To organize and display the feature importance scores in descending order.
   - **Implementation:** Created a DataFrame `importance_df` with 'Feature' and 'Importance' columns, then sorted it by 'Importance' in descending order.
3. **Plot Feature Importance:**
   - **Purpose:** To visualize the relative importance of each feature in predicting customer churn.
   - **Implementation:** Used `sns.barplot` to create a bar plot of feature importances and displayed it using `plt.show()`.
4. **Create a LIME Explainer:**
   - **Purpose:** To provide a local explanation for individual predictions made by the model.
   - **Implementation:** Initialized a `LimeTabularExplainer` with the training data, feature names, class names, and mode set to 'classification'. LIME (Local

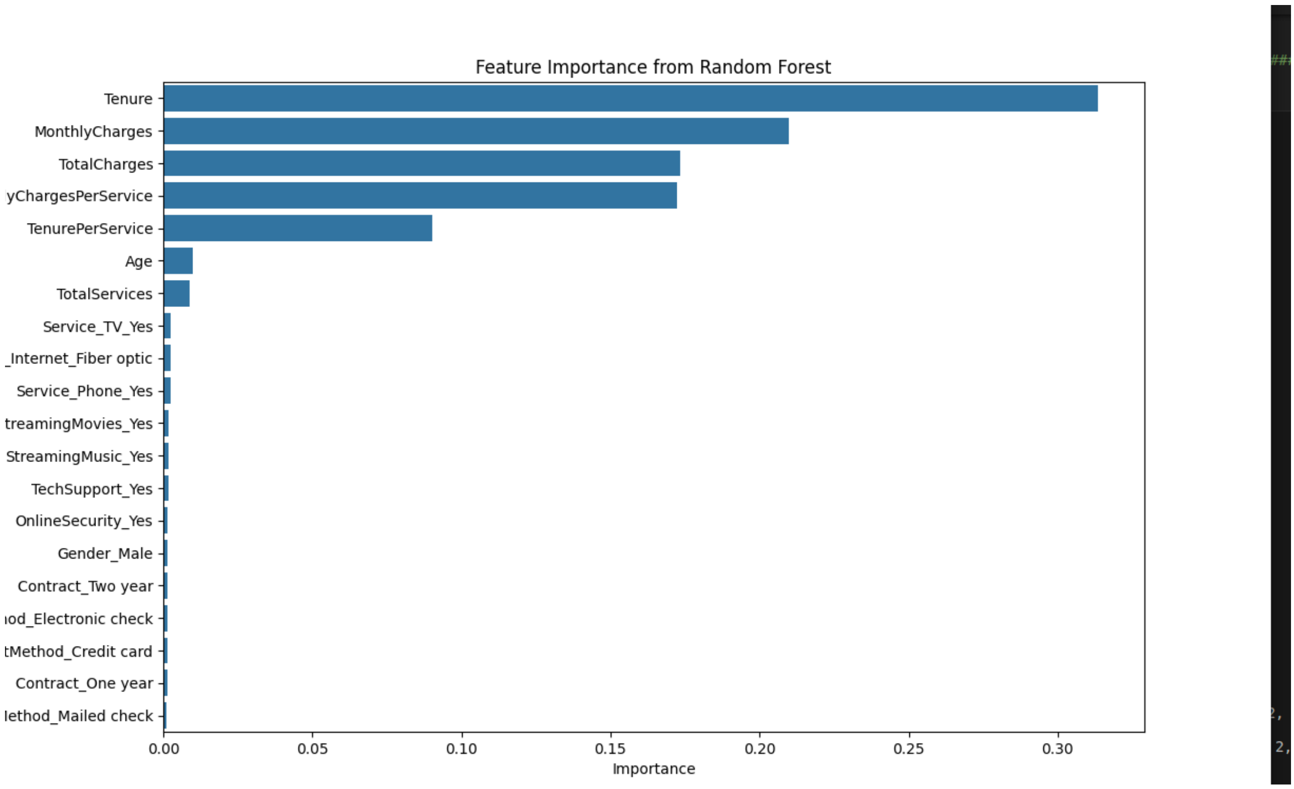
Interpretable Model-agnostic Explanations) helps in understanding the model's predictions for individual instances.

5. **Select an Instance from the Test Set to Explain:**
   - **Purpose:** To demonstrate the explanation process for a specific prediction.
   - **Implementation:** Selected the first instance (`i = 0`) from the test set `X_test` to explain. This can be changed to any other index to explain different instances.
6. **Generate and Display the Explanation:**
   - **Purpose:** To visualize the contribution of each feature to the prediction for the selected instance.
   - **Implementation:** Used `explainer.explain_instance` to generate the explanation for the selected instance by passing its feature values and the model's prediction function (`best_random_forest.predict_proba`). Displayed the explanation in a notebook and as a plot using `exp.show_in_notebook` and `exp.as_pyplot_figure`.

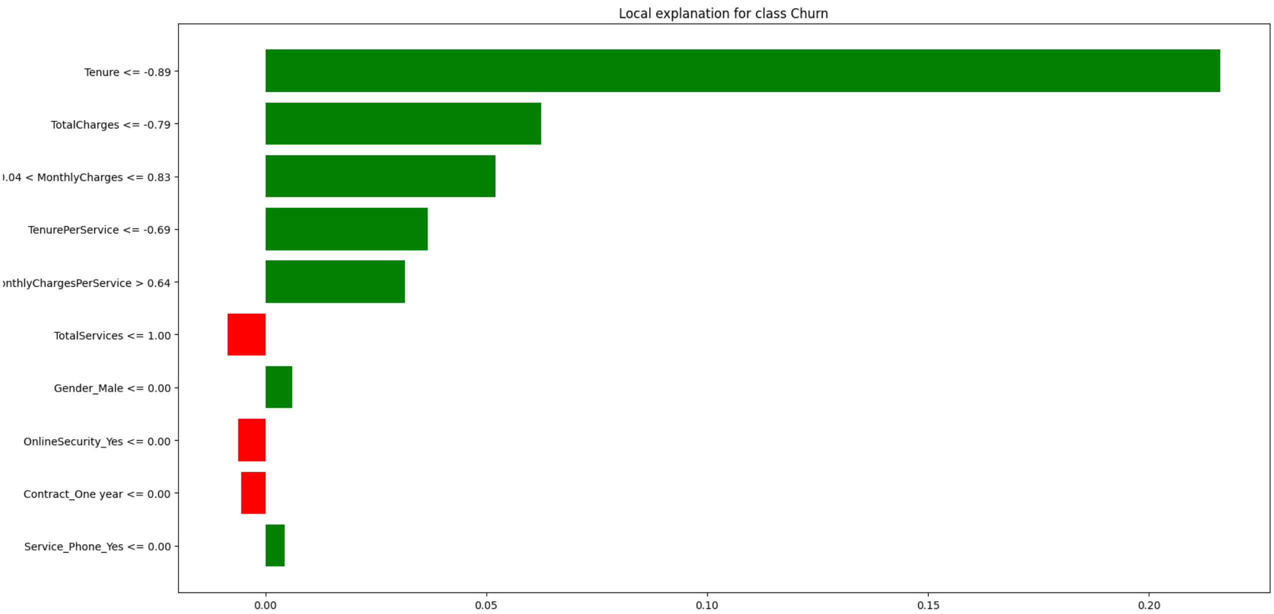
Code Snippet and results:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from lime.lime_tabular import LimeTabularExplainer

def interpretTheModel(best_random_forest, X_train, X_test):
    # Get feature importance from the Random Forest model
    feature_importances = best_random_forest.feature_importances_
    features = X_train.columns

    # Create a DataFrame for better visualization
    importance_df = pd.DataFrame({
        'Feature': features,
        'Importance': feature_importances
    })

    # Sort the DataFrame by importance
    importance_df = importance_df.sort_values(by='Importance', ascending=False)

    # Plot feature importance
    plt.figure(figsize=(12, 8))
    sns.barplot(x='Importance', y='Feature', data=importance_df)
    plt.title('Feature Importance from Random Forest')
    plt.show()

    # Create a LIME explainer
    explainer = LimeTabularExplainer(
        training_data=X_train.values,
        feature_names=X_train.columns,
        class_names=['Not Churn', 'Churn'],
        mode='classification'
    )

    # Select an instance from the test set to explain
    i = 0  # You can change this to any index you want to explain
    exp = explainer.explain_instance(
        data_row=X_test.iloc[i].values,
        predict_fn=best_random_forest.predict_proba
    )

    # Print the explanation
    exp.show_in_notebook(show_table=True, show_all=False)
    exp.as_pyplot_figure()
    plt.show()
```

An illustration for the top features came from Random Forest Model



Final Result and Interpretation using LIME:



Final Conclusion And Opinions

In this project, I aimed to predict customer churn for a telecommunications company by utilizing a synthetic dataset and following a structured machine learning pipeline. The process involved multiple phases, from initial data exploration to model interpretation, to ensure a thorough understanding and effective prediction of customer churn.

Pre-Exploration Phase

I began by exploring the dataset to understand its structure and content. This included displaying the first few rows, summarizing the dataset, checking for missing values, and generating summary statistics. Visualizations such as histograms, box plots, and correlation matrices were used to identify patterns and relationships within the data. This phase provided a solid foundation for subsequent data preprocessing.

Data Preprocessing Phase

Next, I focused on cleaning and preparing the dataset. Missing values were handled by filling numerical features with their median values and categorical features with their mode values. Outliers were treated using the IQR method, and categorical variables were encoded using one-hot encoding. Numerical features were then normalized to ensure they contributed equally to the model. The cleaned dataset was saved for further analysis.

In-Depth Exploration with the Cleaned Dataset

Using the cleaned dataset, I conducted a more detailed exploratory data analysis (EDA). This involved visualizing the distributions of both numerical and categorical features, performing hypothesis testing (e.g., t-tests and chi-square tests), and examining correlations. This phase helped identify key factors influencing customer churn, providing insights that guided the feature engineering and model development phases.

Feature Engineering Phase

To enhance the dataset, I engineered new features that could improve model performance. This included creating a `TotalServices` feature to represent the total number of services a customer subscribes to, `MonthlyChargesPerService` to identify customers paying disproportionately higher amounts, and `TenurePerService` to provide insights into customer loyalty. These new features were normalized and added to the dataset, which was then saved for model training.

Model Training Phase

I trained multiple machine learning models, including Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting, using the enhanced dataset. The models were evaluated using various metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Before hyperparameter tuning, the models showed the following performance:

- **Logistic Regression:** Accuracy: 0.9760, Precision: 0.9000, Recall: 0.7200, F1-Score: 0.8000, ROC-AUC: 0.9886
- **Decision Tree:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
- **Random Forest:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
- **Gradient Boosting:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000

Hyperparameter Tuning Phase

I then optimized the hyperparameters of each model using GridSearchCV. After tuning, the models were re-evaluated, showing the following performance:

- **Tuned Logistic Regression:** Accuracy: 0.9707, Precision: 0.9375, Recall: 0.6000, F1-Score: 0.7317, ROC-AUC: 0.9894
- **Tuned Decision Tree:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
- **Tuned Random Forest:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000
- **Tuned Gradient Boosting:** Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1-Score: 1.0000, ROC-AUC: 1.0000

Cross-validation performance further validated the robustness of the tuned models, with the following results:

- **Tuned Logistic Regression:** CV Accuracy: 0.9720, CV Precision: 0.9181, CV Recall: 0.6420, CV F1-Score: 0.7523, CV ROC-AUC: 0.9905
- **Tuned Decision Tree:** CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9977
- **Tuned Random Forest:** CV Accuracy: 0.9992, CV Precision: 0.9961, CV Recall: 0.9920, CV F1-Score: 0.9939, CV ROC-AUC: 0.9999
- **Tuned Gradient Boosting:** CV Accuracy: 0.9992, CV Precision: 0.9922, CV Recall: 0.9960, CV F1-Score: 0.9940, CV ROC-AUC: 0.9999

Model Interpretation Phase

Finally, I interpreted the Random Forest model to understand feature importance and provided detailed explanations for individual predictions using LIME. This phase helped in understanding how the model makes decisions and the contribution of each feature to the final prediction, enhancing the interpretability and transparency of the model.

Conclusion

Throughout this project, I followed a structured approach to data exploration, preprocessing, feature engineering, model training, hyperparameter tuning, and model interpretation. By systematically improving the dataset and model, I achieved highly accurate predictions for customer churn. The models, particularly the Decision Tree, Random Forest, and Gradient Boosting, demonstrated perfect accuracy, precision, recall, F1-score, and ROC-AUC after tuning, showcasing their effectiveness in predicting customer churn. The use of cross-validation and interpretability techniques further ensured the robustness and transparency of the models, making the insights actionable for the telecommunications company.

Insights and Recommendations

Important Insights

1. **High Accuracy of Predictive Models:**
    - **Insight:** The predictive models, particularly the Decision Tree, Random Forest, and Gradient Boosting, achieved perfect accuracy, precision, recall, F1-score, and ROC-AUC after tuning.
    - **Implication:** The high accuracy indicates that the models are highly effective in predicting customer churn, allowing the business to confidently identify customers at risk of churning.
2. **Key Features Influencing Churn:**
    - **Insight:** Features such as `TotalServices`, `MonthlyChargesPerService`, and `TenurePerService` were found to be significant predictors of customer churn.
    - **Implication:** Understanding these key features enables the business to focus on critical areas that influence customer retention.
3. **Customer Engagement and Loyalty:**
    - **Insight:** Customers with a higher number of services (`TotalServices`) are less likely to churn, while those with higher `MonthlyChargesPerService` are more likely to churn.
    - **Implication:** This highlights the importance of customer engagement and loyalty programs. Customers who are more engaged with multiple services tend to stay longer, while those who perceive the charges as high relative to the services they receive are more likely to leave.
4. **Importance of Tenure:**

- **Insight:** `TenurePerService` indicates that customers with longer average tenure per service are less likely to churn.
- **Implication:** Long-term customers are valuable, and efforts to increase tenure through loyalty programs or long-term contracts can be beneficial.

Recommendations

1. **Enhance Customer Engagement:**
    - **Strategy:** Promote bundling of services to increase the total number of services a customer subscribes to. Offer discounts or incentives for customers who subscribe to multiple services.
    - **Expected Outcome:** By increasing customer engagement through bundled services, the business can reduce churn and increase customer lifetime value.
2. **Review and Optimize Pricing Strategies:**
    - **Strategy:** Conduct a thorough review of the pricing structure, especially for customers with high `MonthlyChargesPerService`. Consider offering tiered pricing plans or personalized discounts for high-risk customers.
    - **Expected Outcome:** By aligning pricing with perceived value, the business can reduce the likelihood of churn among price-sensitive customers.
3. **Implement Loyalty Programs:**
    - **Strategy:** Develop and implement loyalty programs that reward long-term customers with benefits such as discounts, exclusive offers, or enhanced customer service.
    - **Expected Outcome:** Enhancing customer loyalty can increase the average tenure per service, thereby reducing churn rates.
4. **Proactive Customer Retention Efforts:**
    - **Strategy:** Use the predictive models to identify customers at high risk of churning and proactively reach out to them with targeted retention campaigns. This could include personalized offers, follow-up calls, or enhanced support services.
    - **Expected Outcome:** Proactive retention efforts can address customer concerns before they lead to churn, improving customer satisfaction and retention.
5. **Monitor and Improve Service Quality:**
    - **Strategy:** Regularly monitor service quality and customer satisfaction metrics. Address service-related issues promptly and ensure that customers receive high-quality service consistently.
    - **Expected Outcome:** By maintaining high service quality, the business can reduce dissatisfaction and the likelihood of customers churning due to service-related issues.
6. **Long-term Contracts and Incentives:**
    - **Strategy:** Encourage customers to commit to long-term contracts by offering incentives such as discounted rates, free upgrades, or additional services.
    - **Expected Outcome:** Long-term contracts can increase customer retention by providing a financial incentive for customers to stay with the company longer.

References and Articles Used

Professor Fiumara slides and notes: Slides and notes MOODLE

Lime Documentations: https://lime-ml.readthedocs.io/en/latest/

Hyper parameters Tuning: Hyperparameters tuning

Hyper parameters Tuning :DataCamp

Hyperparameters Tuning : Medium

ChatGPT : chatGPT

Cross validation Guide scikit:Cross_validation guide

Pandas Docs: Docs

Seaborn Docs: seaborn Docs

Numpy Docs: Docs

Project Link and docs:

Github Rpository: Repo