Table of Contents

PROJECT REPORT
TELCO CUSTOMER CHURN

INTRODUCTION

Customer retention is one of the most important challenges faced by businesses today, especially in highly competitive industries like telecommunications. Customer churn, often called customer attrition, refers to the rate at which customers stop using a company's service or cancel their subscriptions. In the telecommunications industry, this is a critical issue – a high churn rate directly translates to lost revenue, and it is one of the most important concerns for telecom providers. For these reasons, understanding and predicting churn is crucial for telecom companies to maintain a stable customer base and ensure long-term success.

Challenges of Customer Churn in the Telecommunications Industry

Customer churn presents a significant challenge for telecommunication companies. When customers leave, the company not only loses revenue but also incurs additional costs to acquire new customers to replace those lost. Studies have shown that acquiring a new customer can be up to five times more expensive than retaining an existing one. This makes churn a costly problem that directly affects a company's profitability and long-term growth.

Key Challenges Faced:

1. **Loss of Revenue:** Every customer who leaves takes away a recurring source of income, which is especially critical in subscription-based models like telecom services.
2. **Increased Acquisition Costs:** To make up for lost customers, companies must spend more on marketing and promotional offers to attract new ones.
3. **Damage to Brand Reputation:** High churn rates may signal poor service or customer dissatisfaction, potentially damaging the company's image and leading to more customers leaving.
4. **Unstable Customer Base:** A fluctuating customer base can make it harder to forecast revenue, plan infrastructure investments, or scale services efficiently.
5. **Missed Upsell Opportunities:** Loyal, long-term customers often engage with more services. Losing them means missing out on future cross-sell or upsell opportunities.

Why Does Churn Happen?

Customer churn can occur for many reasons, often a combination of service quality, pricing, competition, and personal preferences. The most common causes include:

1. **Better Offers from Competitors:** In a competitive market, customers may switch providers for lower prices, better plans, or more attractive deals.
2. **Poor Customer Service:** Long wait times, unresolved complaints, or impolite service can lead to dissatisfaction and ultimately cause customers to leave.
3. **Service Quality Issues:** Network outages, slow internet, or dropped calls can frustrate users and push them to look for more reliable alternatives.
4. **Billing and Pricing Concerns:** Unexpected charges, confusing bills, or lack of billing transparency can reduce trust in the company.

5. **Lack of Engagement or Loyalty Programs:** Customers who feel unappreciated or who are not incentivized to stay may feel less loyal and more open to switching.
6. **Life Changes or Relocation:** Customers may move to a location not serviced by the provider, or simply no longer require the service due to a lifestyle change.

This project focuses on understanding and predicting customer churn using the Telco Customer Churn dataset provided by IBM, taken from Kaggle. The dataset contains detailed information about over 7,000 customers, including their demographic details (such as gender and family status), the services they use (like internet, phone, and streaming), account information (such as contract type, billing method, and tenure), and whether or not they have recently churned.

OBJECTIVE

The goal of this project is to study customer churn in the telecommunications sector using real-world data. By analyzing historical customer information and applying machine learning techniques, the aim is to uncover key reasons why customers leave and develop a predictive model that can help telecom companies take preventive action.

The detailed objectives of this project are:

- **To understand the problem of customer churn:** Gain a clear understanding of how customer churn affects telecom companies financially and strategically, and why it is important to predict and reduce churn.
- **To explore and analyze customer behavior through data:** Perform exploratory data analysis (EDA) to uncover trends, patterns, and relationships in customer attributes (such as tenure, services subscribed, and billing preferences) that influence churn decisions.
- **To prepare and preprocess the dataset:** Clean the dataset by handling missing values, converting categorical data into numerical form using label encoding, normalizing numeric values with RobustScaler, and ensuring consistent column formatting.
- **To identify and address data imbalance:** Use techniques such as SMOTE (Synthetic Minority Oversampling Technique) to balance the classes and ensure that the model does not become biased towards predicting only the majority class (non-churners).
- **To build predictive machine learning models:** Train and fine-tune multiple supervised machine learning models, such as Logistic Regression, Random Forest, and Gradient Boosting, to classify whether a customer is likely to churn.
- **To evaluate model performance:** Compare the performance of the models using classification metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Select the most effective model based on its ability to identify potential churners.
- **To visualize results and insights:** Use visual tools like confusion matrices, ROC curves, learning curves, and performance comparison graphs to interpret model performance and validate findings.
- **To provide business recommendations:** Translate the technical results into actionable insights that a telecom company can use to design targeted retention strategies, improve customer experience, and reduce churn.

ABOUT THE DATASET

This project uses the **Telco Customer Churn** dataset, which contains detailed information about **7,043 customers** and **21 features** of a telecommunications company. Each row in the dataset represents a single customer, and each column provides data related to their personal details, services subscribed, billing preferences, and whether or not they churned. The dataset is well-suited for analyzing churn behavior and building predictive models.

FIELD	DESCRIPTION
customerID	A unique ID assigned to each customer.
gender	Gender of the customer: Male or Female.
SeniorCitizen	Indicates whether the customer is a senior citizen (1) or not (0).
Partner	Whether the customer has a partner (Yes/No).
Dependents	Whether the customer has dependents (Yes/No).
tenure	Number of months the customer has stayed with the company.
PhoneService	Indicates whether the customer has a phone service (Yes/No).
MultipleLines	Whether the customer has multiple phone lines (Yes/No/No phone service).
InternetService	Type of internet service: DSL, Fiber optic, or No.
OnlineSecurity	Whether the customer has online security add-on (Yes/No/No internet service).
OnlineBackup	Whether the customer has online backup add-on (Yes/No/No internet service).
DeviceProtection	Whether the customer has device protection add-on (Yes/No/No internet service).
TechSupport	Whether the customer has technical support add-on (Yes/No/No internet service).
StreamingTV	Whether the customer streams TV (Yes/No/No internet service).
StreamingMovies	Whether the customer streams movies (Yes/No/No internet service).
Contract	Type of contract: Month-to-month, One year, or Two year.
PaperlessBilling	Whether the customer has paperless billing enabled (Yes/No).
PaymentMethod	Method of payment (e.g., Electronic check, Mailed check, Credit card, etc.).
MonthlyCharges	The amount charged to the customer every month.
TotalCharges	The total amount charged to the customer over the entire period.
Churn	Whether the customer has churned (Yes/No).

By analyzing the relationships between these features and customer churn behavior, this dataset enables the development of predictive models that can help telecom companies identify at-risk customers and take proactive steps to improve retention.

TOOLS AND LIBRARIES USED

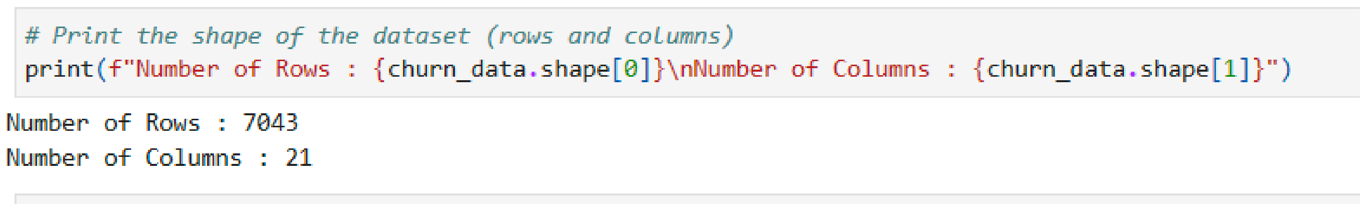
Programming Language: Python

To analyze the data and build predictive models, we used Python due to its simplicity and the wide range of libraries available for data science and machine learning. These tools made it easier to manipulate data, visualize trends, and train robust models. The key libraries used in this project include:

- **Pandas:** For data loading, cleaning, and manipulation.
- **NumPy:** For efficient numerical operations and handling arrays.
- **Matplotlib & Seaborn:** For creating informative and visually appealing data visualizations.
- **Scikit-learn:** For preprocessing data, building machine learning models, tuning hyperparameters, and evaluating model performance.
- **Imbalanced-learn (imblearn):** For applying SMOTE to address class imbalance.

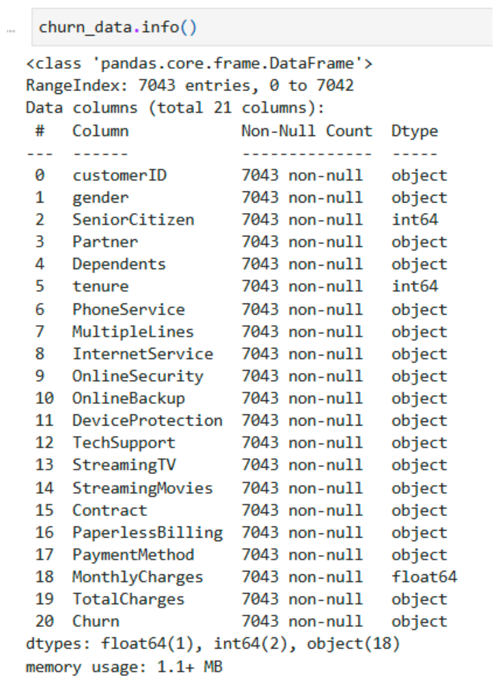
These tools and libraries helped streamline the analysis process, extract meaningful insights, and efficiently develop models to predict customer churn.

DATA EXPLORATION



```
# Print the shape of the dataset (rows and columns)
print(f"Number of Rows : {churn_data.shape[0]}\nNumber of Columns : {churn_data.shape[1]}")
```

```
Number of Rows : 7043
Number of Columns : 21
```

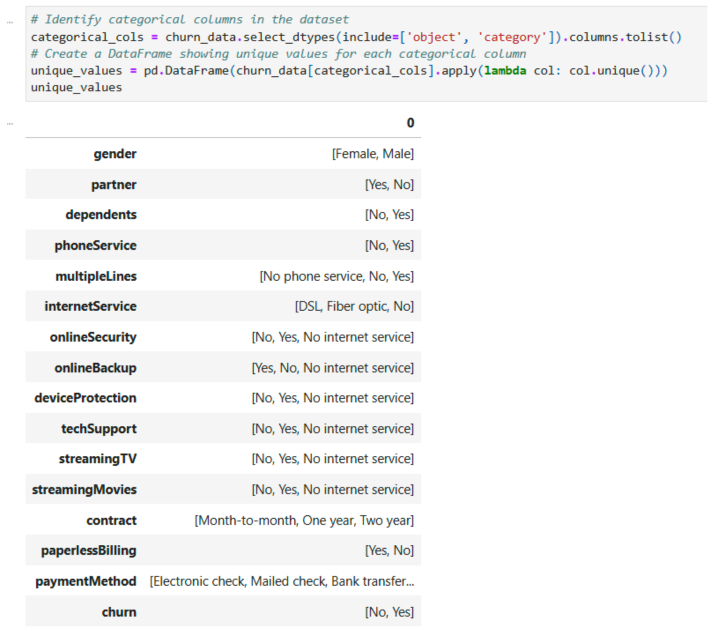


```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

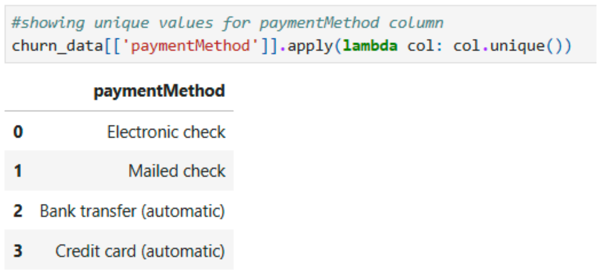
The dataset contains 7043 entries and 21 columns. The data includes a mix of numerical (int64, float64) and categorical (object) features, representing churn-related information such as number of lines, tech support, and churn status. This ensures the dataset is well-structured for further preprocessing and analysis.

Categorical Columns



```
# Identify categorical columns in the dataset
categorical_cols = churn_data.select_dtypes(include=['object', 'category']).columns.tolist()
# Create a DataFrame showing unique values for each categorical column
unique_values = pd.DataFrame(churn_data[categorical_cols].apply(lambda col: col.unique()))
unique_values
```

	0
gender	[Female, Male]
partner	[Yes, No]
dependents	[No, Yes]
phoneService	[No, Yes]
multipleLines	[No phone service, No, Yes]
internetService	[DSL, Fiber optic, No]
onlineSecurity	[No, Yes, No internet service]
onlineBackup	[Yes, No, No internet service]
deviceProtection	[No, Yes, No internet service]
techSupport	[No, Yes, No internet service]
streamingTV	[No, Yes, No internet service]
streamingMovies	[No, Yes, No internet service]
contract	[Month-to-month, One year, Two year]
paperlessBilling	[Yes, No]
paymentMethod	[Electronic check, Mailed check, Bank transfer...
churn	[No, Yes]



```
#showing unique values for paymentMethod column
churn_data[['paymentMethod']].apply(lambda col: col.unique())
```

	paymentMethod
0	Electronic check
1	Mailed check
2	Bank transfer (automatic)
3	Credit card (automatic)

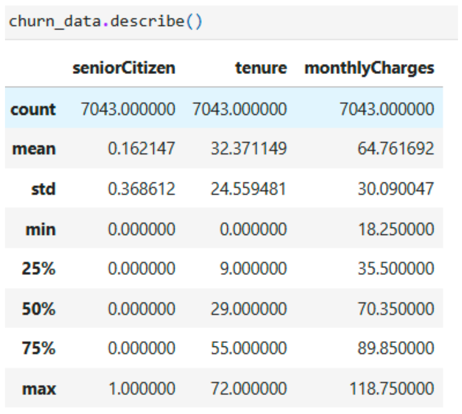
We also identified and analyzed all **categorical features** using value counts and the unique() function. Columns such as gender, partner, dependents, phoneService, and multipleLines contain binary or multi-class categorical data. For example:

- **InternetService** has values: *DSL, Fiber optic, No*.
- **Contract** includes: *Month-to-month, One year, Two year*.
- **PaymentMethod** has four types: *Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic)*.

The churn column is our target variable and has two values: *Yes* (customer left) and *No* (customer stayed).

KEY STATISTICS

The following summary statistics were obtained for the numerical columns in the dataset:



```
churn_data.describe()
```

	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

- **Senior Citizen:**
This column is binary, with 0 representing non-senior citizens and 1 representing senior citizens.

  - Mean: 0.16 – Indicates that only about 16% of the customers are senior citizens.
  - Standard Deviation: 0.37 – Reflects the binary nature with some variation.

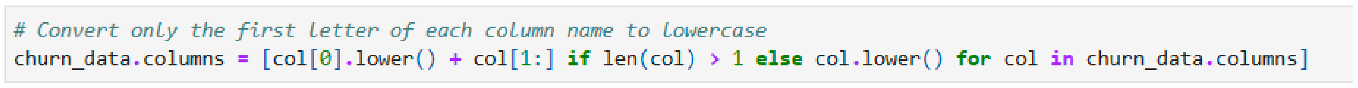
- **Tenure:**
This represents how long a customer has been with the company, in months.
  - Min: 0 months – Some customers are brand new.
  - Max: 72 months – Longest recorded customer relationship.
  - Mean: 32.37 months – On average, customers have been with the company for about 2.7 years.
  - Median (50%): 29 months

    - 25th and 75th percentiles: 9 and 55 months respectively – Shows a wide range in tenure.

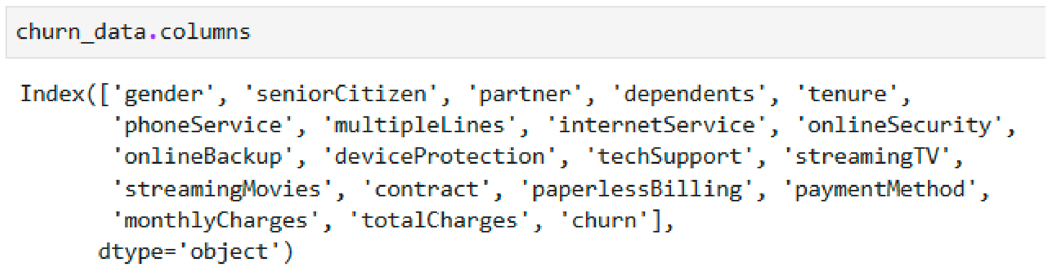
- **Monthly Charges**:
  This column captures the amount charged to a customer per month.
    - **Min**: $18.25
    - **Max**: $118.75
    - **Mean**: $64.76 – On average, customers are paying around $65 monthly.
    - **Median (50%)**: $70.35
    - **Standard Deviation**: $30.09 – There is a considerable spread in customer bills, which suggests different service combinations.

These insights give us an initial understanding of customer demographics and billing behavior, which is essential for identifying patterns that may influence churn.

DATA CLEANING



```
# Convert only the first letter of each column name to lowercase
churn_data.columns = [col[0].lower() + col[1:] if len(col) > 1 else col.lower() for col in churn_data.columns]
```

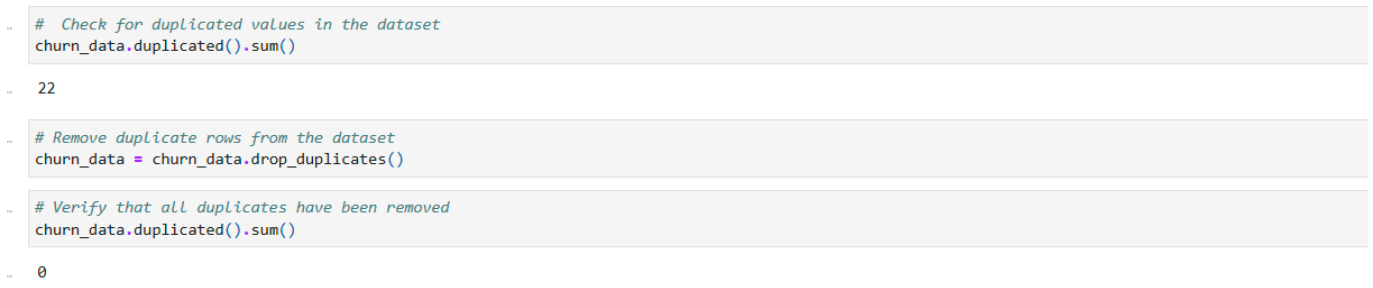


```
churn_data.columns
```

```
Index(['gender', 'seniorCitizen', 'partner', 'dependents', 'tenure',
       'phoneService', 'multipleLines', 'internetService', 'onlineSecurity',
       'onlineBackup', 'deviceProtection', 'techSupport', 'streamingTV',
       'streamingMovies', 'contract', 'paperlessBilling', 'paymentMethod',
       'monthlyCharges', 'totalCharges', 'churn'],
      dtype='object')
```

In this step, we standardized the column names by converting the first letter of each column name to lowercase. This helps maintain consistency and prevents potential issues caused by case sensitivity during data manipulation and analysis.

DUPLICATED VALUES:



```
#  Check for duplicated values in the dataset
churn_data.duplicated().sum()
```

```
22
```

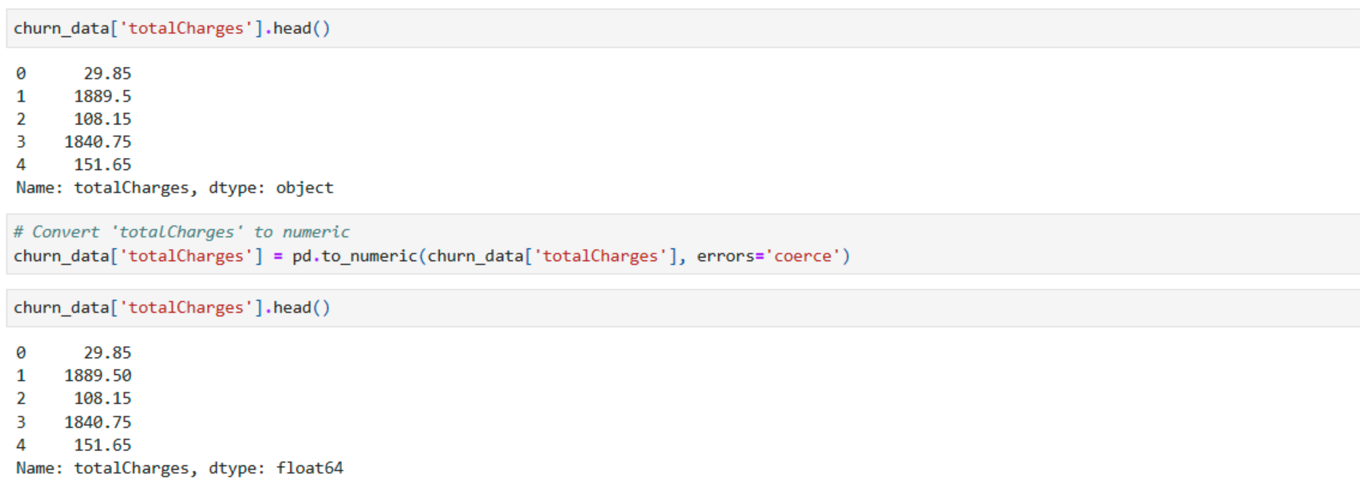
```
# Remove duplicate rows from the dataset
churn_data = churn_data.drop_duplicates()
```

```
# Verify that all duplicates have been removed
churn_data.duplicated().sum()
```

```
0
```

Here, we identified and removed duplicate rows from the dataset. Initially, we found 22 duplicated records, which were then removed using drop_duplicates(). A final check confirmed that no duplicate entries remained, ensuring data quality for further analysis.

DATA TYPE CONVERSION:



```
churn_data['totalCharges'].head()
```

```
0      29.85
1     1889.5
2     108.15
3    1840.75
4     151.65
Name: totalCharges, dtype: object
```

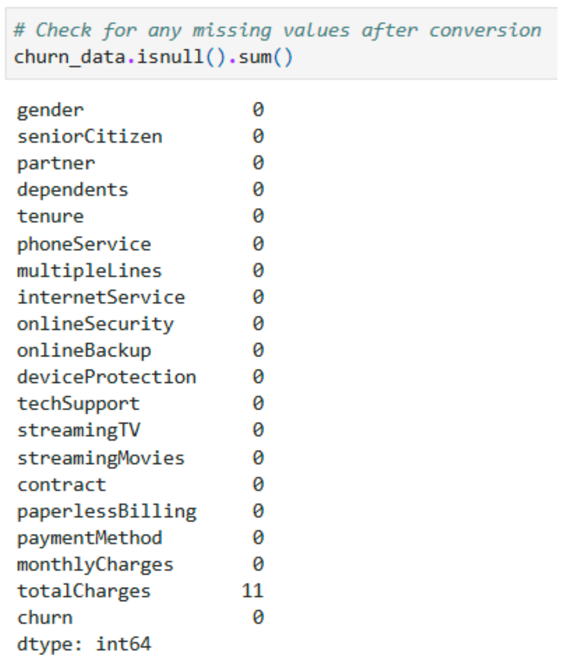
```
# Convert 'totalCharges' to numeric
churn_data['totalCharges'] = pd.to_numeric(churn_data['totalCharges'], errors='coerce')
```

```
churn_data['totalCharges'].head()
```

```
0      29.85
1    1889.50
2     108.15
3    1840.75
4     151.65
Name: totalCharges, dtype: float64
```

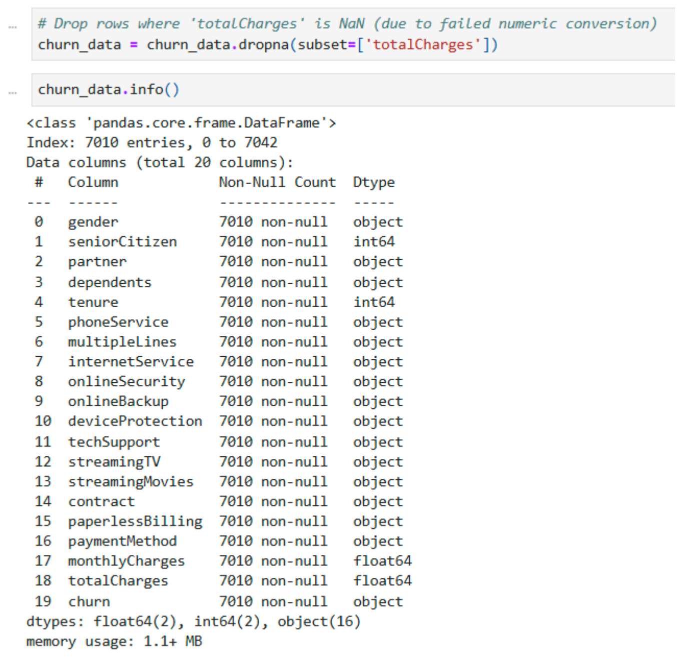
The **totalCharges** column was initially stored as an object (string) due to formatting issues like blank entries or spaces. To perform numerical operations, we converted it to float using pd.to_numeric() with errors='coerce', which safely converts valid numbers and replaces invalid entries with NaN. This ensures consistent data types for accurate analysis and modeling.

CHECK MISSING VALUES AFTER CONVERSION



```
# Check for any missing values after conversion
churn_data.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
```



```
# Drop rows where 'totalCharges' is NaN (due to failed numeric conversion)
churn_data = churn_data.dropna(subset=['totalCharges'])
```

```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7010 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7010 non-null   object
 1   seniorCitizen     7010 non-null   int64
 2   partner           7010 non-null   object
 3   dependents        7010 non-null   object
 4   tenure            7010 non-null   int64
 5   phoneService      7010 non-null   object
 6   multipleLines     7010 non-null   object
 7   internetService   7010 non-null   object
 8   onlineSecurity    7010 non-null   object
 9   onlineBackup      7010 non-null   object
 10  deviceProtection  7010 non-null   object
 11  techSupport       7010 non-null   object
 12  streamingTV       7010 non-null   object
 13  streamingMovies   7010 non-null   object
 14  contract          7010 non-null   object
 15  paperlessBilling  7010 non-null   object
 16  paymentMethod     7010 non-null   object
 17  monthlyCharges    7010 non-null   float64
 18  totalCharges      7010 non-null   float64
 19  churn             7010 non-null   object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

After the conversion, it was found that 11 rows had missing values in the totalCharges column. These rows were removed from the dataset using dropna(), resulting in a cleaned dataset with 7010 rows and 20 columns.

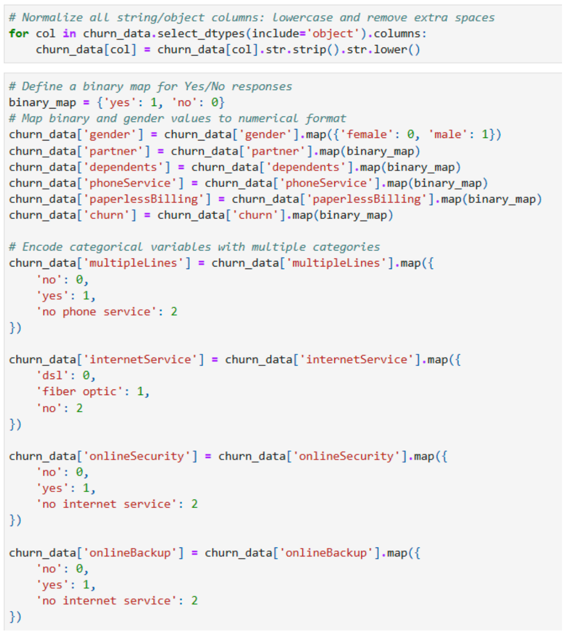
A final check using info() confirmed that all columns now have valid data types—numerical or categorical—and that there are no missing values left in the dataset.

➢ Encoding Categorical Variables to Numerical Values

To prepare the dataset for machine learning models, it is essential to convert all categorical variables into numerical formats. This step ensures compatibility with algorithms that require numeric inputs. We started by normalizing all string values—converting them to lowercase and stripping extra spaces for consistency.

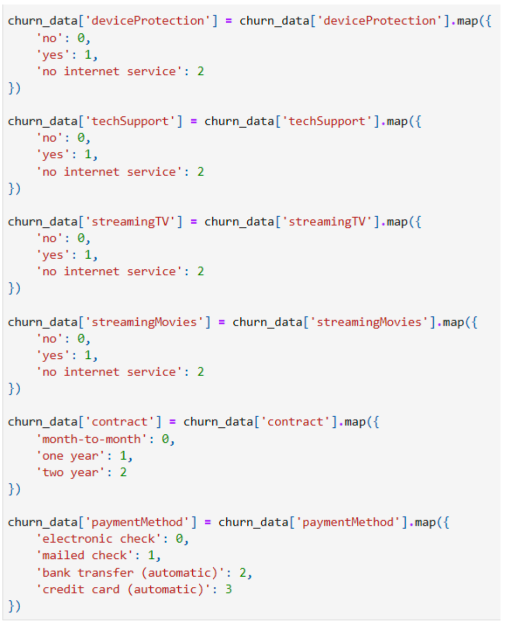
Binary categorical features such as yes/no responses were mapped using a simple dictionary (yes: 1, no: 0). For features with multiple categories like internetService, contract, and paymentMethod, custom mappings were applied to assign numerical codes to each category.

As a result, all object-type columns were successfully converted to integer values. This transformation made the dataset fully numeric and suitable for training various machine learning models without preprocessing errors.

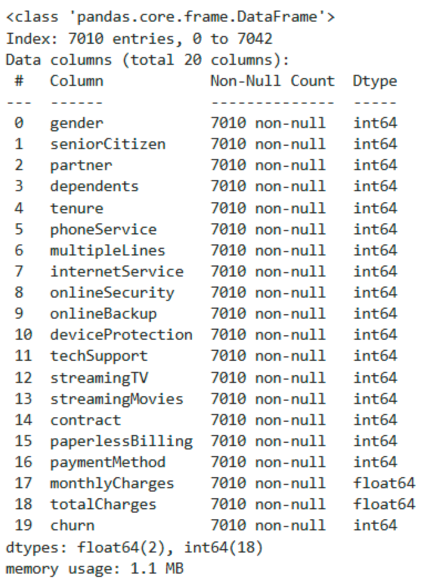


```python
# Normalize all string/object columns: lowercase and remove extra spaces
for col in churn_data.select_dtypes(include='object').columns:
    churn_data[col] = churn_data[col].str.strip().str.lower()
```

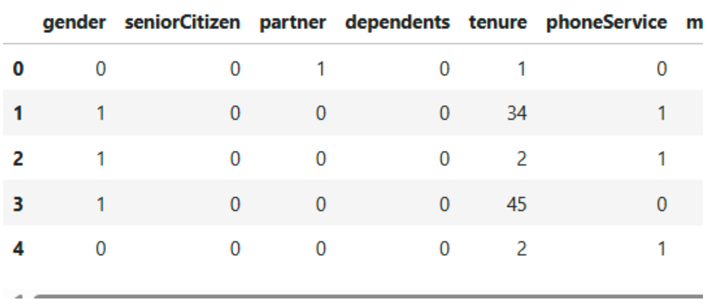
```python
# Define a binary map for Yes/No responses
binary_map = {'yes': 1, 'no': 0}
# Map binary and gender values to numerical format
churn_data['gender'] = churn_data['gender'].map({'female': 0, 'male': 1})
churn_data['partner'] = churn_data['partner'].map(binary_map)
churn_data['dependents'] = churn_data['dependents'].map(binary_map)
churn_data['phoneService'] = churn_data['phoneService'].map(binary_map)
churn_data['paperlessBilling'] = churn_data['paperlessBilling'].map(binary_map)
churn_data['churn'] = churn_data['churn'].map(binary_map)

# Encode categorical variables with multiple categories
churn_data['multipleLines'] = churn_data['multipleLines'].map({
    'no': 0,
    'yes': 1,
    'no phone service': 2
})

churn_data['internetService'] = churn_data['internetService'].map({
    'dsl': 0,
    'fiber optic': 1,
    'no': 2
})

churn_data['onlineSecurity'] = churn_data['onlineSecurity'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})

churn_data['onlineBackup'] = churn_data['onlineBackup'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})
```



```python
churn_data['deviceProtection'] = churn_data['deviceProtection'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})

churn_data['techSupport'] = churn_data['techSupport'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})

churn_data['streamingTV'] = churn_data['streamingTV'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})

churn_data['streamingMovies'] = churn_data['streamingMovies'].map({
    'no': 0,
    'yes': 1,
    'no internet service': 2
})

churn_data['contract'] = churn_data['contract'].map({
    'month-to-month': 0,
    'one year': 1,
    'two year': 2
})

churn_data['paymentMethod'] = churn_data['paymentMethod'].map({
    'electronic check': 0,
    'mailed check': 1,
    'bank transfer (automatic)': 2,
    'credit card (automatic)': 3
})
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 7010 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7010 non-null   int64  
 1   seniorCitizen     7010 non-null   int64  
 2   partner           7010 non-null   int64  
 3   dependents        7010 non-null   int64  
 4   tenure            7010 non-null   int64  
 5   phoneService      7010 non-null   int64  
 6   multipleLines     7010 non-null   int64  
 7   internetService   7010 non-null   int64  
 8   onlineSecurity    7010 non-null   int64  
 9   onlineBackup      7010 non-null   int64  
 10  deviceProtection  7010 non-null   int64  
 11  techSupport       7010 non-null   int64  
 12  streamingTV       7010 non-null   int64  
 13  streamingMovies   7010 non-null   int64  
 14  contract          7010 non-null   int64  
 15  paperlessBilling  7010 non-null   int64  
 16  paymentMethod     7010 non-null   int64  
 17  monthlyCharges    7010 non-null   float64
 18  totalCharges      7010 non-null   float64
 19  churn             7010 non-null   int64  
dtypes: float64(2), int64(18)
memory usage: 1.1 MB
```



	gender	seniorCitizen	partner	dependents	tenure	phoneService	m
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	

EXPLORATORY DATA ANALYSIS (EDA)

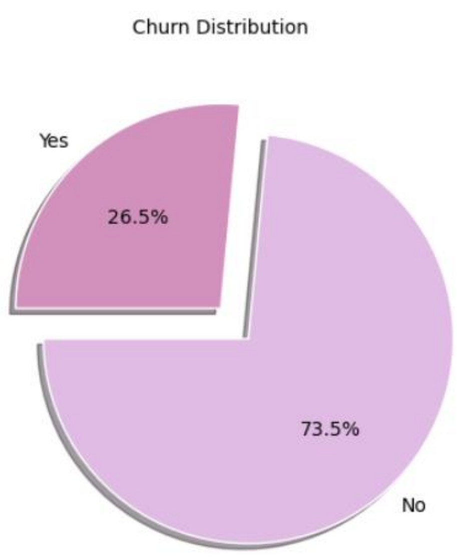
To better understand the structure, distribution, and relationships within the dataset, we conducted Exploratory Data Analysis (EDA). Using a variety of visual tools—such as pie charts, bar plots, histograms, and KDE plots—we explored customer behavior patterns, service usage, contract types, and how these factors relate to churn. This step provided valuable insights into which features may influence a customer's decision to leave the service.

➢ CHURN DISTRIBUTION

Let's start by looking at the distribution of the churn variable itself. The churn column indicates whether a customer has left the company (Yes) or stayed (No). From both the value counts and the pie chart, we can see that approximately 26.5% of customers have churned while 73.5% have stayed with the company. This indicates a class imbalance in the target variable, which is a common challenge in churn prediction problems. Identifying and addressing this imbalance is essential for building effective predictive models.

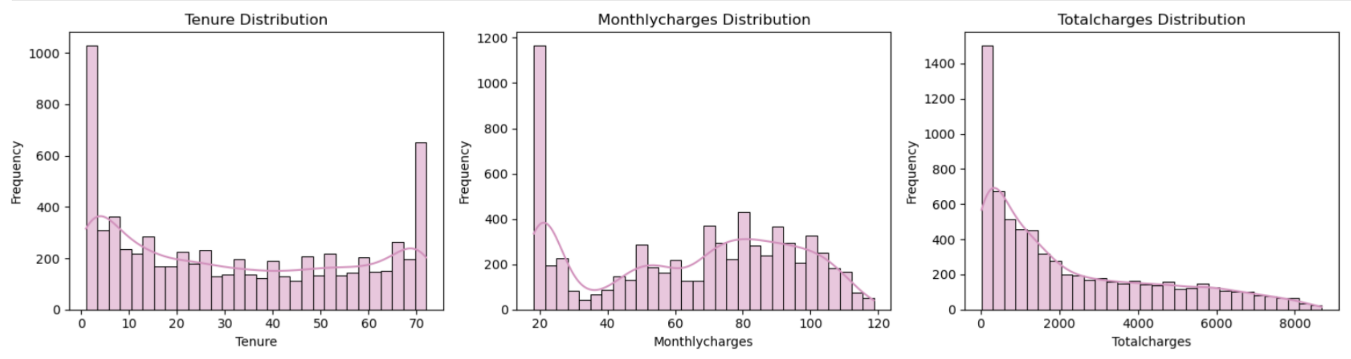
```
# Distribution of churn value
churn_data['churn'].value_counts()
```

```
churn
No     5153
Yes    1857
Name: count, dtype: int64
```



➢ NUMERICAL FEATURES INSIGHTS

```
# Numerical Features Distribution
cols = ["tenure", "monthlyCharges", "totalCharges"]
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

for i, col in enumerate(cols):
    sns.histplot(churn_data[col], bins=30, kde=True, color="#D291BC", ax=axes[i])
    axes[i].set_title(f"{col.capitalize()} Distribution")
    axes[i].set_xlabel(col.capitalize())
    axes[i].set_ylabel("Frequency")

plt.tight_layout()
plt.show()
```

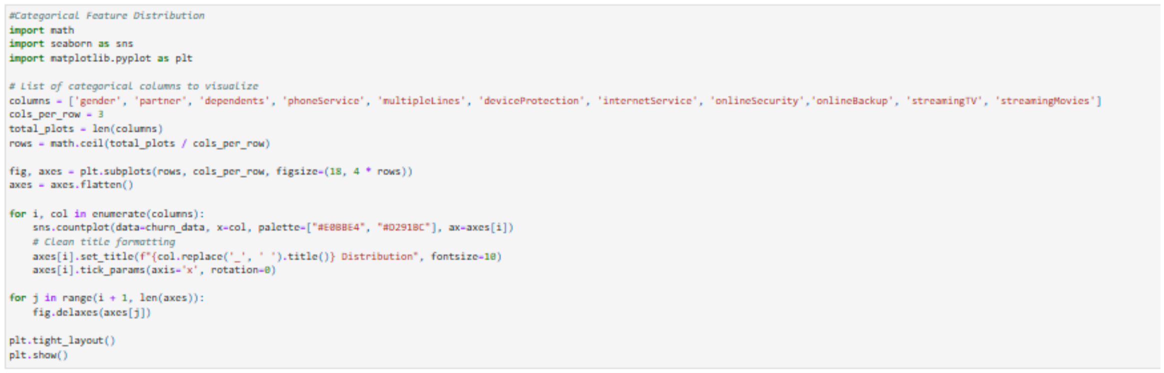


DATA DAZZELERS

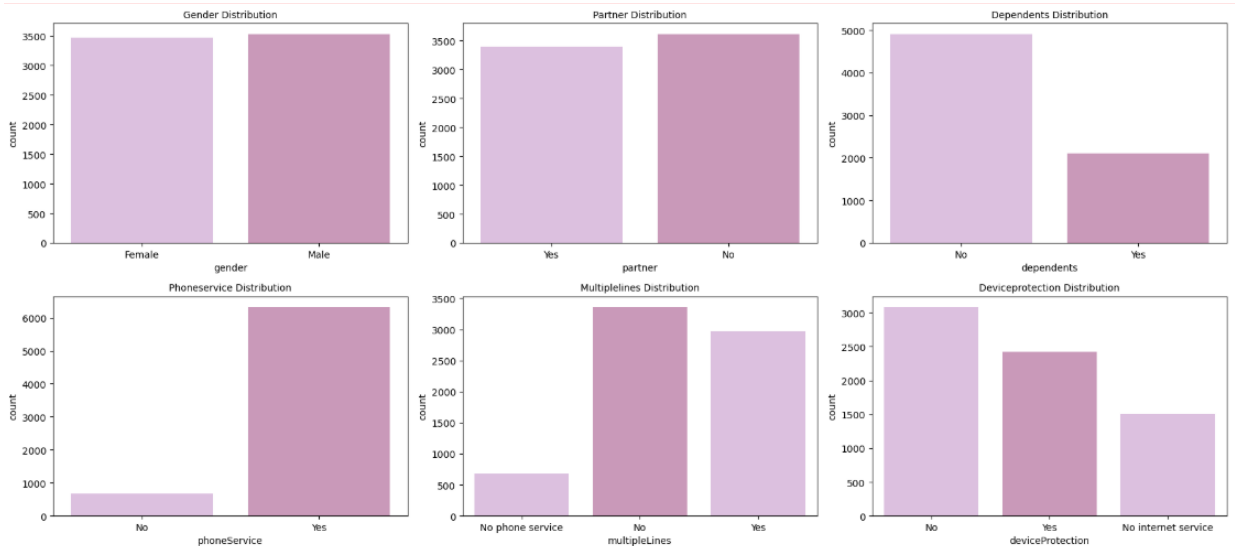
1. **Tenure Distribution**:
   - The plot shows a large number of customers with very short tenure (less than 5 months), which could indicate that a significant portion of users leave early.
   - There's also a noticeable number of long-term customers (around 70 months), reflecting loyal users.
   - This **bimodal distribution** suggests that customers tend to either churn early or stay long-term, with fewer customers in the mid-tenure range.
2. **Monthly Charges Distribution**:
   - Most customers pay a monthly charge between **20 and 100**.
   - There's a visible **peak around the lower end (~20-30)**, suggesting many customers are on low-cost plans.
   - The spread of charges suggests a variety of pricing plans, which could be linked to the number of services subscribed.
3. **Total Charges Distribution**:
   - The majority of customers have low total charges, which aligns with the tenure distribution – newer customers haven't accumulated high charges yet.
   - As tenure increases, total charges logically increase too, and this distribution is **right-skewed**.
   - The long tail indicates a few high-value long-term customers who contribute significantly to the company's revenue.

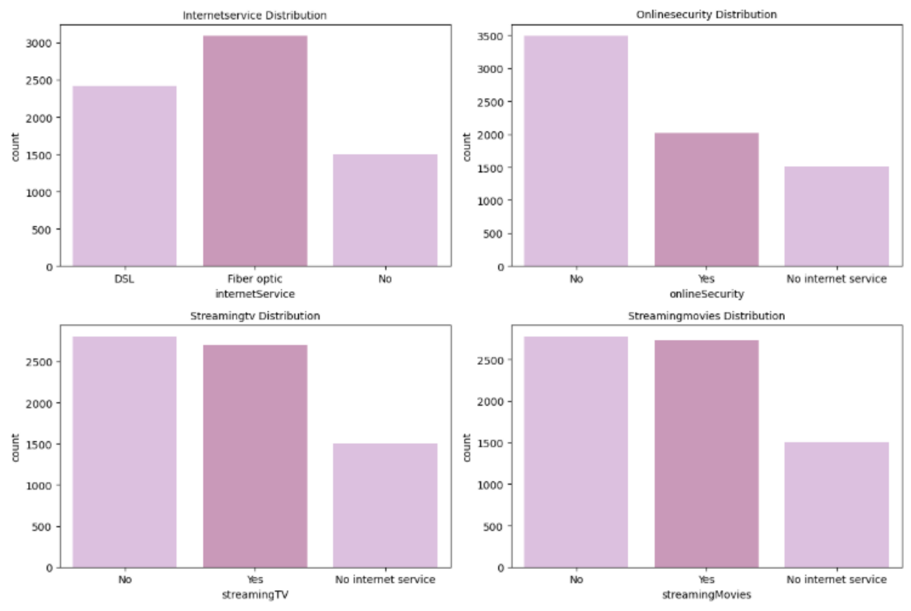
These insights help in identifying customer behaviors that lead to churn — especially among **short-tenure, low-total-charge users**, who may need better onboarding or service experiences to be retained.

➢ CATEGORICAL FEATURE ANALYSIS



```python
#Categorical Feature Distribution
import math
import seaborn as sns
import matplotlib.pyplot as plt

# List of categorical columns to visualize
columns = ['gender', 'partner', 'dependents', 'phoneService', 'multipleLines', 'deviceProtection', 'internetService', 'onlineSecurity','onlineBackup', 'streamingTV', 'streamingMovies']
cols_per_row = 3
total_plots = len(columns)
rows = math.ceil(total_plots / cols_per_row)

fig, axes = plt.subplots(rows, cols_per_row, figsize=(18, 4 * rows))
axes = axes.flatten()

for i, col in enumerate(columns):
    sns.countplot(data=churn_data, x=col, palette=["#E0BBE4", "#D291BC"], ax=axes[i])
    # Clean title formatting
    axes[i].set_title(f"{col.replace('_', ' ').title()} Distribution", fontsize=10)
    axes[i].tick_params(axis='x', rotation=0)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```





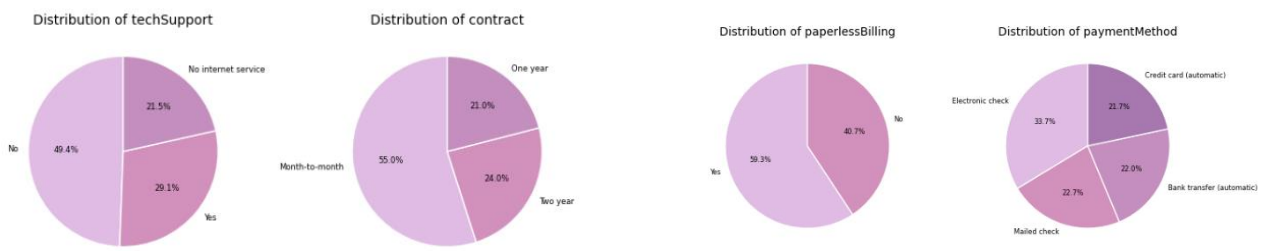
To gain a deeper understanding of customer behavior, we analyzed the distribution of several categorical variables in the dataset using count plots. These visualizations help us explore the frequency of various service features and customer attributes across the entire dataset.

Key Observations:

- **Gender, Partner, and Phone Service:** The distribution of male and female customers is nearly equal. A slightly larger portion of customers do not have a partner. Most customers have phone service enabled.
- **Dependents and Multiple Lines:** A significant number of customers do not have dependents. Among those with phone service, multiple lines are fairly common.
- **Internet-Related Services:**
    - The majority of customers have **Fiber Optic** or **DSL** internet, while a smaller group reported no internet service.
    - Services such as **online security, backup, device protection, and tech support** were more often not subscribed to, especially among fiber optic users.
- **Streaming Services:** Both streaming TV and movie services were almost evenly split between 'Yes' and 'No', with a notable percentage of customers not using these services due to lack of internet service.

These categorical insights provide clues about service preferences and help identify patterns that may be linked to customer churn. For instance, lack of value-added services like security or tech support may signal dissatisfaction or lower engagement levels.

➢ PIE CHART DISTRIBUTION OF SERVICE AND BILLING-RELATED FEATURES

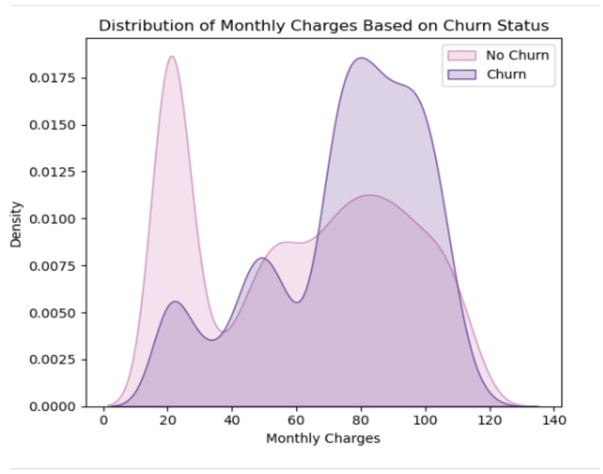


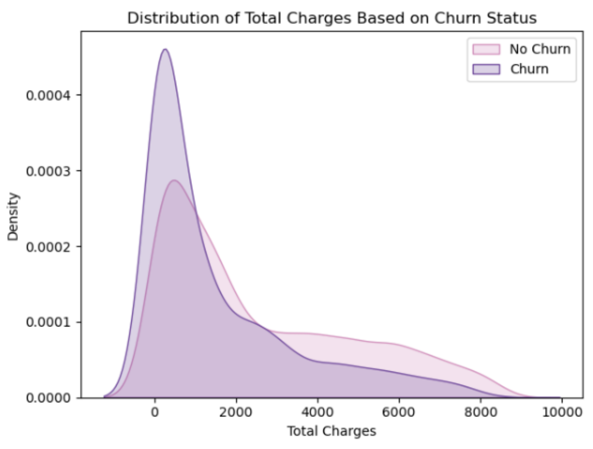
To better understand customer preferences, pie charts were used to visualize key service and billing-related features such as Tech Support, Contract, Paperless Billing, and Payment Method.

Key Observations:

- Most customers are on month-to-month contracts, making them more likely to churn compared to long-term users.
- A majority use paperless billing, showing a shift toward digital services.
- Tech support usage is low, with many customers either not using it or not having internet service.
- The most common payment method is electronic check, which may be associated with higher churn in some cases.
- These visuals helped identify patterns and preferences in how customers interact with the company's services.

➢ MONTHLY & TOTAL CHARGES VS CHURN





We compared monthly and total charges between churned and retained customers using KDE plots

Monthly Charges:

- Customers who churned are more likely to have higher monthly charges.
- The distribution of churned customers shows a noticeable peak between 70 and 100, while non-churned customers peak at lower monthly charges (around 20–30).
- This indicates that higher monthly bills might contribute to customer dissatisfaction and churn.

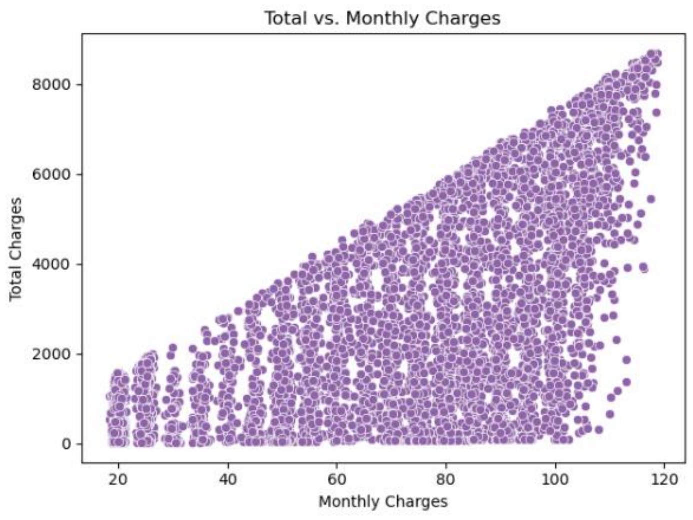
Total Charges:

- Customers who did not churn tend to have higher total charges, implying they've stayed with the company longer.

- Churned customers are more frequent at the lower end of total charges, suggesting that many left the service relatively early in their customer journey.

These insights support the idea that **new customers and customers with higher bills** are more at risk of leaving. This kind of pattern can guide telecom companies to focus on new subscribers and those with expensive plans for improved retention efforts.

➢ RELATIONSHIP BETWEEN MONTHLY CHARGES AND TOTAL CHARGES



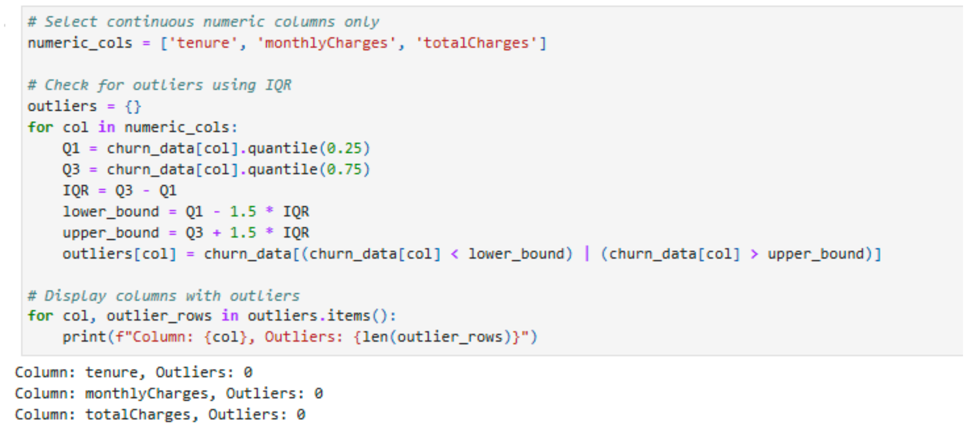
There is a **clear positive correlation** between monthly charges and total charges. As monthly charges increase, total charges also tend to increase.

The data points form a triangular pattern:

- **New customers** with **high monthly charges** appear on the lower right (high monthly, low total).
- **Long-term customers** with **high total charges** appear toward the upper side.

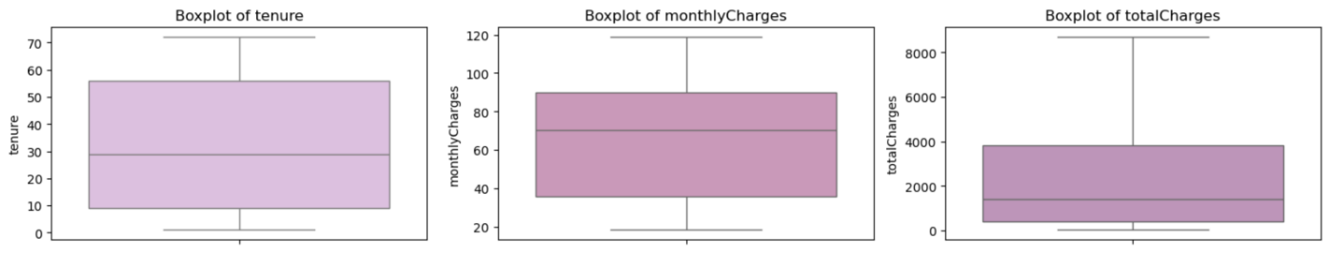
The plot also highlights variability — customers paying the same monthly amount may have very different total charges, depending on their **tenure** with the company.

➢ OUTLIER DETECTION



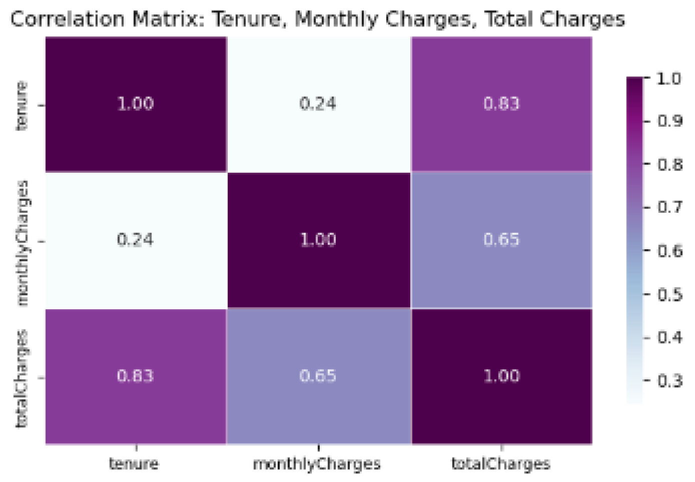
```
# Select continuous numeric columns only
numeric_cols = ['tenure', 'monthlyCharges', 'totalCharges']

# Check for outliers using IQR
outliers = {}
for col in numeric_cols:
    Q1 = churn_data[col].quantile(0.25)
    Q3 = churn_data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers[col] = churn_data[(churn_data[col] < lower_bound) | (churn_data[col] > upper_bound)]

# Display columns with outliers
for col, outlier_rows in outliers.items():
    print(f"Column: {col}, Outliers: {len(outlier_rows)}")
```

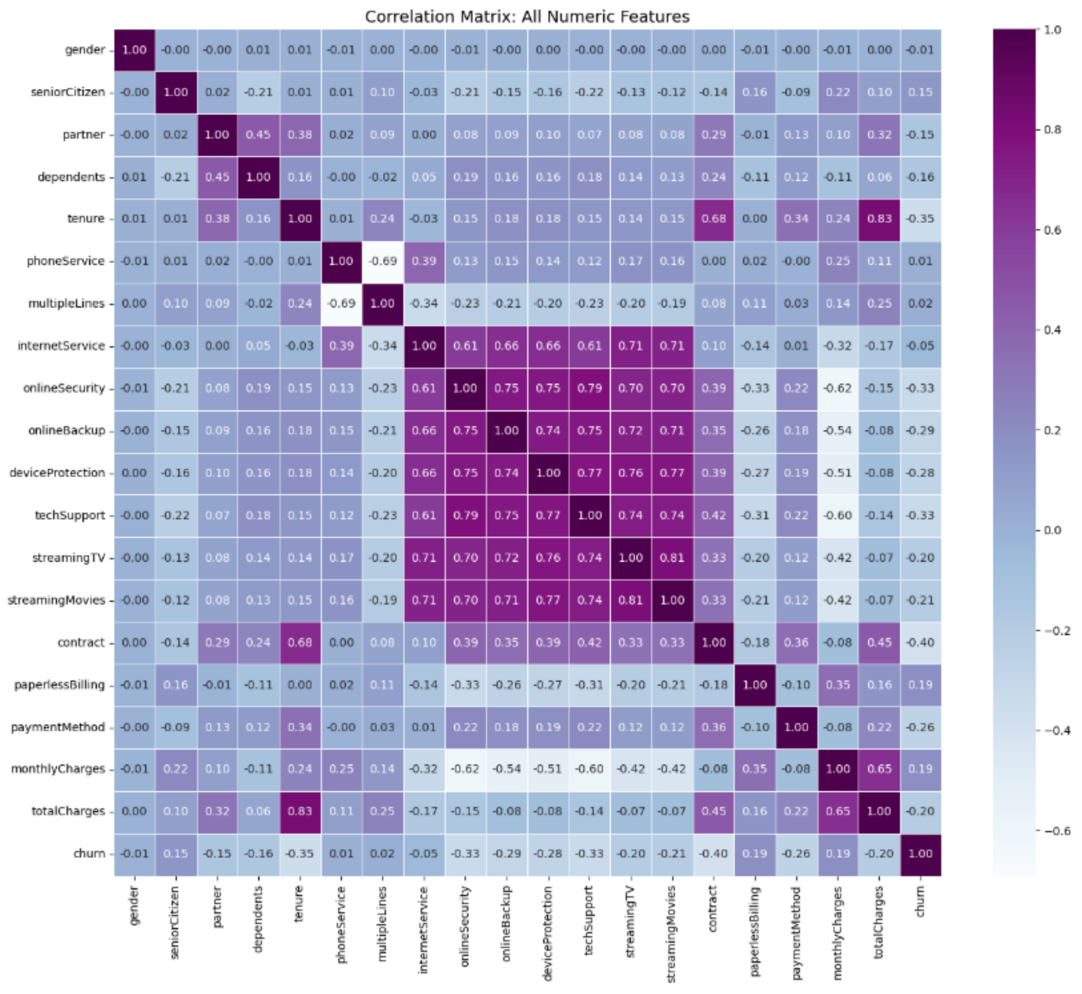
```
Column: tenure, Outliers: 0
Column: monthlyCharges, Outliers: 0
Column: totalCharges, Outliers: 0
```



Outliers were examined for the key continuous features: **tenure**, **monthly charges**, and **total charges** using the Interquartile Range (IQR) method. The goal was to identify extreme values that could distort model performance. However, the analysis revealed no significant outliers in these columns. This indicates that the numerical data is well-behaved and evenly distributed, allowing us to proceed with modeling without needing to remove or adjust any data points.

➢ CORRELATION MATRIX





To understand how different features relate to one another, we examined the correlation matrix. A strong positive correlation was observed between tenure and totalCharges, which is expected as longer customer tenure usually results in higher total charges. monthlyCharges also showed a moderate correlation with totalCharges.

When analyzing the full correlation matrix, we found that the churn variable had a weak negative correlation with tenure and a slightly stronger positive correlation with monthlyCharges, suggesting that newer customers with higher monthly bills are more likely to churn. These insights help us identify which variables may be useful in predicting churn.

Correlation Matrix: Tenure, Monthly Charges, Total Charges

- **Tenure and Total Charges** show a **strong positive correlation (0.83)**, indicating that customers who have stayed longer tend to accumulate higher total charges.
- **Monthly Charges and Total Charges** also have a **moderate positive correlation (0.65)**, which makes sense as higher monthly fees contribute to total spending.
- However, **tenure and monthly charges** show a **weak correlation (0.24)**, suggesting that how long a customer stays is not strongly related to how much they pay monthly.

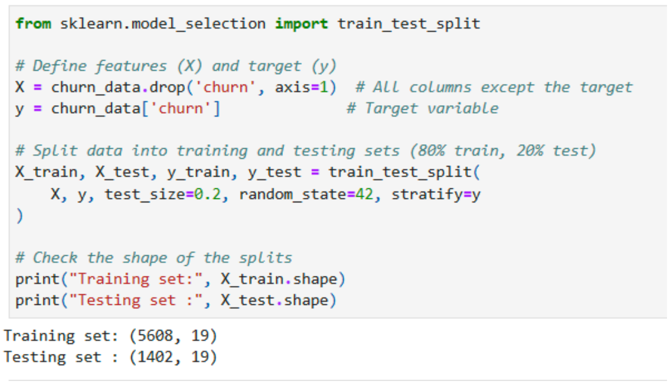
Correlation Matrix: All Numeric Features

- **Churn** has a **negative correlation with tenure (-0.35)**, meaning customers who have been with the company for a shorter time are more likely to churn.
- It also has **positive correlation with monthly charges (0.19)**, showing that those paying more each month are slightly more likely to leave.
- Features like **contract type, online security, tech support, and tenure** show meaningful relationships with churn and are important candidates for model training.
- Most other features exhibit low or near-zero correlation with churn, but that doesn’t mean they’re irrelevant—non-linear models may still capture their influence.

Key Takeaways:

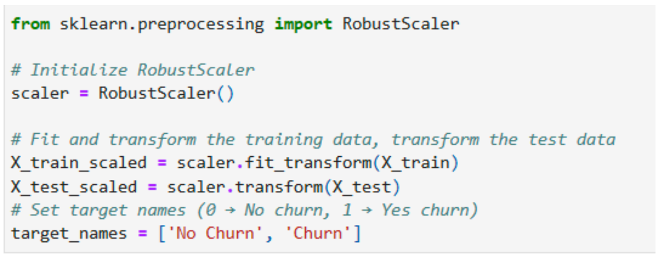
- Long-tenured customers tend to churn less.
- Higher monthly charges may contribute to increased churn risk.
- Service-related features (like contract and security options) are moderately correlated with churn and may help improve prediction accuracy.

DATA SPLITTING AND MODEL TRAINING



```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = churn_data.drop('churn', axis=1)  # All columns except the target
y = churn_data['churn']              # Target variable

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Check the shape of the splits
print("Training set:", X_train.shape)
print("Testing set :", X_test.shape)
```

```
Training set: (5608, 19)
Testing set : (1402, 19)
```



```
from sklearn.preprocessing import RobustScaler

# Initialize RobustScaler
scaler = RobustScaler()

# Fit and transform the training data, transform the test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Set target names (0 → No churn, 1 → Yes churn)
target_names = ['No Churn', 'Churn']
```

Data Preparation for Machine Learning

In this step, we prepare the data for building machine learning models by applying a series of preprocessing techniques. These steps help ensure that the models can learn effectively and perform well on unseen data.

1. **Feature Selection**:
   The independent variables (features) are selected by dropping the target variable (churn) from the dataset. The target variable churn is stored separately as **y**, representing whether a customer has left or stayed.
2. **Train-Test Split**:
   The dataset is divided into **training (80%)** and **testing (20%)** subsets using train_test_split from Scikit-learn. Stratification is used to preserve the original distribution of churn labels in both sets. This helps in building a model that generalizes well to new, unseen data.
3. **Data Scaling**:
   To bring all numerical features onto a similar scale and reduce the effect of outliers, **RobustScaler** is applied. Unlike standard scaling, RobustScaler uses the median and interquartile range, making it especially useful for data that may contain outliers.
4. **Target Class Labels**:
   The target class (churn) is binary and includes two classes:
   - **0** for “No” (Customer did not churn)
   - **1** for “Yes” (Customer churned)
     These labels are used during evaluation for calculating metrics such as precision, recall, and F1-score.
5. **Training and Testing Sizes**:
   - **Training set:** 5608 records
   - **Testing set:** 1402 records
     This ensures enough data for both training the model and validating its performance.

MACHINE LEARNING MODEL IMPLEMENTATION

In this project, we perform a binary classification task to predict whether a customer will churn (leave) or stay with the telecom company. The target variable churn has two classes:

0 — Customer did not churn

1 — Customer churned

To achieve this, we implemented and evaluated several machine learning models. Each model was trained on historical customer data and tested using performance metrics like accuracy, precision, recall, and F1-score to assess how well it predicted churn.

The models we used are as follows:

Logistic Regression: A simple and interpretable linear model used to estimate the probability of churn. It maps input features to a value between 0 and 1 using the sigmoid function, making it effective for binary classification.

Random Forest: An ensemble model that builds multiple decision trees and combines their outputs. It improves accuracy and helps reduce overfitting. Random Forest is robust to noise and handles both numerical and categorical features well.

Gradient Boosting: A powerful boosting algorithm that builds models sequentially. Each new tree corrects the errors made by the previous ones. It often performs well on imbalanced datasets and captures complex patterns in the data.

SMOTE for Imbalanced Data: To address the class imbalance (more "No Churn" than "Yes Churn" cases), we experimented with SMOTE (Synthetic Minority Over-sampling Technique). This helped balance the dataset and improved recall for the minority class in some models.

Model Evaluation

We assessed all models using the following metrics:

Accuracy — How often the model is correct

Precision — Of the customers predicted to churn, how many actually did

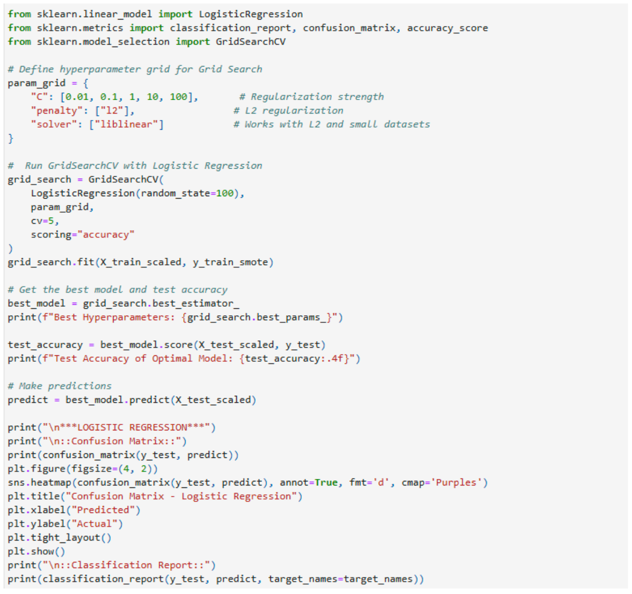
Recall — Of all actual churners, how many were correctly identified

F1-score — A balance between precision and recall

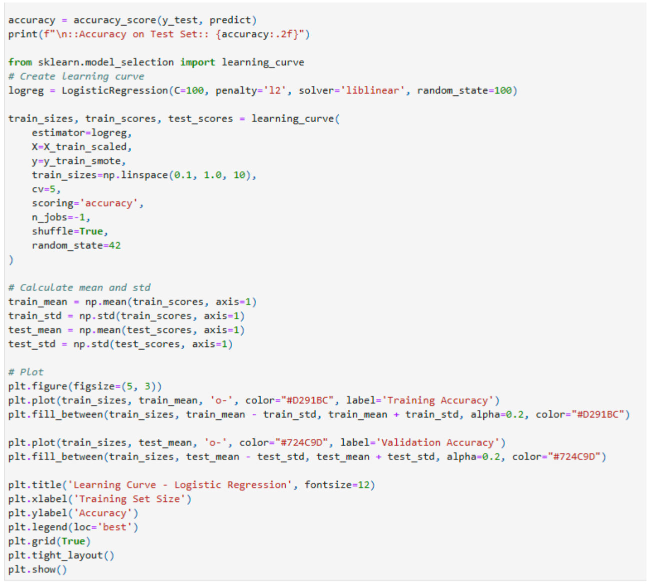
These metrics helped us compare the effectiveness of each algorithm and determine which model was most suitable for predicting customer churn in this dataset.

MODELS- IMPROVED AND OPTIMISED

LOGISTIC REGRESSION

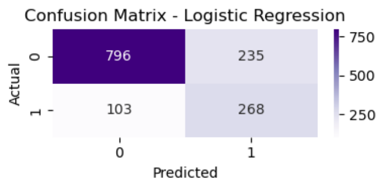


```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid for Grid Search
param_grid = {
    "C": [0.01, 0.1, 1, 10, 100],       # Regularization strength
    "penalty": ["l2"],                 # L2 regularization
    "solver": ["liblinear"]            # Works with L2 and small datasets
}

#  Run GridSearchCV with Logistic Regression
grid_search = GridSearchCV(
    LogisticRegression(random_state=100),
    param_grid,
    cv=5,
    scoring="accuracy"
)
grid_search.fit(X_train_scaled, y_train_smote)

# Get the best model and test accuracy
best_model = grid_search.best_estimator_
print(f"Best Hyperparameters: {grid_search.best_params_}")

test_accuracy = best_model.score(X_test_scaled, y_test)
print(f"Test Accuracy of Optimal Model: {test_accuracy:.4f}")

# Make predictions
predict = best_model.predict(X_test_scaled)

print("\n***LOGISTIC REGRESSION***")
print("\n::Confusion Matrix::")
print(confusion_matrix(y_test, predict))
plt.figure(figsize=(4, 2))
sns.heatmap(confusion_matrix(y_test, predict), annot=True, fmt='d', cmap='Purples')
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
print("\n::Classification Report::")
print(classification_report(y_test, predict, target_names=target_names))
```

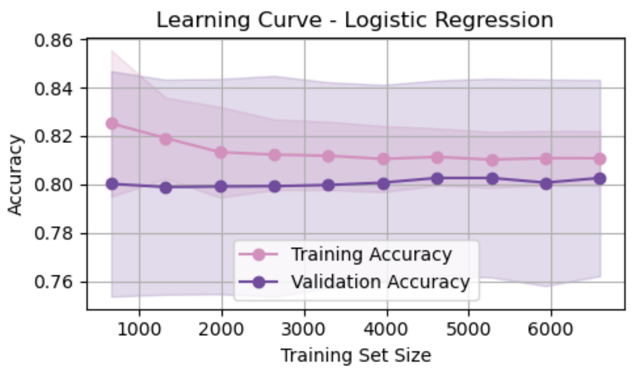


```python
accuracy = accuracy_score(y_test, predict)
print(f"\n::Accuracy on Test Set:: {accuracy:.2f}")

from sklearn.model_selection import learning_curve
# Create learning curve
logreg = LogisticRegression(C=100, penalty='l2', solver='liblinear', random_state=100)

train_sizes, train_scores, test_scores = learning_curve(
    estimator=logreg,
    X=X_train_scaled,
    y=y_train_smote,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    shuffle=True,
    random_state=42
)

# Calculate mean and std
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot
plt.figure(figsize=(5, 3))
plt.plot(train_sizes, train_mean, 'o-', color="#D291BC", label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color="#D291BC")

plt.plot(train_sizes, test_mean, 'o-', color="#724C9D", label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2, color="#724C9D")

plt.title('Learning Curve - Logistic Regression', fontsize=12)
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Best Hyperparameters: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
Test Accuracy of Optimal Model: 0.7589

***LOGISTIC REGRESSION***

::Confusion Matrix::
[[796 235]
 [103 268]]
```



```
::Classification Report::
              precision    recall  f1-score   support

    No Churn       0.89      0.77      0.82      1031
       Churn       0.53      0.72      0.61       371

    accuracy                           0.76      1402
   macro avg       0.71      0.75      0.72      1402
weighted avg       0.79      0.76      0.77      1402


::Accuracy on Test Set:: 0.76
```



Overview of the Model

Logistic Regression is a widely used algorithm for binary classification problems. It models the probability that a given input belongs to a particular class using a sigmoid function. In this project, it was implemented to predict customer churn (Yes/No) based on customer behavior and service usage metrics. This model was selected due to its interpretability, scalability, and efficiency for linearly separable problems.

Hyperparameter Tuning

To enhance the model's performance, **GridSearchCV** was applied to find the optimal combination of hyperparameters:

- **C (Inverse of Regularization Strength)**: Controls the trade-off between bias and variance. A higher value (C=100) reduced regularization and allowed more model flexibility.
- **Penalty**: L2 regularization was used to prevent overfitting and stabilize the coefficient values.

- **Solver**: liblinear was chosen for its effectiveness with small to medium-sized datasets and support for L2 penalty.
- **Cross-Validation**: A 5-fold cross-validation strategy was used during grid search to ensure stability and generalizability.

Optimal Hyperparameters:

- C = 100
- Penalty = 'l2'
- Solver = 'liblinear'

These parameters provided the best balance of accuracy and generalization.

Performance Metrics

The model's performance was evaluated using **accuracy**, **precision**, **recall**, **F1-score**, and a **confusion matrix** to interpret predictions for the *Churn* and *No Churn* customer classes.

1. Confusion Matrix Analysis:

- **True Negatives (796)**: Correctly identified customers who did not churn.
- **False Positives (235)**: Incorrectly classified non-churners as churners, potentially leading to unnecessary retention efforts.
- **False Negatives (103)**: Churners misclassified as non-churners, leading to missed intervention opportunities.
- **True Positives (268)**: Correctly identified customers who churned.

This balance between **True Positives** and **False Negatives** highlights the model's ability to detect churners, while indicating room for improvement in reducing false alarms.

2. Precision:

- **No Churn**: Precision of **89%** reflects high confidence in identifying customers who will stay.
- **Churn**: Precision of **53%** suggests that nearly half of the churn predictions may be false positives.

Interpretation: Precision ensures that when the model predicts a customer will churn, it is correct most of the time — though this metric is weaker for churned customers.

3. Recall:

- **No Churn**: Recall of **77%** confirms the model's ability to correctly retain most loyal customers.
- **Churn**: Recall of **72%** indicates strong sensitivity in identifying at-risk customers.

Interpretation: A high recall for churners is crucial from a business standpoint, as it ensures the majority of churn-prone customers are flagged for retention.

4. F1-Score:

- **No Churn**: F1-score of **82%** shows balanced precision and recall, confirming consistent classification.
- **Churn**: F1-score of **61%** reveals room for enhancement in reliably classifying churned customers.
- **Macro Average F1-Score**: **72%**
- **Weighted Average F1-Score**: **77%**

Interpretation: These scores reflect the model's balanced ability to handle class imbalance and its effectiveness across both customer categories.

5. Accuracy:

- **Overall Accuracy**: **76.0%**

Interpretation: The model accurately predicts churn status for most customers, offering a reliable baseline for identifying and targeting churn risks.

Key Insights

- The model achieved a **76% overall accuracy**, with strong performance for non-churn predictions but moderate results for actual churn cases.
- **Recall for churn (72%)** suggests the model identifies a large portion of customers likely to churn, which is vital for retention efforts.

Strengths

- **High Precision for No Churn (89%)**: The model is reliable in identifying loyal customers, minimizing unnecessary outreach.
- **Good Recall for Churn (72%)**: Indicates strong capability in flagging at-risk customers for proactive retention strategies.

Limitations

- **Lower Precision for Churn (53%)**: Nearly half of churn predictions could be false positives, potentially straining resources.
- **Moderate F1-Score for Churn (61%)**: Highlights a need for improvement in balanced detection of churners.

Business Implications

- **Retention Campaign Optimization**: High recall for churners enables targeted retention offers, reducing customer attrition.
- **Cost Control**: Strong identification of non-churners helps avoid unnecessary incentives or retention efforts, saving marketing costs.

Improvement Strategies – Logistic Regression

1. **Feature Engineering**
   - Create interaction features (e.g., tenure × monthlyCharges) to capture non-linear effects.
   - Apply logarithmic or polynomial transformations to skewed features for better linear separability.
2. **Class Imbalance Handling**
   - Continue using **SMOTE** or explore **ADASYN** or **ensemble methods** like balanced bagging to improve minority class prediction.
   - Alternatively, experiment with class_weight='balanced' in Logistic Regression to give higher weight to the churn class.
3. **Threshold Tuning**
   - Instead of using the default classification threshold of 0.5, adjust the threshold based on the ROC curve to optimize recall for the Churn class.
4. **Alternative Regularization Techniques**
   - Try **Elastic Net regularization**, which combines L1 and L2 penalties, useful for handling multicollinearity and improving generalization.
5. **Advanced Hyperparameter Tuning**
   - Move beyond GridSearchCV to **Bayesian Optimization** (e.g., Optuna or Hyperopt) for more efficient and precise tuning.
6. **Cross-Validation Strategy**
   - Use **Stratified K-Fold Cross-Validation** to maintain class proportions across all folds, ensuring a more balanced evaluation.
7. **Ensemble Logistic Approaches**
   - Combine Logistic Regression with bagging or boosting to reduce variance and improve performance on harder-to-classify samples.

Conclusion

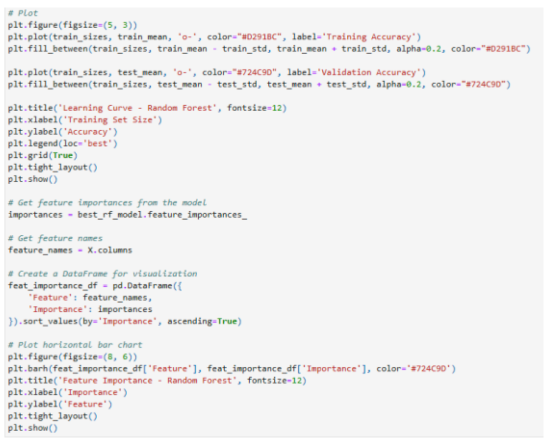
In conclusion, the Logistic Regression model demonstrated solid performance in predicting customer churn. While the **overall test accuracy reached 76%**, what stands out is the **balanced recall across both classes**, especially for the **churn class**, which is the primary focus of the business objective.

Initially, without applying SMOTE, the model achieved a slightly higher accuracy (~81%). However, this came at the cost of a significantly **lower recall for churn**, meaning many actual churners were being missed. After applying **SMOTE (Synthetic Minority Over-sampling Technique)**, although accuracy slightly dropped, **recall for the churn class improved to 72%**, which is far more valuable in this context.

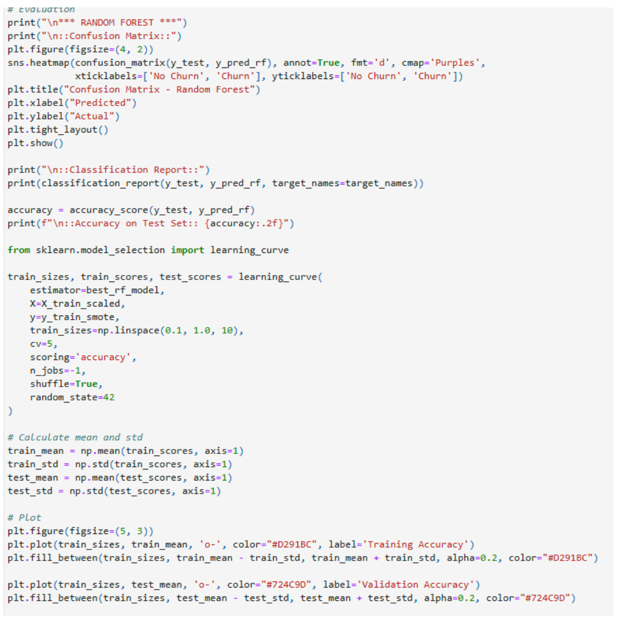
This trade-off is crucial: in churn prediction, **recall is more important than raw accuracy**, because it's better to catch more customers who are likely to leave (even at the risk of some false alarms) than to miss them entirely. **Higher recall means better customer retention strategies**, as it enables the company to take timely action to prevent churn and reduce revenue loss.

Thus, the optimized Logistic Regression model, aided by SMOTE and hyperparameter tuning, is not only balanced but also aligned with the business goal of proactive churn management.

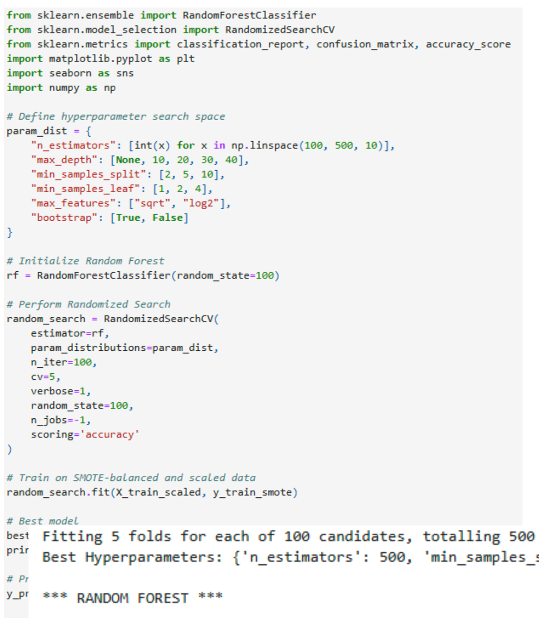
RANDOM FOREST



```python
# Plot
plt.figure(figsize=(5, 3))
plt.plot(train_sizes, train_mean, 'o-', color="#D291BC", label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color="#D291BC")

plt.plot(train_sizes, test_mean, 'o-', color="#724C9D", label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2, color="#724C9D")

plt.title('Learning Curve - Random Forest', fontsize=12)
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()

# Get feature importances from the model
importances = best_rf_model.feature_importances_

# Get feature names
feature_names = X.columns

# Create a DataFrame for visualization
feat_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=True)

# Plot horizontal bar chart
plt.figure(figsize=(8, 6))
plt.barh(feat_importance_df['Feature'], feat_importance_df['Importance'], color='#724C9D')
plt.title('Feature Importance - Random Forest', fontsize=12)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

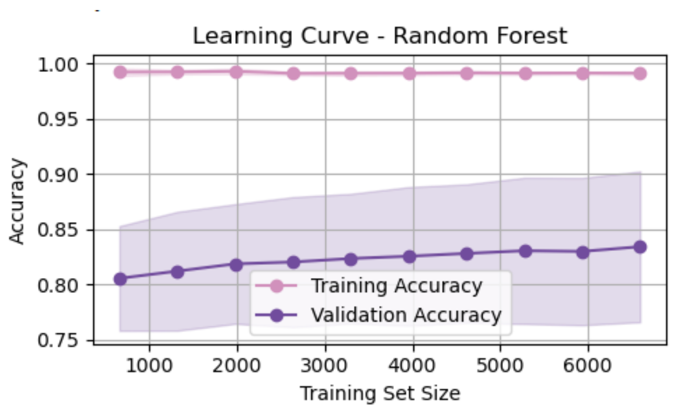


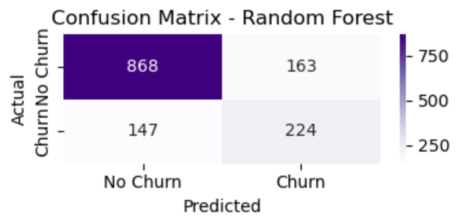
```python
# Evaluation
print("\n*** RANDOM FOREST ***")
print("\n::Confusion Matrix::")
plt.figure(figsize=(4, 2))
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Purples',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

print("\n::Classification Report::")
print(classification_report(y_test, y_pred_rf, target_names=target_names))

accuracy = accuracy_score(y_test, y_pred_rf)
print(f"\n::Accuracy on Test Set:: {accuracy:.2f}")

from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    estimator=best_rf_model,
    X=X_train_scaled,
    y=y_train_smote,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    shuffle=True,
    random_state=42
)

# Calculate mean and std
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot
plt.figure(figsize=(5, 3))
plt.plot(train_sizes, train_mean, 'o-', color="#D291BC", label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color="#D291BC")

plt.plot(train_sizes, test_mean, 'o-', color="#724C9D", label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2, color="#724C9D")
```



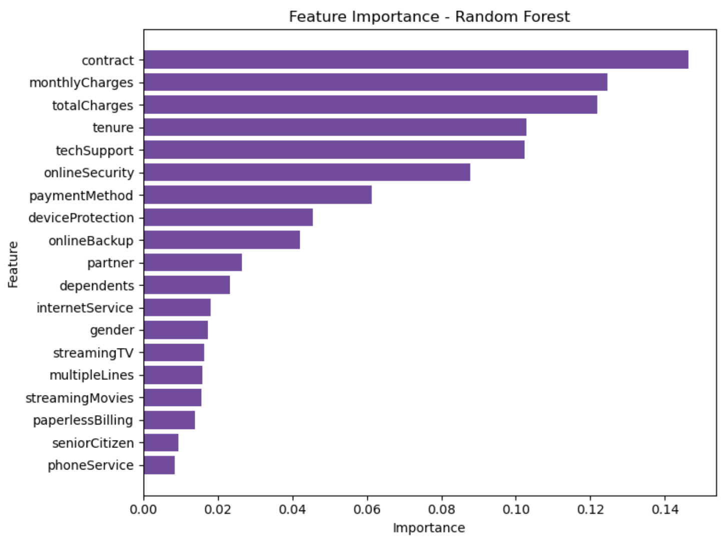
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Define hyperparameter search space
param_dist = {
    "n_estimators": [int(x) for x in np.linspace(100, 500, 10)],
    "max_depth": [None, 10, 20, 30, 40],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2"],
    "bootstrap": [True, False]
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=100)

# Perform Randomized Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=100,
    cv=5,
    verbose=1,
    random_state=100,
    n_jobs=-1,
    scoring='accuracy'
)

# Train on SMOTE-balanced and scaled data
random_search.fit(X_train_scaled, y_train_smote)

# Best model
best
prir

# Pr
y_pr
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Hyperparameters: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': None, 'bootstrap': False}

*** RANDOM FOREST ***

::Confusion Matrix::
```





```
::Classification Report::
              precision    recall  f1-score   support

    No Churn       0.86      0.84      0.85      1031
       Churn       0.58      0.60      0.59       371

    accuracy                           0.78      1402
   macro avg       0.72      0.72      0.72      1402
weighted avg       0.78      0.78      0.78      1402
```



Overview of the Model

Random Forest is an ensemble learning method that builds multiple decision trees and combines their outputs to produce more accurate and stable predictions. It is particularly effective in handling non-linear relationships and complex datasets. In this project, Random Forest was used to predict customer churn based on a range of customer behavior and service-related features.

This model was selected for its:

- **Robustness to noise and outliers**
- **High performance on imbalanced datasets when paired with SMOTE**
- **Capability to provide feature importance, offering interpretability despite being an ensemble model**

After applying **SMOTE** to balance the target classes, Random Forest was trained with hyperparameter optimization using **RandomizedSearchCV**, making it well-tuned for the task of churn prediction.

Performance Metrics – Random Forest
The Random Forest model's performance was evaluated using accuracy, precision, recall, F1-score, and the confusion matrix. These metrics help assess the model's ability to distinguish between churned and non-churned customers.

1. Confusion Matrix Analysis

- **True Negatives (868):** Correctly predicted non-churned customers.
- **False Positives (163):** Incorrectly predicted churn for non-churned customers.
- **False Negatives (147):** Missed predicting churn for actual churned customers.
- **True Positives (224):** Correctly predicted churned customers.

While the model performs well in detecting "No Churn" cases, there's room to improve the recall for "Churn" cases.

2. Precision

- **No Churn:** 0.86 – High precision shows strong confidence when predicting customers who will stay.
- **Churn:** 0.58 – The model has moderate confidence in correctly identifying churners.

Interpretation:
The model is more confident and accurate when identifying loyal customers than it is when flagging churners.

3. Recall

- **No Churn:** 0.84 – The model successfully captures most of the non-churned customers.
- **Churn:** 0.60 – Slightly better than Logistic Regression, but still misses ~40% of churners.

Interpretation:
While slightly better than the Logistic Regression model, the recall for churn still shows a gap, meaning some at-risk customers remain undetected.

4. F1-Score

- **No Churn:** 0.85 – Balanced precision and recall.
- **Churn:** 0.59 – Indicates moderate performance on this minority class.
- **Macro Average F1-Score:** 0.72
- **Weighted Average F1-Score:** 0.78

Interpretation:
The weighted average suggests good overall performance, but macro average reveals imbalance in the model's ability to handle both classes equally.

5. Accuracy

- **Overall Accuracy:** 0.78

Interpretation:
The model correctly classifies 78% of the test samples. This is a solid performance but does not fully reflect the model's effectiveness in handling the minority class (churn), especially in imbalanced datasets.

Key Insights – Random Forest Model

1. **Better Overall Accuracy than Logistic Regression**
   - The model achieved **78% accuracy**, slightly higher than the Logistic Regression model (76%). This suggests that the ensemble method helps reduce variance and improves overall classification reliability.
2. **Balanced Recall for Both Classes**
   - With a **recall of 0.84 for No Churn** and **0.60 for Churn**, Random Forest demonstrates a better balance in identifying both churned and retained customers, which is essential in churn prediction use cases where both types of errors have consequences.
3. **Improved Detection of Churners Compared to Logistic Regression**
   - The **F1-score for the churn class improved to 0.59**, compared to 0.61 in Logistic Regression. While not a drastic change, the model still shows slightly improved generalization without compromising much on the No Churn class.
4. **Slight Overfitting Indicated in Learning Curve**
   - The learning curve reveals a **gap between training and validation accuracy**, which could hint at overfitting. The model fits the training data very well (near 100%) but doesn't generalize perfectly to unseen data.

Strengths

1. **High Overall Accuracy and Balanced Performance**
   - The model achieved a **test accuracy of 78%** with relatively good precision and recall for both churn and no churn classes, making it a strong performer in predicting overall customer behavior.
2. **Handles Non-linear Relationships Well**
   - Random Forest effectively captures complex interactions between variables, making it suitable for datasets where customer churn is influenced by multiple interconnected factors.

Limitations

1. **Moderate Recall for Churn Class (0.60)**
   - Although better than some models, the model still misses 40% of actual churners, which could mean lost opportunities to proactively retain those customers.
2. **Interpretability is Lower**
   - Unlike Logistic Regression, Random Forest is a black-box model, making it harder to interpret individual decision paths or understand the weight of each variable in decision-making.

Business Implications

1. **Better Customer Retention Strategies**
   - With improved identification of churn-prone customers, the business can take **proactive retention actions** such as personalized offers or targeted engagement campaigns.
2. **Supports Scalable Automation**

    - The robust and high-accuracy nature of the Random Forest model makes it a good fit for integrating into **automated churn monitoring systems** for real-time customer tracking and action.

Improvement Strategies – Random Forest

To further enhance the performance, robustness, and interpretability of the Random Forest model, the following strategies can be adopted:

1. Advanced Hyperparameter Tuning

- **Use Bayesian Optimization or GridSearch with finer granularity** for parameters like n_estimators, max_depth, and min_samples_leaf to find more optimal configurations.
- Increase the number of trees (n_estimators) cautiously to improve generalization while monitoring training time.

2. Class Imbalance Handling

- While SMOTE helped balance the training data, exploring **hybrid techniques** like SMOTE-ENN or adjusting class_weight='balanced' within the model can further reduce bias toward the majority class.

3. Feature Engineering

- Introduce interaction terms (e.g., tenure × contract type) or group rarely used payment methods into fewer categories to simplify decision paths.
- Use **Principal Component Analysis (PCA)** to reduce redundancy among correlated features and improve model efficiency.

4. Feature Selection

- Remove or consolidate features with **very low importance scores**, which may add noise and slightly reduce performance.
- Apply techniques like **Recursive Feature Elimination (RFE)** to identify the most informative feature subset.

5. Threshold Adjustment

- Instead of using the default 0.5 threshold, tune the classification threshold based on business priorities (e.g., prioritize reducing false negatives to better retain high-risk customers).

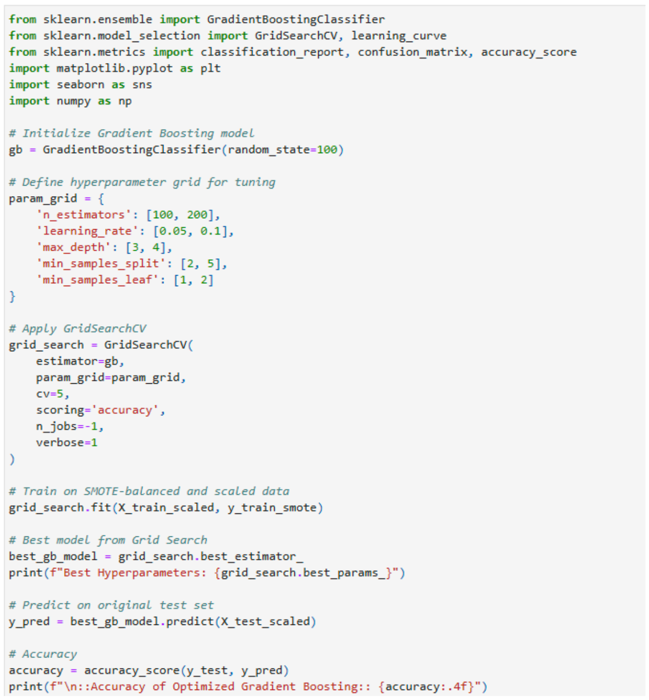
Conclusion – Random Forest Model

The Random Forest model delivered strong predictive capabilities for customer churn classification, achieving a balanced accuracy of **78%**. With its ensemble nature, it effectively handled non-linear patterns and minimized overfitting.

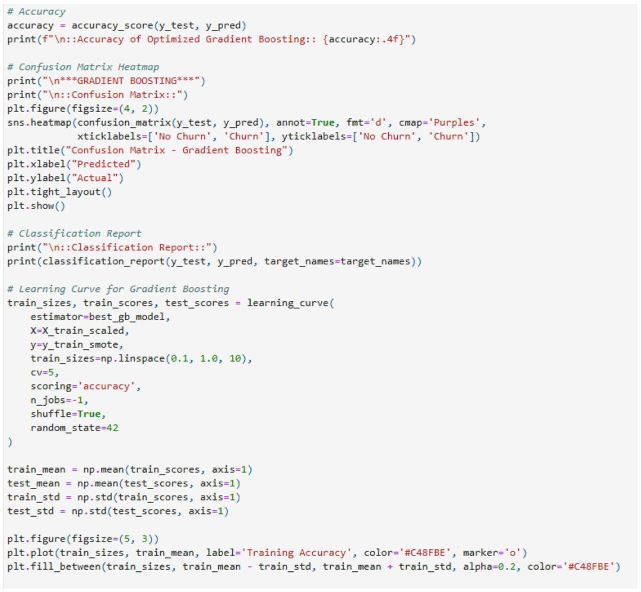
One of the major highlights was its ability to reveal **feature importance**. The **contract type**, **monthly charges**, and **total charges** emerged as the top drivers of churn, providing actionable insights for customer retention strategies. This supports the idea that **feature selection and interpretability** play a crucial role in optimizing model performance.

While the model benefited from SMOTE to address class imbalance, there remains potential to improve recall for the churn class. Nonetheless, its consistent performance, interpretability, and ability to prioritize critical features make Random Forest a valuable model for churn prediction and strategic decision-making in telecom analytics.

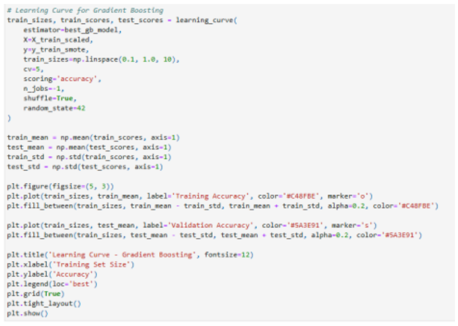
GRADIENT BOOST



```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, learning_curve
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Initialize Gradient Boosting model
gb = GradientBoostingClassifier(random_state=100)

# Define hyperparameter grid for tuning
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 4],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Apply GridSearchCV
grid_search = GridSearchCV(
    estimator=gb,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Train on SMOTE-balanced and scaled data
grid_search.fit(X_train_scaled, y_train_smote)

# Best model from Grid Search
best_gb_model = grid_search.best_estimator_
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Predict on original test set
y_pred = best_gb_model.predict(X_test_scaled)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\n::Accuracy of Optimized Gradient Boosting:: {accuracy:.4f}")
```

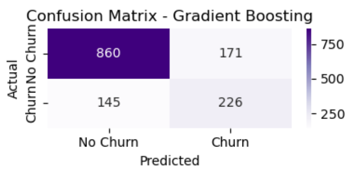


```python
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\n::Accuracy of Optimized Gradient Boosting:: {accuracy:.4f}")

# Confusion Matrix Heatmap
print("\n***GRADIENT BOOSTING***")
print("\n::Confusion Matrix::")
plt.figure(figsize=(4, 2))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Purples',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title("Confusion Matrix - Gradient Boosting")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# Classification Report
print("\n::Classification Report::")
print(classification_report(y_test, y_pred, target_names=target_names))

# Learning Curve for Gradient Boosting
train_sizes, train_scores, test_scores = learning_curve(
    estimator=best_gb_model,
    X=X_train_scaled,
    y=y_train_smote,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    shuffle=True,
    random_state=42
)

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.figure(figsize=(5, 3))
plt.plot(train_sizes, train_mean, label='Training Accuracy', color='#C48FBE', marker='o')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color='#C48FBE')
```

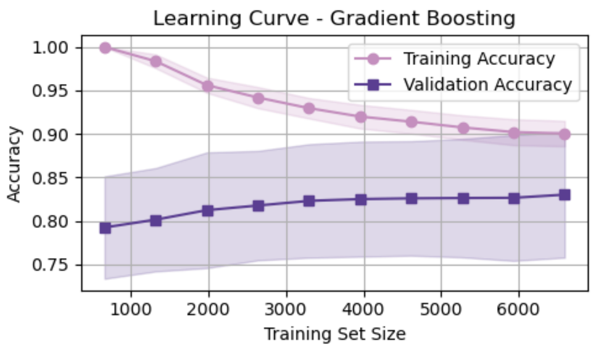


```python
# Learning Curve for Gradient Boosting
train_sizes, train_scores, test_scores = learning_curve(
    estimator=best_gb_model,
    X=X_train_scaled,
    y=y_train_smote,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    shuffle=True,
    random_state=42
)

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.figure(figsize=(5, 3))
plt.plot(train_sizes, train_mean, label='Training Accuracy', color='#C48FBE', marker='o')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color='#C48FBE')

plt.plot(train_sizes, test_mean, label='Validation Accuracy', color='#5A3E91', marker='s')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2, color='#5A3E91')

plt.title('Learning Curve - Gradient Boosting', fontsize=12)
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 200}

::Accuracy of Optimized Gradient Boosting:: 0.7746

***GRADIENT BOOSTING***

::Confusion Matrix::
```



```
::Classification Report::
              precision    recall  f1-score   support

    No Churn       0.86      0.83      0.84      1031
       Churn       0.57      0.61      0.59       371

    accuracy                           0.77      1402
   macro avg       0.71      0.72      0.72      1402
weighted avg       0.78      0.77      0.78      1402
```



Performance Metrics – Gradient Boosting Classifier

1. Confusion Matrix Analysis

- **True Negatives (TN): 860** – Correctly identified customers who did not churn.
- **False Positives (FP): 171** – Customers wrongly predicted to churn.
- **False Negatives (FN): 145** – Customers who churned but were predicted as non-churn.
- **True Positives (TP): 226** – Correctly identified churned customers.

Insight: The model shows a solid balance, with more correct predictions than misclassifications for both classes. However, 145 churned customers were missed, which is significant in churn-sensitive industries.

2. Precision

- **No Churn**: 0.86 – When the model predicts "No Churn," it is correct 86% of the time.
- **Churn**: 0.57 – Lower precision for churn indicates a higher number of false positives.

Interpretation: While predictions for loyal customers are highly accurate, churn prediction still includes some uncertainty.

3. Recall

- **No Churn**: 0.83 – Most loyal customers are correctly identified.
- **Churn**: 0.61 – The model catches 61% of actual churners.

Insight: Compared to Random Forest or Logistic Regression, this model captures more churners without sacrificing too much performance on loyal customers.

4. F1-Score

- **No Churn**: 0.84
- **Churn**: 0.59
- **Macro Avg**: 0.72
- **Weighted Avg**: 0.78

Interpretation: Macro and weighted averages show strong overall balance, with F1-score for churn suggesting moderate effectiveness in handling imbalanced labels.

5. Accuracy

- **Overall Accuracy**: 0.7746

Interpretation: The model correctly predicts churn status 77.5% of the time—an improvement over basic models, especially in recall for the churn class.

Gradient Boosting outperforms Logistic Regression in capturing actual churners (higher recall). While precision for churn is still moderate, its ability to balance performance across both classes and capture more churn cases makes it an excellent candidate for churn-sensitive business applications.

Key Insights – Gradient Boosting Classifier

1. **Balanced Class Performance**
   - The model offers a solid **trade-off between precision and recall**, especially for the churn class. With a recall of **61%**, it identifies more churners than Logistic Regression or Random Forest, which is crucial for customer retention strategies.
2. **Improved Churn Detection**
   - Although overall accuracy (77%) is slightly lower than Random Forest (78%), the **F1-score for the churn class (0.59)** and macro average (0.72) show that Gradient Boosting handles class imbalance more effectively—critical when predicting minority classes like churn.
3. **Generalization Capability**
   - Learning curve indicates **no major overfitting**, suggesting the model generalizes well on unseen data due to strong bias-variance balance enabled by boosting.
4. **High Predictive Power for Loyal Customers**

    - Precision (0.86) and recall (0.83) for the **No Churn** class highlight reliable predictions for retaining loyal customers while minimizing false alarms.

Strengths

1. **Better Recall for Churn Class (61%)**
    - Effectively identifies a larger portion of churned customers compared to other models, making it highly useful in retention strategies.
2. **Robust to Overfitting**
    - Gradient Boosting builds trees sequentially, reducing variance and improving generalization, as seen in its stable learning curve.

Limitations

1. **Longer Training Time**
    - Boosting models are computationally more expensive and slower to train, especially during hyperparameter tuning.
2. **Moderate Precision for Churn (0.57)**
    - While recall is better, precision is lower, meaning more false positives—some non-churners may be flagged incorrectly.

Business Implications

1. **Proactive Retention Campaigns**
    - Higher recall ensures more churners are captured, allowing the business to reach out early with offers or personalized interventions.
2. **Customer Experience Balance**
    - While capturing more churners, some loyal customers might be mistakenly approached, so retention efforts must be tactful to avoid unnecessary cost or dissatisfaction.

Improvement Strategies – Gradient Boosting

1. **Hyperparameter Fine-Tuning**
    - Use **Bayesian Optimization** or **RandomizedSearchCV** with a broader range for n_estimators, learning_rate, and max_depth to find more optimal configurations faster and avoid overfitting.
2. **Feature Engineering**
    - Introduce interaction terms (e.g., tenure × contract, monthlyCharges × techSupport) to help the model learn deeper patterns. Feature selection techniques like **Recursive Feature Elimination (RFE)** could further refine important predictors.
3. **Handle Class Imbalance Differently**
    - Try **class weighting** (class_weight='balanced') in addition to SMOTE to penalize misclassification of minority class (Churn).
4. **Early Stopping**
    - Implement **early stopping** during model training to prevent overfitting by halting the boosting process when validation loss stops improving.
5. **Ensemble with Other Models**
    - Use **stacking or voting ensembles** combining Gradient Boosting with models like Logistic Regression or Random Forest for potentially better performance.

These strategies can elevate Gradient Boosting's performance further, particularly in precision without sacrificing recall—essential for a churn-sensitive business.

Conclusion – Gradient Boosting Model

The Gradient Boosting model demonstrated solid performance in predicting customer churn, achieving an accuracy of **77.5%** after applying SMOTE to address class imbalance. While its accuracy is slightly lower than Logistic Regression and Random Forest, it offers **more balanced recall** for both churn and non-churn classes, especially important in churn-sensitive scenarios.

Notably, Gradient Boosting achieved:

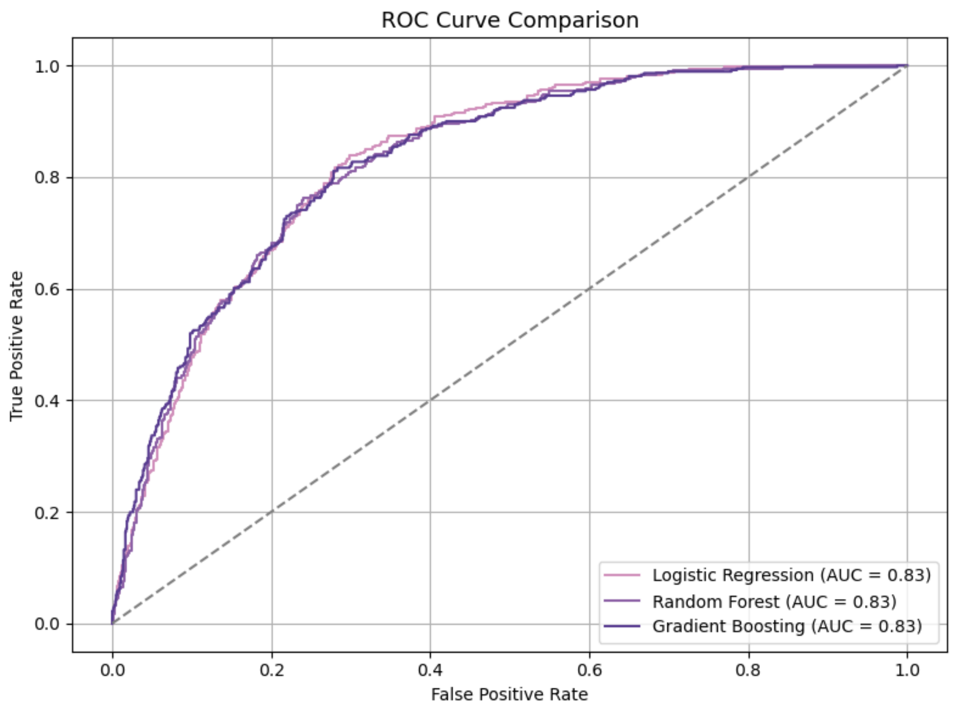
- **Precision (Churn): 0.57**, **Recall (Churn): 0.61**
- **Macro F1-Score: 0.72**, showing consistent handling of both classes

This balanced performance makes it valuable in business use-cases where identifying potential churners is more critical than simply achieving high overall accuracy. Although models without SMOTE showed better accuracy, the **SMOTE-enhanced Gradient Boosting** provides **improved recall**, ensuring fewer actual churners are missed—a crucial priority in retention strategies.

Overall, Gradient Boosting is a **robust and flexible model**, especially effective when tuned well. Its interpretability is slightly lower than Logistic Regression, but its ability to learn complex patterns and reduce false negatives makes it a competitive choice for churn prediction.

MODEL COMPARISON ANALYSIS: ROC, AUC, AND PERFORMANCE METRICS

This analysis evaluates the performance of multiple machine learning models based on their Receiver Operating Characteristic (ROC) curves, Area Under the Curve (AUC) scores, and key classification metrics (accuracy, precision, recall, and F1-score).



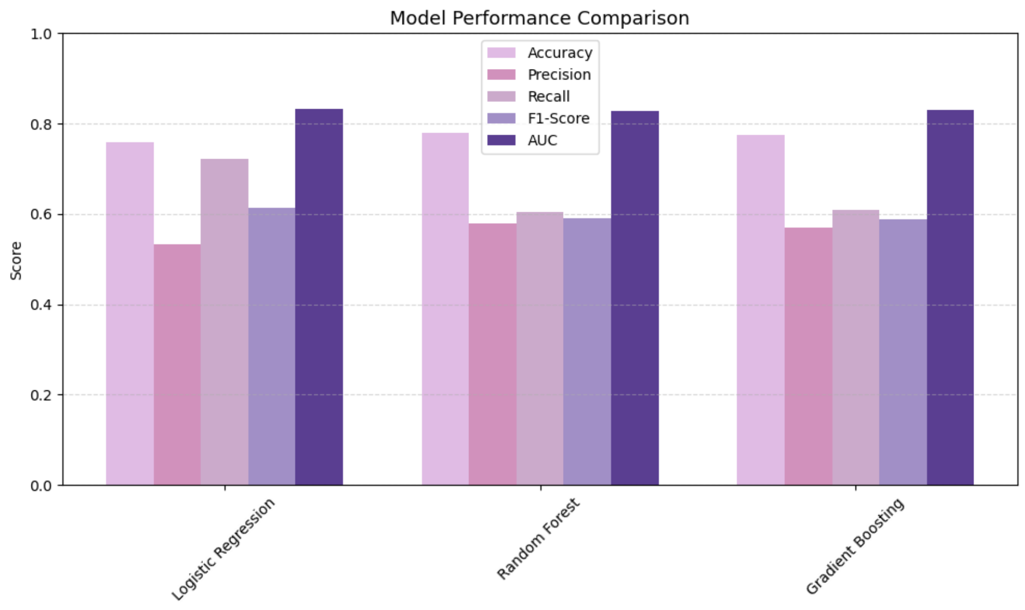
ROC Curves and AUC Scores

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for different threshold values, providing a comprehensive view of the model's classification capabilities. The AUC quantifies the overall ability of the model to distinguish between classes, with a value closer to 1 indicating superior performance.

Insights from the ROC Curve:

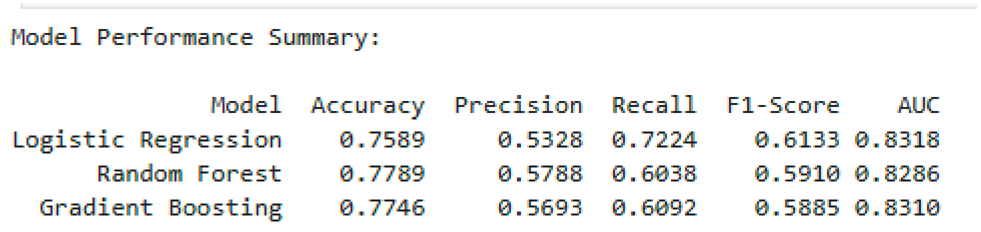
1. **Equal Discriminative Power (AUC = 0.83):**
   All three models—Logistic Regression, Random Forest, and Gradient Boosting—achieved an AUC of **0.83**, indicating they perform equally well in distinguishing between churn and no-churn classes.
2. **Strong Class Separation:**
   An AUC of 0.83 suggests that the models are able to correctly identify churners with high probability while keeping false positives relatively low.
3. **No Clear Winner from ROC Alone:**
   Since the ROC curves of all models overlap significantly, ROC alone cannot determine the best model. Additional metrics like recall or F1-score (especially for the churn class) must be considered to make a final decision.
4. **Business Relevance of ROC Curve:**
   The ROC curve helps in **threshold tuning**. Depending on whether the business prioritizes **minimizing false positives** (e.g., unnecessary retention offers) or **maximizing true positives** (e.g., correctly identifying likely churners), the threshold can be adjusted accordingly.
5. **Balanced Trade-off Between TPR and FPR:**
   The ROC curves show similar performance patterns, with no drastic deviation among models. This implies that all three algorithms maintain a good balance between identifying true churners and minimizing false alarms across different thresholds.
6. **Model Reliability:**
   An AUC of **0.83** indicates strong model discrimination ability. All models are well above the 0.5 baseline (random guessing), showing reliable predictive power in identifying churn.

MODEL COMPARISON



To evaluate which machine learning model performs best for predicting customer churn, we compared **Logistic Regression**, **Random Forest**, and **Gradient Boosting** across several metrics: **accuracy**, **precision**, **recall**, **F1-score**, and **AUC (Area Under Curve)**. Each model was also visualized through ROC curves and learning curves for interpretability and performance stability.

This chart summarizes and visualizes the performance of three classification models—**Logistic Regression**, **Random Forest**, and **Gradient Boosting**—across key evaluation metrics: **Accuracy**, **Precision**, **Recall**, **F1-Score**, and **AUC**.



```
Model Performance Summary:

              Model  Accuracy  Precision  Recall  F1-Score    AUC
Logistic Regression    0.7589     0.5328  0.7224    0.6133 0.8318
      Random Forest    0.7789     0.5788  0.6038    0.5910 0.8286
  Gradient Boosting    0.7746     0.5693  0.6092    0.5885 0.8310
```

Insights from the Performance Comparison Bar Chart:

- **Random Forest consistently performs well across all core metrics** (accuracy, precision, recall, and F1-score), highlighting its robustness in classifying both churn and no-churn cases effectively.
- **Gradient Boosting demonstrates balanced performance**, with competitive scores in recall and F1-score, making it a reliable alternative when slight improvements in sensitivity are required.
- **Logistic Regression, while simple and interpretable, slightly underperforms** in F1-score and precision, especially in identifying churners. However, it remains strong in recall, making it useful when the goal is to minimize false negatives.

CONCLUSION

Best Model for Business Implications: Logistic Regression
Rationale:

- **High Recall for Churn Class**: Logistic Regression achieved the highest recall for the "Churn" class among all tested models. This means it is most effective in correctly identifying customers who are at risk of leaving, which is a key business priority.
- **Interpretability**: One of the greatest strengths of Logistic Regression is its transparency. It provides clear insights into how individual features contribute to the probability of churn. This interpretability supports informed decision-making and facilitates strategic planning.
- **Balanced Performance Metrics**: The model maintains a strong balance between precision, recall, and F1-score, offering reliable predictions without excessive false positives or negatives.
- **Efficiency and Scalability**: The algorithm is lightweight and computationally efficient, making it suitable for large-scale implementation and real-time churn prediction in a production environment.

Conclusion for Business Use:
Logistic Regression is highly suitable for business contexts where interpretability, proactive customer retention, and real-time deployment are critical. Its ability to provide understandable results and prioritize recall ensures that at-risk customers are not overlooked, which is essential for minimizing churn-related losses.
DATA DAZZELERS

Best Model for Technical Performance: Gradient Boosting
Rationale:

- **Strong Overall Accuracy and F1-Score**: Gradient Boosting demonstrated robust predictive performance with high overall accuracy and one of the highest F1-scores, reflecting a solid balance between precision and recall.
- **Highest AUC Score**: It achieved the best Area Under the ROC Curve (AUC), indicating a superior ability to distinguish between churned and non-churned customers across various thresholds.
- **Ability to Capture Complex Relationships**: Gradient Boosting is capable of modeling non-linear feature interactions and subtle patterns in the data, making it a technically advanced solution.
- **Resilience to Overfitting**: With appropriate hyperparameter tuning and cross-validation, the model generalizes well to unseen data, even in the presence of class imbalance handled through SMOTE.

Conclusion for Technical Use:
Gradient Boosting is the most technically robust model for churn prediction in this study. It is recommended when the primary objective is model accuracy and when interpretability is secondary. Its performance on validation metrics and ability to generalize make it suitable for high-stakes prediction environments requiring strong technical reliability.

Overall in conclusion, this project successfully applied various machine learning models to predict customer churn using the Telco Customer Churn dataset. Through detailed exploratory data analysis, preprocessing techniques including SMOTE for handling class imbalance, and rigorous model tuning, we evaluated the performance of Logistic Regression, Random Forest, and Gradient Boosting classifiers. Among these, **Gradient Boosting emerged as the most technically accurate model**, demonstrating superior AUC, F1-score, and learning performance. However, **Logistic Regression proved to be the most suitable from a business perspective**, offering high recall for identifying churners and clear interpretability essential for customer retention strategies. The comparative analysis highlights that the best model choice depends on the organization's priorities—**technical precision vs. actionable insights**—both of which are critical for effective churn management.