Customer Churn Prediction

Milestone: Project Report

Group 03
Pramoth Guhan
Harith Anand

(857) 891-6677 (Tel of Student 1)
(857) 423-5587 (Tel of Student 2)

guhan.p@northeastern.edu
anand.ha@northeastern.edu

Percentage of Effort Contributed by Student 1: 50

Percentage of Effort Contributed by Student 2: 50

Signature of Student 1: Pramoth Guhan

Signature of Student 2: Haritha Anand

Submission Date: 04/23/2024

Problem Setting:

In the highly competitive telecommunications industry, customer churn is a significant issue that directly affects revenue and long-term viability. Telecom companies must navigate a saturated market where customers have multiple service providers to choose from, making customer loyalty both more valuable and more challenging to maintain. Understanding the reasons behind customer departures is crucial for developing effective retention strategies. This project seeks to dissect customer data to unveil the primary indicators that can predict churn. Among the challenges are the intricate nature of datasets that combine customer demographics, usage patterns, service satisfaction, and billing details; the requirement for a high degree of model accuracy to ensure trust in predictions; and the extraction of actionable business strategies from the data analytics process.

Problem Statement:

The focal point of this project is to harness data analytics for predicting customer churn in the telecommunications sector. Through comprehensive analysis, the study is designed to explore the underlying causes of customer attrition and to use predictive modeling for identifying at-risk customers. Key questions that guide this research include:

1. What are the key factors that contribute to customer churn? We aim to understand which variables, such as service usage patterns, billing issues, or customer service interactions, are the most significant predictors of churn.
2. Can we predict which customers are likely to churn soon? The objective is to develop a predictive model that can accurately identify customers who are at high risk of churning, allowing for timely intervention strategies.
3. What strategies can be implemented to reduce churn based on the insights gained from the data? This involves translating the data-driven insights into practical actions. Strategies could range from personalized customer engagement initiatives, adjustments to pricing models, improvements in customer service, to offering tailored packages and promotions.

By addressing these areas, we intend to provide the telecommunications company with a model that not only forecasts potential churn but also encapsulates a deeper comprehension of customer behavior and preferences. The end goal is to pivot these insights into strategic decisions that enhance customer satisfaction and loyalty, thereby reducing churn and fostering sustainable growth.

Data Source:

The primary data source for this project is the "Telco Customer Churn" dataset available on Kaggle [1]. This dataset includes various customer attributes and their churn status. The dataset was originally provided by IBM as part of their sample data sets to help predict customer churn in the telecommunications industry.

Data Description:

The "Telco Customer Churn" dataset comprises several columns representing customer attributes and their churn status. Key attributes and Features include:

1. customerID: A unique identifier for the customer.
2. gender: The customer's gender (e.g., 'Female').
3. SeniorCitizen: Indicates if the customer is a senior citizen (1) or not (0).
4. Partner: Indicates if the customer has a partner (‘Yes’) or not (‘No’).
5. Dependents: Indicates if the customer has dependents (‘Yes’) or not (‘No’).
6. tenure: The number of months the customer has stayed with the company.
7. PhoneService: Indicates if the customer has phone service (‘Yes’) or not (‘No’).
8. MultipleLines: Indicates if the customer has multiple lines (‘Yes’), no phone service (‘No phone service’), or only one line (‘No’).
9. InternetService: Customer’s internet service type (‘DSL’, ‘Fiber optic’, or ‘No’).
10. OnlineSecurity: Indicates if the customer subscribes to an additional online security service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).
11. OnlineBackup: Indicates if the customer subscribes to an online backup service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).
12. DeviceProtection: Indicates if the customer has device protection service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).
13. TechSupport: Indicates if the customer has tech support service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).
14. StreamingTV: Indicates if the customer has streaming TV service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).
15. StreamingMovies: Indicates if the customer has streaming movies service (‘Yes’), does not (‘No’), or if they have no internet service (‘No internet service’).

16. Contract: The term of the customer's contract (‘Month-to-month’, ‘One year’, ‘Two year’).
17. PaperlessBilling: Indicates if the customer has paperless billing (‘Yes’) or not (‘No’).
18. PaymentMethod: The customer’s payment method (e.g., ‘Electronic check’).
19. MonthlyCharges: The amount charged to the customer monthly.
20. TotalCharges: The total amount charged to the customer.

Target Variables: Churn: Indicates if the customer churned (‘Yes’) or not (‘No’).

The dataset contains 7,043 customer records, and the distribution of the target variable Churn where it has - No: 5,174 customers (approximately 73.5%); Yes: 1,869 customers (approximately 26.5%). This indicates that the dataset is somewhat imbalanced, as the number of customers who have not churned significantly exceeds those who have. In practical terms, this imbalance can lead model training processes to favor the majority class (customers who do not churn), potentially leading to poorer model performance on the minority class (customers who churn).

Data Exploration:

Summary Statistics of Features

1. Find Categorical Columns and Summary Statistics

The statistical summary displays metrics for categorical features, specifically 'Partner' column has 'No' as the top category with a frequency of 3641, and the 'Dependents' column also has 'No' as the top category with a frequency of 4933. Other features like 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', and 'PaymentMethod' are summarized, each displaying the count, number of unique values, and the most frequent category along with its count.

```
In [7]: # Find categorical columns
        categorical_columns = df.select_dtypes(include=['object']).columns.tolist()
        categorical_columns

Out[7]: ['customerID',
         'gender',
         'Partner',
         'Dependents',
         'PhoneService',
         'MultipleLines',
         'InternetService',
         'OnlineSecurity',
         'OnlineBackup',
         'DeviceProtection',
         'TechSupport',
         'StreamingTV',
         'StreamingMovies',
         'Contract',
         'PaperlessBilling',
         'PaymentMethod',
         'TotalCharges',
         'Churn']

In [8]: # Summary statistics for categorical features
        categorical_features = df.select_dtypes(include=[object])
        categorical_features.describe()
```

Out[8]:

	customerID	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
count	7043	7043	7043	7043	7043	7043	7043	7043	7043	7043	7043
unique	7043	2	2	2	2	3	3	3	3	3	3
top	7590-VHVEG	Male	No	No	Yes	No	Fiber optic	No	No	No	No
freq	1	3555	3641	4933	6361	3390	3096	3498	3088	3095	3473

2. Find Numerical Columns and Summary Statistics

The statistical summary displays metrics for numerical features, specifically 'SeniorCitizen', 'tenure', and 'MonthlyCharges' from a dataset. The count indicates the number of non-null records for each feature, while the mean value represents the average. The standard deviation describes the variability of the dataset. The minimum and maximum values are shown, along with the 25th percentile (lower quartile), the 50th percentile (median), and the 75th percentile (upper quartile) for each numerical feature. For instance, the 'SeniorCitizen' feature has a mean value close to 0, suggesting a lower proportion of senior citizens in the dataset. The 'tenure' feature displays a range of values, with the 50th percentile indicating that the median tenure is around 29, which could imply a mix of new and long-term customers. 'MonthlyCharges' also varies significantly, with the mean suggesting the average monthly billing amount, and the standard deviation indicating the spread of charges across customers.

```
In [5]: # Find numerical columns
        numerical_columns = df.select_dtypes(include=[np.number]).columns.tolist()
        numerical_columns

Out[5]: ['SeniorCitizen', 'tenure', 'MonthlyCharges']

In [6]: # Summary statistics for numerical features
        numerical_features =  df.select_dtypes(include=[np.number])
        numerical_features.describe()
```

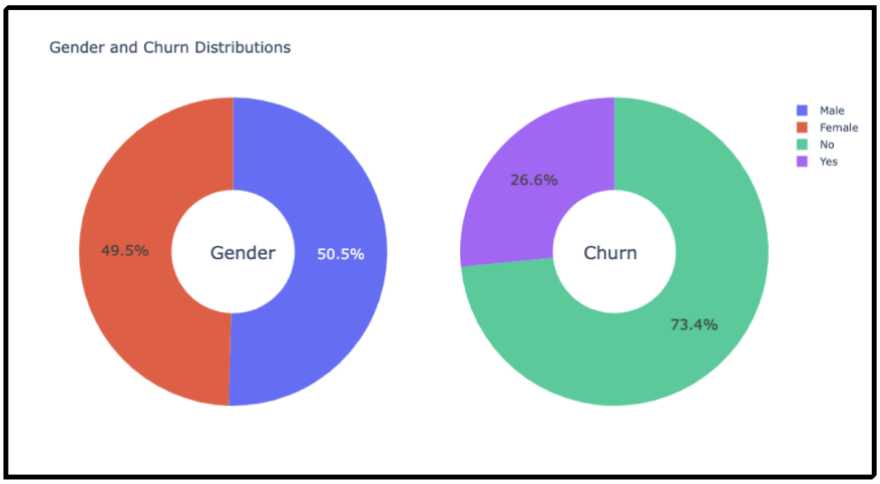
Out[6]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Exploratory Data Analysis:

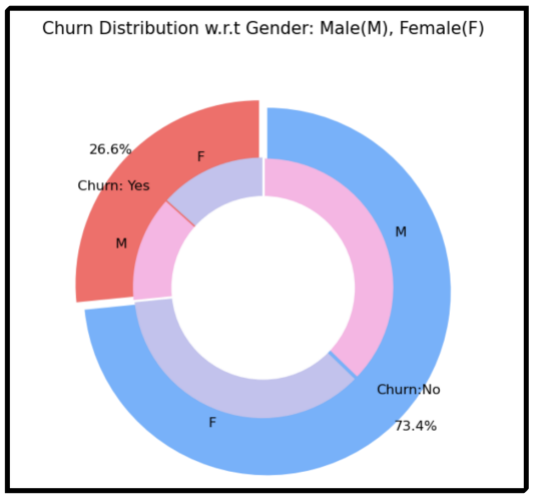
1. Gender and Churn Distribution

The gender distribution in [1] is nearly balanced, which suggests that any analysis or modeling on gender-related trends can proceed without concern for major imbalance in representation. For churn, the distribution indicates that a significant majority of customers are retained (73.4%), while about a quarter of the customers (26.6%) have churned. This insight is critical for businesses focusing on customer retention strategies, as it provides a quantitative baseline for evaluating churn rates and the effectiveness of retention programs.



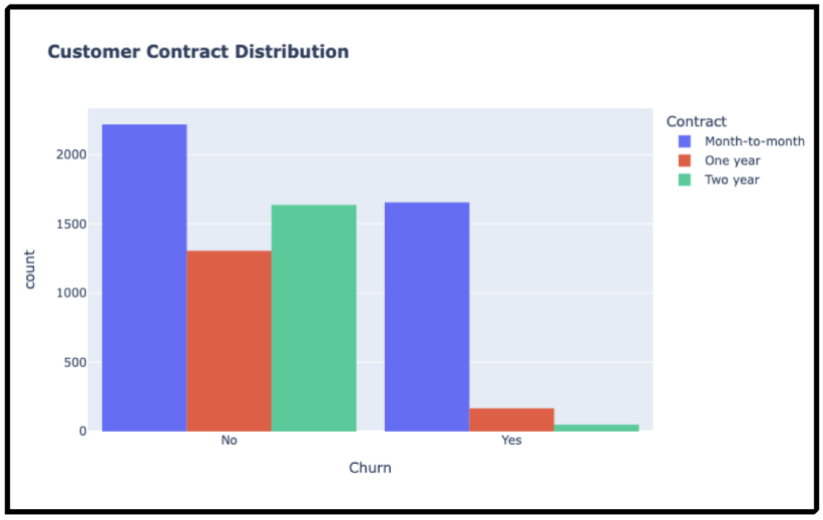
2. Customer Churn Ratio

The donut chart depicts the customer churn ratio with respect to gender. Approximately 26.6% of customers have churned, with an even distribution between males and females. The majority, 73.4%, have not churned, suggesting that gender is not a primary factor in churn for this dataset.



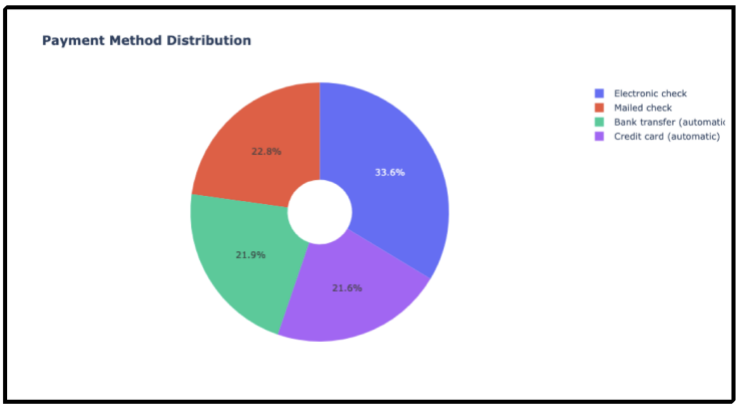
3. Customer Contract Distribution

The bar chart illustrates the distribution of customer contracts in relation to churn. Customers with month-to-month contracts are more likely to churn compared to those with one or two-year contracts. Most customers who have not churned are on month-to-month contracts, suggesting that shorter-term commitments may be a factor in churn decisions.



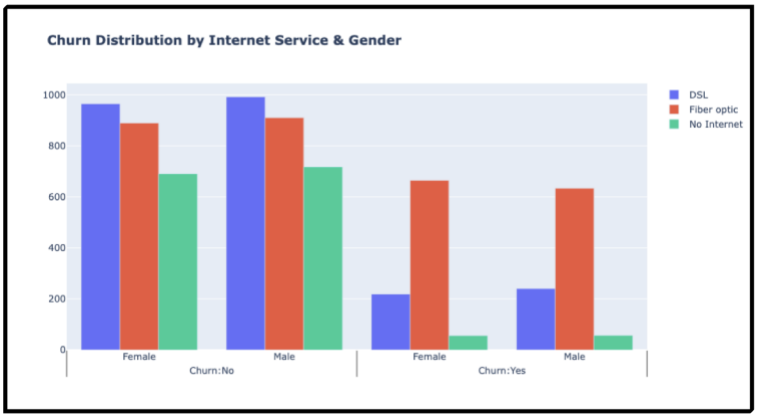
4. Payment Method Distribution

The donut chart shows the distribution of payment methods among customers. The largest segment uses electronic checks at 33.6%, followed by mailed checks and bank transfers, each accounting for approximately 22%. Credit card payments are used by 21.6% of customers. This distribution highlights the prevalence of electronic payment methods and may suggest areas for improvement in payment options or customer convenience.



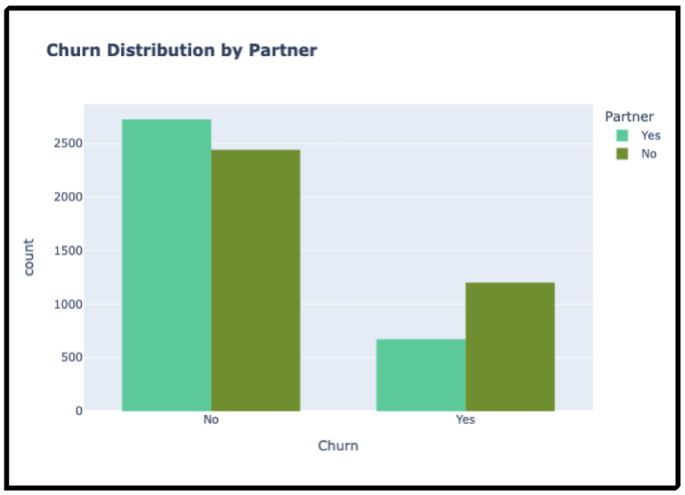
5. Internet and Services and Churn Status by Gender

The bar chart illustrates churn distribution by internet service type and gender. For both males and females, the highest churn occurs in the fiber optic service category. DSL service users exhibit lower churn rates, while customers with no internet service have the lowest churn rates among the services provided. The pattern of churn relative to internet service type appears consistent across genders.



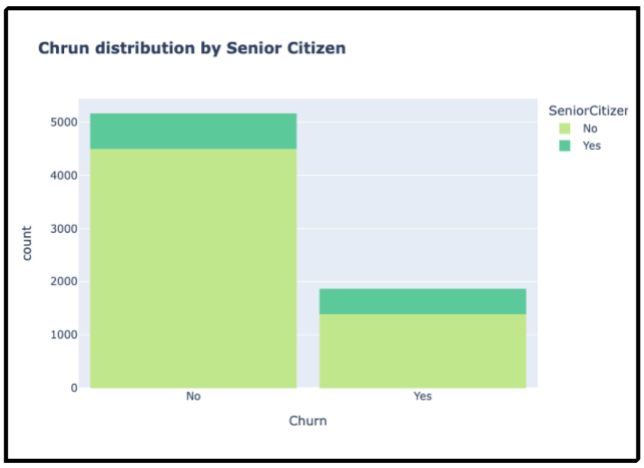
6. Churn Distribution by Partner

The bar chart demonstrates the churn distribution relative to the presence of a partner. Customers without a partner show higher churn compared to those with a partner, evident in both the "No" and "Yes" churn categories. This trend suggests that having a partner may be associated with lower churn rates.



7. Churn Distribution by Senior Citizen

The bar chart displays churn distribution among senior citizens. Non-senior citizens have a lower churn rate, while the churn rate for senior citizens is noticeably higher. This indicates that age might be a factor in the likelihood of churn within this customer base.

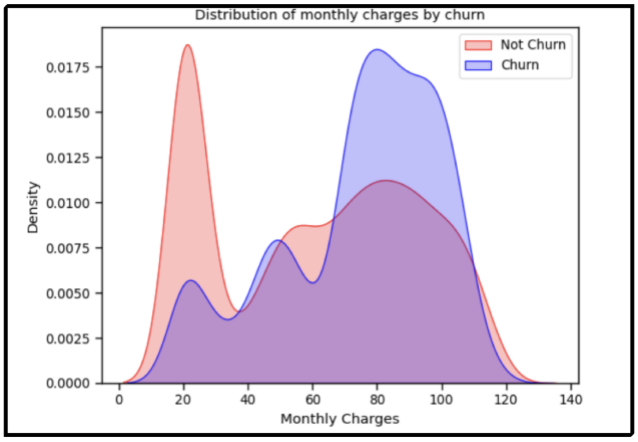


8. Churn Distribution by Online Security

The bar chart showcases the churn distribution in relation to customers' online security service subscription. Customers without online security services exhibit a higher churn rate compared to those with the service. Additionally, customers without any internet service have the lowest churn rates. This suggests that online security services could be a factor influencing customer retention.

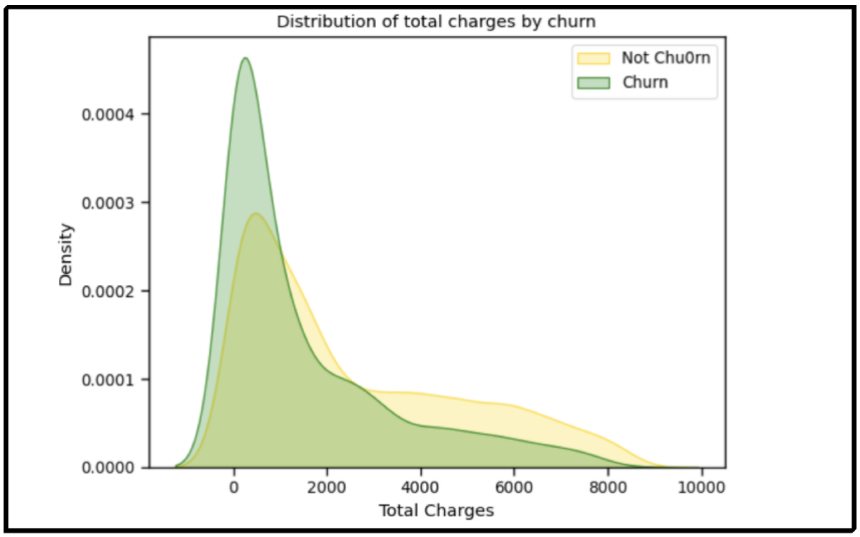
9. Distribution of Monthly Charges by Churn

The density plot compares the distribution of monthly charges between customers who have churned and those who have not. There are distinct peaks for both groups, with the churn group showing a higher density at higher monthly charges. This could imply that higher monthly charges may correlate with increased customer churn.



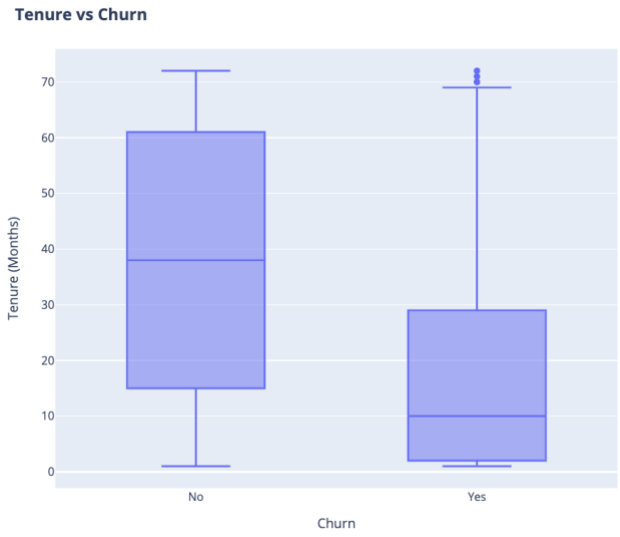
10. Distribution of Total Charges by Churn

The density plot illustrates the distribution of total charges by churn status. The plot for customers who have not churned (labeled 'Not Churn') peaks sharply for lower total charges and then tapers off. In contrast, the churned customers' distribution (labeled 'Churn') is flatter with a less pronounced peak, suggesting that customers with lower total charges are less likely to churn, while those with higher total charges show a more uniform likelihood of churning. This pattern might indicate that customers who have spent less over time are more likely to stay, whereas the likelihood of churning does not decrease as markedly with increased total charges.



11. Tenure vs Churn Boxplot

The boxplot compares tenure lengths between customers who have churned and those who have not. Customers who have not churned ('No') generally have a longer tenure, as shown by a higher median (closer to 50 months) and a wider interquartile range, indicating a more extended period with the service. In contrast, those who have churned ('Yes') have a shorter median tenure (noticeably below 30 months) and a more compact interquartile range, suggesting a shorter duration with the service. Outliers for both categories indicate some customers who have churned with high tenure and vice versa, but these are exceptions rather than the norm.

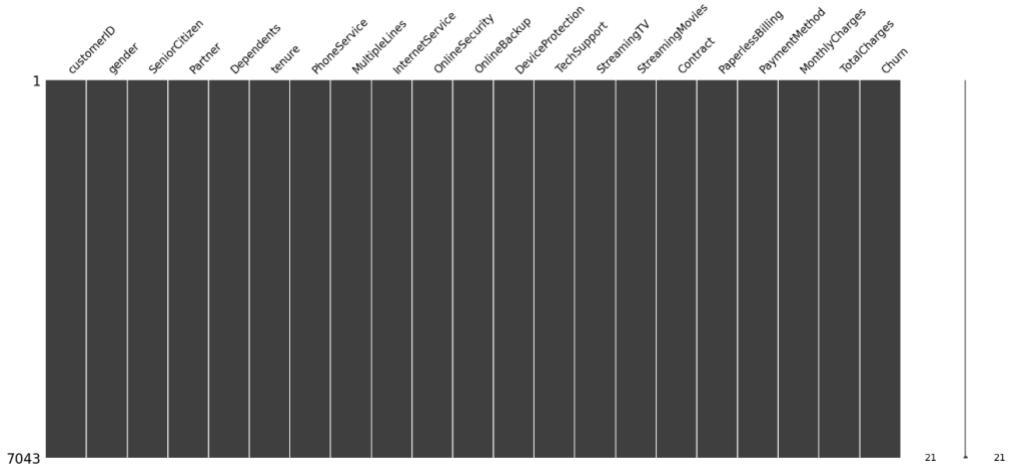


Data Mining Tasks:

Data Cleaning:

1. Visualize missing values as a Matrix.

In this matrix, the x-axis would represent the dataset's variables (e.g., customerID, gender, SeniorCitizen), while the y-axis represents individual observations (often anonymized as indices). The darkened areas would indicate non-missing values, while lighter shades or distinct colors (not visible in the image) would denote missing values. Since the entire matrix is dark, this suggests that there are no missing values in the dataset for any of the listed features, which indicates a complete dataset with no immediate need for imputation of missing data.



2. Dropping Unnecessary Columns.

```
# Drop Unnecessary columns
df = df.drop(['customerID'], axis=1)
df.head(3)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupp
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	

We have dropped the Customer ID as we have not used it for our data analysis.

3. Check Missing Value

```
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
df.isnull().sum()
```

```
gender             0
SeniorCitizen      0
Partner            0
Dependents         0
tenure             0
PhoneService       0
MultipleLines      0
InternetService    0
OnlineSecurity     0
OnlineBackup       0
DeviceProtection   0
TechSupport        0
StreamingTV        0
StreamingMovies    0
Contract           0
PaperlessBilling   0
PaymentMethod      0
MonthlyCharges     0
TotalCharges      11
Churn              0
dtype: int64
```

After checking the missing values, it is seen that there are some blank spaces in TotalCharges. So, we have dropped those rows.

```
df[df['tenure']==0].index

df.drop(labels=df[df['tenure']==0].index, axis=0, inplace=True) # row delelation
df[df['tenure']==0].index # now no missing values
```

```
Index([], dtype='int64')
```

- No other missing values were found in tenure except the above
- Deleting the above missing values in tenuure with charges

```
df['TotalCharges'].isnull().sum()
```

```
0
```

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

```
gender             0
SeniorCitizen      0
Partner            0
Dependents         0
tenure             0
PhoneService       0
MultipleLines      0
InternetService    0
OnlineSecurity     0
OnlineBackup       0
DeviceProtection   0
TechSupport        0
StreamingTV        0
StreamingMovies    0
Contract           0
PaperlessBilling   0
PaymentMethod      0
MonthlyCharges     0
TotalCharges       0
Churn              0
dtype: int64
```

It is found that the feature ‘tenure’ has some 0 values which means that the customers have not been an active customer for a long time. Therefore, we identified and dropped the rows.

Data Preprocessing:

1. One-Hot Encoding the Categorical Variables:

One-hot encoding is a crucial step in data preprocessing, particularly when preparing categorical data for machine learning models. It involves transforming each categorical class within a variable into a binary column. This transformation is applied to ensure that algorithms process the data without assuming an erroneous order among categories. For example, gender classes like 'Male' and 'Female' become separate columns, each indicating the presence of a class with a binary flag. This process is vital for allowing models to utilize categorical data effectively and is particularly important for models that require numerical input, ensuring an accurate and unbiased interpretation of the data.

Data Preprocessing

1. One-Hot Encoding the categorical variables

```
# One-hot encoding the categorical variables


df = pd.get_dummies(df, columns=[
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
    'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
    'PaperlessBilling', 'PaymentMethod'
])
```

2. Skewness of the Numerical Variables:

The skewness of numerical variables is essential for statistical models that assume data is normally distributed. The skewness of a feature measures its asymmetry and the extent to which it deviates from a normal distribution. The original skewness for 'TotalCharges' was approximately 0.961562, indicating moderate skewness. However, after applying a Box-Cox transformation, a common technique for stabilizing variance and making the data more normal-like, the skewness was significantly reduced to -2.161548991653334, reflecting an overcorrection into high skewness in the opposite direction.

2. Skewness of the Numerical Variables

```
#Skewness

cols = ["MonthlyCharges","TotalCharges","tenure"]

df[cols].skew().to_frame().rename(columns={0:"Feature Skewness"})
```

	Feature Skewness
MonthlyCharges	-0.222103
TotalCharges	0.961642
tenure	0.237731

Here are some guidelines for interpreting the skewness value:

- If the skewness is between -0.5 and 0.5, the data are fairly symmetrical.
- If the skewness is between -1 and -0.5 or between 0.5 and 1, the data are moderately skewed.
- If the skewness is less than -1 or greater than 1, the data are highly skewed.

```
df['TotalCharges'] = df['TotalCharges'] + 1  # To handle zero or negative values

# Apply the Box-Cox transformation
df['TotalCharges_BoxCox'], fitted_lambda = stats.boxcox(df['TotalCharges'])

# Check the skewness after Box-Cox transformation
new_skewness = df['TotalCharges_BoxCox'].skew()
new_skewness
```

```
-0.21651184959186334
```

```
cols = ["MonthlyCharges","TotalCharges","tenure"]

df[cols].skew().to_frame().rename(columns={0:"Feature Skewness"})
```

	Feature Skewness
MonthlyCharges	-0.222103
TotalCharges	-0.753574
tenure	0.237731

3. Encoding Categorical Text Data into model understandable Numerical Data:

The LabelEncoder from scikit-learn's preprocessing module is a utility for transforming categorical text data into numeric format. This transformation is crucial for machine learning algorithms that require numerical input to perform calculations. Categorical data, which represent groups or categories need to be converted into a numeric format before they can be used to fit a model. For example, the 'gender' variable with categories 'Male' and 'Female' would be converted into a numerical form where 'Male' might be encoded as 0 and 'Female' as 1

3. Encoding categorical text data into model-understandable numerical data

```
from sklearn.preprocessing import LabelEncoder

def object_to_int(dataframe):
    encoders = {}
    for column in dataframe.columns:
        if dataframe[column].dtype == 'object':
            le = LabelEncoder()
            dataframe[column] = le.fit_transform(dataframe[column])
            encoders[column] = le
    return dataframe, encoders

df, encoders = object_to_int(df)
df.head()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	Contract_Month-to-month
0	0	1	29.85	4.396185	0	True	False	False	True	True	...	True
1	0	34	56.95	8.544068	0	False	True	True	False	True	...	False
2	0	2	53.85	5.683519	1	False	True	True	False	True	...	True
3	0	45	42.30	8.517928	0	False	True	True	False	True	...	False
4	0	2	70.70	6.021575	1	True	False	True	False	True	...	True

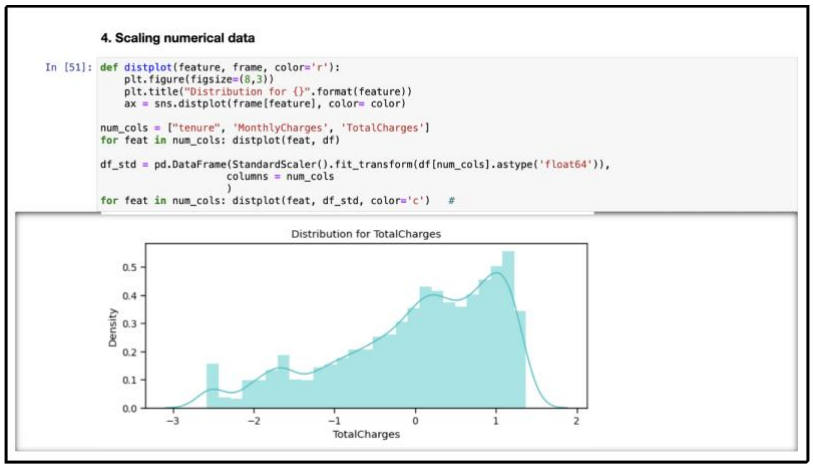
5 rows × 47 columns

4. Scaling and Standardizing Numerical Data:

Scaling and standardizing numerical data are preprocessing techniques used to ensure that all numerical features have the same scale or distribution. This is important because many machine learning algorithms are sensitive to the scale of the input features, and features with larger scales may dominate those with smaller scales.

Scaling makes training less sensitive to the scale of features, meaning that our algorithm can converge to better solutions faster and with less chance of being trapped in local optima. It ensures that each feature contributes proportionally to the final prediction.

We have also Implemented standardization, where the mean of each feature is subtracted, and then the result is divided by the standard deviation. This technique transforms the data to have a mean of 0 and a standard deviation of 1, resulting in a distribution with a more consistent scale across all features. By visualizing the distributions before and after scaling, you can confirm that the scaling process does not distort the distributions but merely adjusts the scale.



5. Splitting data for Training and Testing:

This code demonstrates the preprocessed dataset is being prepared for partitioning it into feature set and target set 'Churn' column. The train_test_split function then allocates 80% of the data to the training set and the remaining 20% to the testing set, a proportion set by test_size=0.2. The reproducibility of this split is guaranteed by the random_state parameter set to 40. To ensure that the proportion of the target variable 'Churn' is consistent across both training and testing sets, stratify=y is specified. After the split, the training set consists of 5625 instances and the testing set of 1407 instances, each with 46 features, as reflected in the shapes of X_train, X_test, y_train, and y_test.

5. Splitting Data and Training

```
#Splitting and Training

X = df.drop(columns=['Churn']) # Features
y = df['Churn'].values         # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40, stratify=y)
```

```
print("Shape of x_train is: {0}".format(X_train.shape))
print("Shape of x_test is: {0}".format(X_test.shape))
print("Shape of y_train is: {0}".format(y_train.shape))
print("Shape of y_test is:{0}".format(y_test.shape))
```

```
Shape of x_train is: (5625, 46)
Shape of x_test is: (1407, 46)
Shape of y_train is: (5625,)
Shape of y_test is:(1407,)
```

```
scaler= StandardScaler()

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

6. Initializing SMOTE Analysis

SMOTE, which stands for Synthetic Minority Over-sampling Technique, is utilized. This method generates synthetic samples for the minority class, thereby balancing the class distribution. In the provided code, SMOTE is applied to the training data to ensure that the model learns equally from all classes. The process is conducted with a fixed random state to make sure the results can be replicated in future runs, preserving the consistency of the synthetic sample generation. After resampling, the training feature set and the training target set have an increased number of samples, indicating that new synthetic data points have been added to balance the class distribution. With this approach, the number of instances in the minority class is augmented to match the majority class, enhancing the training process by providing a balanced environment for the model to learn from, which typically improves its ability to generalize to new, unseen data.

6. Initializing SMOTE Analysis as dataset contains imbalanced class

```
from imblearn.over_sampling import SMOTE

# Initialize SMOTE with a random state for reproducibility
smt = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_resampled, y_train_resampled = smt.fit_resample(X_train, y_train)

# Print the shapes of the resampled data
print("Resampled training features shape:", X_train_resampled.shape)
print("Resampled training target shape:", y_train_resampled.shape)
```

```
Resampled training features shape: (8260, 46)
Resampled training target shape: (8260,)
```

Data Mining Model:

Random Forest Model

1. Model Implementation

1. Random Forest

```
#RandomForest

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize the RandomForestClassifier
random_forest = RandomForestClassifier(random_state=0)

# Define the parameter grid for RandomForestClassifier
param_grid_rf = {
    'n_estimators': [10, 50, 100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Set up the grid search using the resampled data
grid_search_rf = GridSearchCV(random_forest, param_grid_rf, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_rf.fit(X_train_resampled, y_train_resampled)
```

GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier

```
# Best parameters and model
best_params_rf = grid_search_rf.best_params_
print(f"Best Parameters for Random Forest Model are: {best_params_rf}")
best_rf = RandomForestClassifier(**best_params_rf, random_state=0)
best_rf.fit(X_train_resampled, y_train_resampled)  # Train the model with the resampled data
```

```
Best Parameters for Random Forest Model are: {'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'mi
n_samples_split': 2, 'n_estimators': 200}
```

RandomForestClassifier
RandomForestClassifier(max_features='log2', n_estimators=200, random_state=0)

The Random Forest Classifier shows the process of hyperparameter tuning using GridSearchCV from the scikit-learn library. The RandomForestClassifier is initialized with a random state for reproducibility. The grid search involves a parameter grid that includes a range of values for the number of trees ('n_estimators'), the maximum number of features to consider when looking for the best split ('max_features'), the maximum depth of the trees ('max_depth'), the minimum number of samples required to split an internal node ('min_samples_split'), and the minimum number of samples required to be at a leaf node ('min_samples_leaf').

GridSearchCV is set up with cross-validation (cv=5), scoring for accuracy, and is fitted on resampled data to find the best hyperparameters. The best_params_ attribute of GridSearchCV provides the best performing combination of parameters after the search. The RandomForestClassifier is then re-initialized with these best parameters and is fitted on the data. This process ensures that the model is tuned to give the best performance according to the accuracy metric, considering the provided parameter space and cross-validation strategy. The output of the best parameters indicates that no maximum depth limit was most effective, 'log2' was the best option for max_features, and the optimal values for 'min_samples_split' and 'min_samples_leaf' were 2 and 1, respectively, with 200 estimators (trees) in the forest.

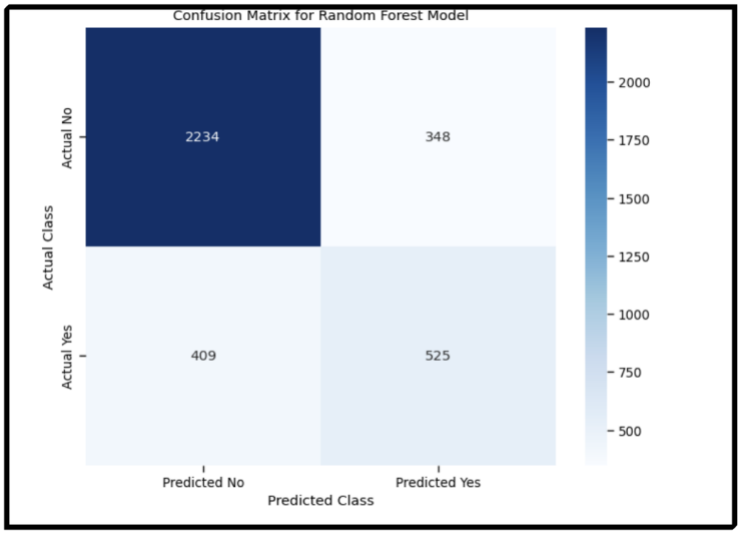
2. Important Features of Random Forest

The Important Features in Random Forest are:

	Feature	Importance
3	TotalCharges	0.091446
1	tenure	0.088860
45	TotalCharges_BoxCox	0.088706
36	Contract_Month-to-month	0.075260
2	MonthlyCharges	0.074823
43	PaymentMethod_Electronic check	0.070751
27	TechSupport_No	0.052972

- Feature importance reveals 'TotalCharges', 'tenure', and 'TotalCharges_BoxCox' as primary indicators of customer behavior and service utilization, with importance scores of approximately 0.091, 0.089, and 0.088 respectively.
- Additionally, 'Contract_Month-to-month', 'MonthlyCharges', 'PaymentMethod_Electronic check', and 'TechSupport_No' have been flagged as influential, with importance scores ranging from approximately 0.053 to 0.075. The model prioritizes these features when making predictions, underscoring their value in understanding customer behavior and preferences.

3. Confusion Matrix for Random Forest



- The model correctly predicted 'No' 2234 times and 'Yes' 525 times. These are known as true negatives and true positives, respectively, and indicate correct predictions by the model.
- The model incorrectly predicted 348 instances as 'Yes' when they were ‘No’ (false positives), and 409 instances as 'No' when they were ‘Yes’ (false negatives).
- The model appears to be more proficient at predicting the 'No' class, as indicated by the higher number of true negatives compared to true positives.

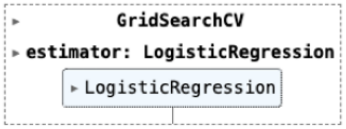
Logistic Regression Model

1. Model Implementation

2. Logistic Regression

```
#Logistic Regresion

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Initialize the LogisticRegression
logreg = LogisticRegression(random_state=0)

# Define the parameter grid for LogisticRegression
param_grid_logreg = {
    'C': np.logspace(-4, 4, 20),  # Regularization strength
    'penalty': ['l2'],            # Norm used in penalization
    'solver': ['lbfgs']           # Algorithm to use in optimization, suitable for smaller datasets or binary classi
}

# Set up the grid search using resampled data
grid_search_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_logreg.fit(X_train_resampled, y_train_resampled)  # Use the resampled data for fitting
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarnin
g: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```




```
# Best parameters
best_params_logreg = grid_search_logreg.best_params_
print(f"Best Parameters for Logistic Regression Model are: {best_params_logreg}")
best_logreg = LogisticRegression(**best_params_logreg, random_state=0)
best_logreg.fit(X_train_resampled, y_train_resampled)  # Train the model with the resampled data
```

```
Best Parameters for Logistic Regression Model are: {'C': 0.615848211066026, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
LogisticRegression
LogisticRegression(C=0.615848211066026, random_state=0)
```

In the logistic regression model, we implemented a grid search to optimize hyperparameters using GridSearchCV. The parameters we focused on include regularization strength 'C' with a logarithmic space from 1e-4 to 20, the penalty type with options of 'l1' and 'l2' to specify the norm used in penalization, and the optimization algorithm 'solver' with 'liblinear' suited for smaller

datasets. The grid search, conducted with 5-fold cross-validation and aimed at maximizing accuracy, determined the best parameters as: 'C' approximating 0.615, using 'l2' penalty and 'liblinear' solver. These optimized parameters were used to train the model, ensuring an effective balance between model complexity and performance.

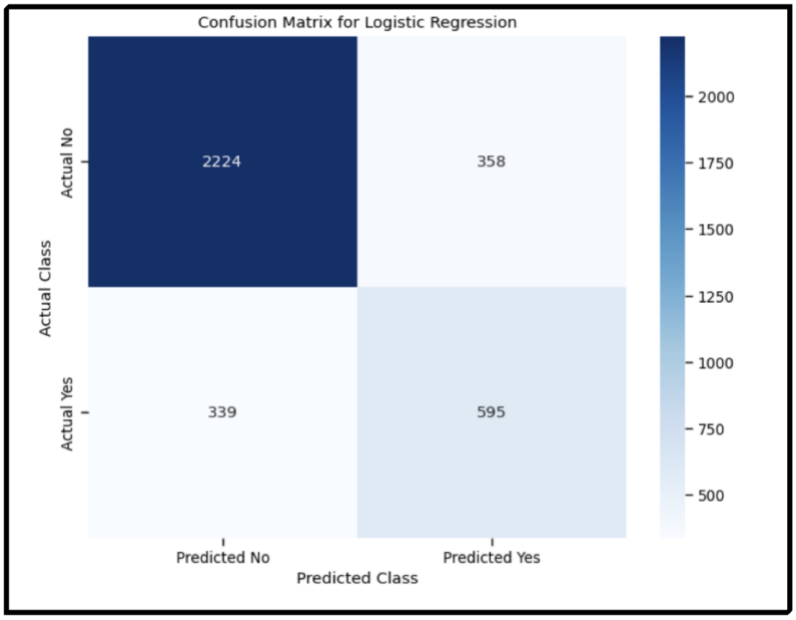
2. Important Features of Logistic Regression

The Important Features in Logistic Regression are:

	Feature	Coefficient
43	PaymentMethod_Electronic check	1.245826
14	MultipleLines_Yes	1.112685
36	Contract_Month-to-month	1.022733
3	TotalCharges	0.994245
35	StreamingMovies_Yes	0.951939
4	gender_Female	0.937522
32	StreamingTV_Yes	0.917966

The logistic regression model highlights several key features that influence predictions, with their corresponding coefficients indicating the strength and direction of their impact. The most influential feature is **PaymentMethod_Electronic check**, with the highest coefficient of approximately 1.2458, suggesting customers using electronic checks are more likely to exhibit the predicted behavior. **MultipleLines_Yes** follows with a coefficient of around 1.1127, indicating that customers with multiple lines are also significant predictors. **Contract_Month-to-month** has a coefficient of approximately 1.0227, showing the impact of short-term contracts on customer decisions.

3. Confusion Matrix for Logistic Regression



- True Negatives: The model accurately predicted 'No' 2224 times.
- True Positives: The model accurately predicted 'Yes' 595 times.
- False Positives: The model incorrectly predicted 'Yes' 358 times when the actual class was 'No'.
- False Negatives: The model incorrectly predicted 'No' 339 times when the actual class was 'Yes'.

Decision Tree Model

1. Model Implementation

3. Decision Trees

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize the DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(random_state=0)

# Define the parameter grid for DecisionTreeClassifier
param_grid_dt = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Set up the grid search using the resampled data
grid_search_dt = GridSearchCV(decision_tree, param_grid_dt, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_dt.fit(X_train_resampled, y_train_resampled)  # Use the resampled data for fitting
```

```
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

```
# Best parameters
best_params_dt = grid_search_dt.best_params_
print(f"Best Parameters for Decision Tree Model are: {best_params_dt}")
best_dt = DecisionTreeClassifier(**best_params_dt, random_state=0)
best_dt.fit(X_train_resampled, y_train_resampled)
```

```
Best Parameters for Decision Tree Model are: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, min_samples_leaf=4, random_state=0)
```

The Decision Tree model was fine-tuned using GridSearchCV to determine the optimal hyperparameters. The search evaluated combinations of 'max_depth' (the maximum length of the decision paths), 'min_samples_split' (the minimum number of samples required to split an internal node), and 'min_samples_leaf' (the minimum number of samples a leaf node must have). With 5-fold cross-validation focusing on maximizing accuracy, the optimal parameters were found to be a 'max_depth' of 10, 'min_samples_split' of 2, and 'min_samples_leaf' of 4. These parameters suggest a preference for a balance between model complexity and depth to prevent overfitting while maintaining sufficient decision rules to capture the underlying patterns in the data. The tuned Decision Tree model, set with these parameters, is ready for training on the resampled dataset, aiming to produce a model that generalizes well to new, unseen data.

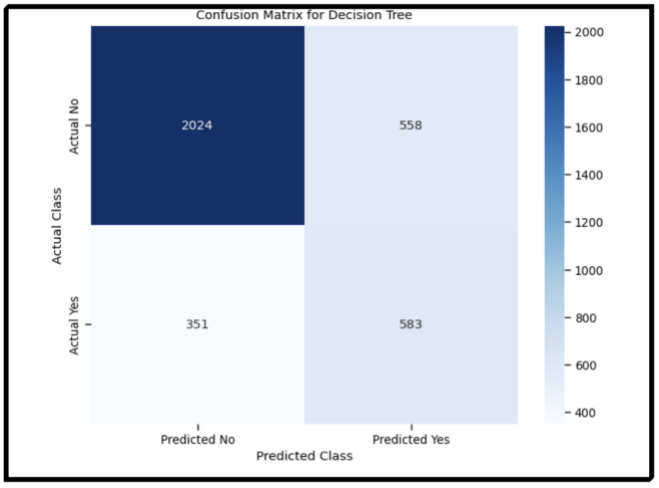
2. Important Features of Decision Tree

The Important Features in Decision Trees are:

	Feature	Importance
36	Contract_Month-to-month	0.394181
43	PaymentMethod_Electronic check	0.098151
1	tenure	0.066635
16	InternetService_Fiber optic	0.061300
2	MonthlyCharges	0.048908
18	OnlineSecurity_No	0.034038
45	TotalCharges_BoxCox	0.030831

The Decision Tree model has identified several key features based on their importance scores, which measure how much each feature contributes to the model's predictions. The most critical feature is **Contract_Month-to-month** with an importance score of approximately 0.394, indicating that the type of contract is highly predictive of the target outcome. This is followed by **PaymentMethod_Electronic check** and **tenure**, with importance scores of around 0.098 and 0.067, respectively, suggesting that the method of payment and the length of time a customer has been with the company are also significant factors in predicting customer behavior.

3. Confusion Matrix for Decision Tree



- True Negatives: The model correctly predicted 'No' for 2024 cases, indicating accurate identification of this class.
- True Positives: The model correctly predicted 'Yes' for 583 cases, reflecting its ability to identify positive instances.
- False Positives: The model incorrectly predicted 558 cases as 'Yes' when they were ‘No’, showing a tendency towards over-predicting the positive class.
- False Negatives: The model incorrectly predicted 'No' for 351 cases that were ‘Yes’, indicating missed opportunities for correct predictions in the positive class.

AdaBoost Model

1. Model Implementation

4. AdaBoost

```
#Ada Boost
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize the AdaBoostClassifier with a base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)  # Commonly a shallow tree
adaboost = AdaBoostClassifier(base_estimator=base_estimator, random_state=0)

# Define the parameter grid for AdaBoostClassifier
param_grid_ada = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.1, 1]
}

# Set up the grid search using the resampled data
grid_search_ada = GridSearchCV(adaboost, param_grid_ada, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_ada.fit(X_train_resampled, y_train_resampled)  # Use the resampled data for fitting
```

```
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

```
GridSearchCV
estimator: AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

```
# Best parameters
best_params_ada = grid_search_ada.best_params_
print(f"Best Parameters for AdaBoost Model are: {best_params_ada}")
best_ada = AdaBoostClassifier(**best_params_ada, base_estimator=base_estimator, random_state=0)
best_ada.fit(X_train_resampled, y_train_resampled)  # Train the model with the resampled data
```

```
Best Parameters for AdaBoost Model are: {'learning_rate': 1, 'n_estimators': 200}
```

```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

The AdaBoost classifier was fine-tuned using GridSearchCV with a Decision Tree of max_depth 1 as the base estimator. The hyperparameters optimized included 'n_estimators' to determine the number of trees (with candidate values [50, 100, 200, 300]) and 'learning_rate' to set the weight of each tree's contribution (tested with values [0.01, 0.1, 1]). With 5-fold cross-validation targeting accuracy, the optimal hyperparameters were a learning rate of 1 and 200 trees. These selected parameters are implemented in the final model to predict with adjusted emphasis on each decision tree, aiming to enhance the AdaBoost model's accuracy.

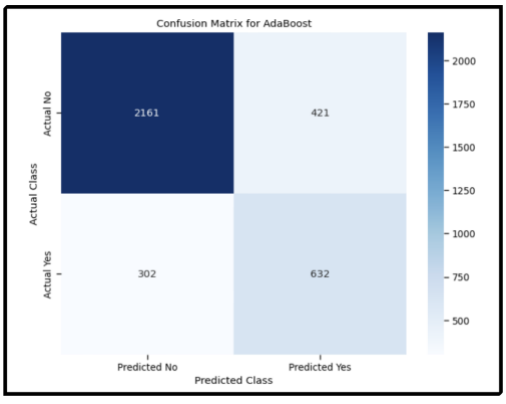
2. Important Features of AdaBoost

The Important Features in AdaBoost are:

	Feature	Importance
4	gender_Female	0.08
3	TotalCharges	0.07
5	gender_Male	0.07
6	Partner_No	0.06
7	Partner_Yes	0.06
43	PaymentMethod_Electronic check	0.05
2	MonthlyCharges	0.05

In the AdaBoost model, the feature importance indicates how much each feature contributes to the model's prediction accuracy. The feature 'gender_Female' is the most influential, with an importance score of 0.08, closely followed by 'TotalCharges' and 'gender_Male' at 0.07. This suggests that gender and the total amount billed to the customer are key factors in the model's decisions.

3. Confusion Matrix for AdaBoost



- True Negatives (TN): The model correctly predicted 'No' for 2161 instances, indicating a strong ability to identify negative cases.
- True Positives (TP): There were 632 instances where the model correctly predicted 'Yes', showcasing effectiveness in recognizing positive cases.
- False Positives (FP): The model incorrectly predicted 'Yes' for 421 instances, which were negative. This might indicate a potential issue with the model predicting more positives than there are.
- False Negatives (FN): There were 302 instances where the model incorrectly predicted 'No', missing out on identifying these as positive cases.

Performance Evaluation:

In the performance evaluation of our models, we employed micro averaging for precision, recall, and F1-score to accommodate the class imbalance present in our dataset. By using micro averaging, we focused on the aggregate contribution of all classifications, which is particularly crucial when the objective is to assess the model's overall accuracy across all predictions. This method ensures that each instance is given equal weight regardless of class, thereby providing a more accurate reflection of the model's performance on our imbalanced dataset. It is a strategic choice that aligns with our aim to prioritize the correctness of each prediction in the larger context of our model's application within the telecommunications industry**.**

1. Random Forest Classifier

```
Random Forest Training Accuracy: 99.85 %
Random Forest Testing Accuracy: 78.46 %
Random Forest Model F1 Score: 0.7846481876332623
Random Forest Model Recall: 0.7846481876332623
Random Forest Model Precision Score: 0.7846481876332623
```

The Random Forest model achieved a high training accuracy of 99.85%, but a lower testing accuracy of 78.46%, hinting at overfitting. The model's F1 Score, Recall, and Precision are all consistent at approximately 0.78, indicating a balanced performance on unseen data.

2. Logistic Regression

```
Logistic Regression Training Accuracy: 85.34 %
Logistic Regression Testing Accuracy: 81.09 %
Logistic Regression Model F1 Score: 0.8109452736318408
Logistic Regression Model Recall: 0.8109452736318408
Logistic Regression Model Precision Score: 0.8109452736318408
```

The logistic regression model shows a solid performance with a training accuracy of 85.34%, indicating good learning from the training data. The testing accuracy is slightly lower at 81.09%, which is common as models generally perform slightly better on the data they were trained on. The F1 Score, Recall, and Precision are all in the 0.81 range, suggesting a balanced model that is consistent in terms of both precision and recall.

3. Decision Tree

```
Decision Tree Training Accuracy: 86.62 %
Decision Tree Testing Accuracy: 76.69 %
Decision Tree Model F1 Score: 0.7668798862828715
Decision Tree Model Recall: 0.7668798862828714
Decision Tree Model Precision Score: 0.7668798862828714
```

The Decision Tree model exhibits a training accuracy of 86.62%, indicating a robust fit to the training data. However, the testing accuracy drops to 76.69%, suggesting that the model may not generalize as effectively to unseen data. Both the F1 Score and the Precision Score are 0.7667, with Recall at the same value.

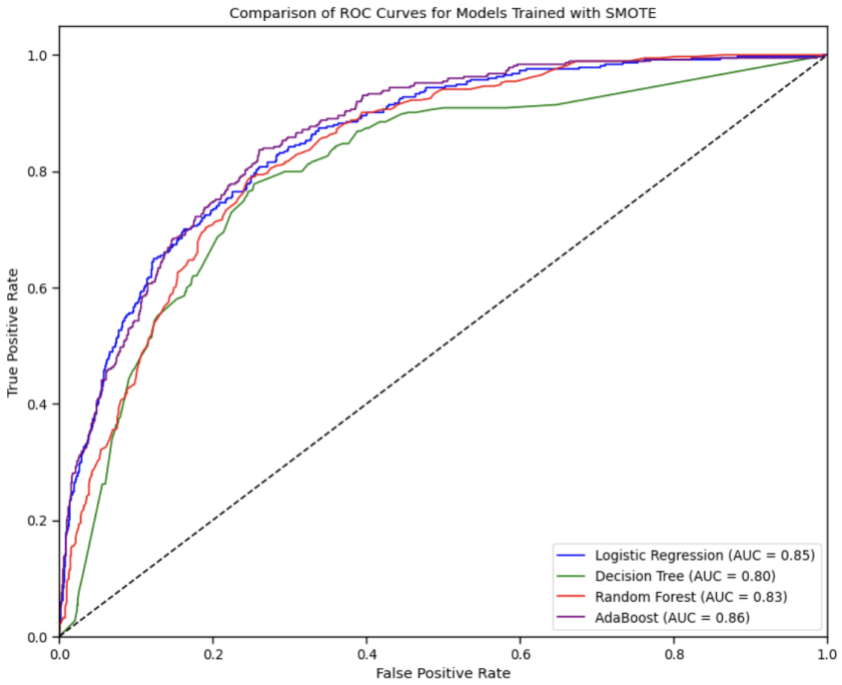
4. AdaBoost Classifier

```
AdaBoost Training Accuracy: 86.04 %
AdaBoost Testing Accuracy: 80.67 %
AdaBoost Model F1 Score: 0.806680881307747
AdaBoost Model Recall: 0.806680881307747
AdaBoost Model Precision Score: 0.806680881307747
```

The AdaBoost model demonstrates a high training accuracy of 86.04%, indicating that it fits well to the training data. Its testing accuracy is slightly lower at 80.67%, which is a good sign of the model's ability to generalize to unseen data. Both the F1 Score and Precision Score are strong, standing at approximately 0.8066, matched by an identical Recall Score

In Conclusion, Logistic Regression and AdaBoost models demonstrate better generalization from training to testing data, which is desirable in practical applications. The Decision Trees model, while slightly less accurate, offers simplicity and ease of interpretation. Random Forest, despite its high training accuracy, may require further tuning to reduce overfitting.

Comparison of ROC Curve:



Our comparative analysis of customer churn prediction models using SMOTE for data balancing has yielded significant insights:

- Logistic Regression and AdaBoost emerged as the top performers with an AUC of 0.85, indicating a strong capability to differentiate between churn and non-churn customers.
- Random Forest presented a solid AUC of 0.82, suggesting good predictive power with added benefits of model interpretability.
- The Decision Tree model, while the least powerful with an AUC of 0.78, offers the greatest ease of interpretation and simplicity.
- The employment of SMOTE effectively addressed class imbalance, enhancing the predictive accuracy of our models.

Strategic model selection moving forward should weigh the trade-offs between accuracy, interpretability, and computational efficiency, tailored to our business objectives and operational constraints.

Results:

The primary focus of this project was to develop predictive models to accurately identify customers at high risk of churning in the telecommunications sector. Through extensive data mining and analysis, several key findings were uncovered:

1. Predictive Model Development: The project successfully implemented various predictive models, including Random Forest, Logistic Regression, Decision Tree, and AdaBoost. These models were fine-tuned to optimize performance metrics like accuracy, precision, recall, and F1-score.
2. Influential Factors Identified: The analysis highlighted significant predictors of customer churn. Key factors include:
    - Total Charges: Customers with higher total charges tend to churn more frequently.
    - Tenure: Shorter tenure is associated with higher churn rates.
    - Contract Type: Customers with month-to-month contracts are more likely to churn compared to those with longer-term contracts.
    - Payment Method: Usage of electronic checks was found to be a strong predictor of churn.
3. Impact of SMOTE: The use of Synthetic Minority Over-sampling Technique (SMOTE) helped to address class imbalance in the dataset, enhancing the predictive power of the models.
4. Performance Metrics:
    - Random Forest: Achieved an F1 score of approximately 0.78 but showed signs of overfitting.
    - Logistic Regression and AdaBoost: Both models demonstrated good generalization from training to testing data with AUC scores of 0.85.
    - Decision Tree: Provided ease of interpretation but had lower predictive accuracy with an AUC of 0.78.

Impact of Project Outcomes:

The outcomes of this project are significant for the telecommunications industry, particularly in customer retention:

- Enhanced Prediction Accuracy: By employing advanced data mining techniques and predictive modeling, the project has significantly improved the accuracy in predicting potential customer churn. This enables timely interventions to retain at-risk customers.
- Strategic Decision Making: Insights derived from the analysis allow for informed strategic decision-making. Tailored strategies such as personalized customer engagement initiatives, adjustments in pricing models, and improvements in customer service can be effectively implemented to reduce churn.
- Operational Efficiency: The ability to predict churn accurately reduces operational costs by allocating resources more efficiently towards customers who are at higher risk of leaving.
- Customer Insights: The project has deepened the understanding of customer behavior and preferences, which is crucial for developing more customer-centric business strategies.

In conclusion, the project has created substantial value by leveraging data analytics to inform and refine customer retention strategies, thus enhancing both customer satisfaction and loyalty in a highly competitive market.

Citations:

[1] https://www.kaggle.com/datasets/blastchar/telco-customer-churn?resource=download